THE TESTABILITY OF MICROPROCESSOR SYSTEMS - AN ASSESSMENT OF SIGNATURE ANALYSIS AS A METHOD OF FIELD-SERVICE.

M.J. Liebelt B.Sc., B.E.(Hons.)

A thesis submitted for the Degree of Master of Engineering Science

in

The Department of Electrical Engineering,

The University of Adelaide.

September 1981.

This thesis embodies the results of supervised project work making up all of the work for the degree.

TABLE OF CONTENTS

Page

Page

SUMMARY

Digital electronic systems must be tested many times throughout their useful lives, from the beginning of the manufacturing process until the time at which they are removed from service. Each of these testing processes can be classified into one of two broad categories: high-volume testing, typically performed by Automatic Test Equipment (ATE) during, and at the end of, the manufacturing process; and low-volume testing, typified by the repair of a single system which has failed in the field (that is, field service).

The advent of Large Scale Integration (LSI) Has had a profound effect in both of these areas of testing. The complexity and architectural characteristics of LSI devices, and microprocessors in particular, have created problems which have necessitated the development of new techniques for both automatic testing and field service. While new developments in ATE have been numerous, a result of vigorous activity in the ATE marketplace, there have been relatively few developments in the area of field service.

One field service technique which has been developed and is widely considered to be more promising than any other method is signature analysis (SA). Many advantages have been claimed for SA, but there has been little documented verification, so a trial implementation of the technique was performed on a small microprocessor system (the Intel SDK-85) with a view to assessing its effectiveness as a field service method. This revealed that a number of factors might limit the effectiveness of SA in practice. The most serious problem appeared to be that there was no accepted approach to developing thorough tests for individual LSI devices as part of the SA procedure.

As a means of overcoming this deficiency the implementation of systematic "functional" test routines in the context of SA was investigated. Self-test routines were developed for the 8085 microprocessor and the 8279 keyboard/display controller in the SDK-85. However, certain architectural properties of the devices and the lack of detailed documentation of their internal workings limited the extent to which the tests could be developed systematically.

It was concluded that the ability to test LSI devices and, in the future, VLSI devices, could only be ensured if some provision is made in the design of each device to make it easily testable - that is, if the principle of design for testability is applied at the chip level. This is seen as being essential if digital systems of the future are to be effectively tested in the field, whether by signature analysis or any other method.

DECLARATION

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university, and to the best of my knowledge and belief, contains no material previously published or written by any other person, except where due reference is made in the text.

M. J. LIEBELT.

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Dr. D.A. Pucknell, for his advice, help and encouragement during the course of this project.

Figures 3.1, 3.3, 5.1 and 6.1 and Appendix A are reprinted by permission of Intel Corporation.

CHAPTER I.

INTRODUCTION

1.1 Test Environments for Digital Systems

In the lifetime of any electronic device or system, from the commencement of manufacture to obsolescence, the unit is, in general, tested many times, in several different environments[1]. A digital integrated circuit, for example, may be tested as an isolated chip before packaging, after packaging, during burn-in (if any), prior to insertion into a printed circuit board, as part of the completed printed circuit board, as part of a complete system containing the printed circuit board and, variously, when the system is tested as a result of failure in the field.

It is important that at each stage of testing, the testing process is performed thoroughly, but also efficiently. Only then can the manufacture and maintenance of a reliable system be carried out economically. Inability to satisfactorily test a device or system at any stage of manufacture may mean that faulty devices or systems will pass the test, only to go through to a later stage of manufacture, or to application in the field, where the fault may be detected and will be much more expensive to repair. The property of test economics that faults become progressively more expensive to detect and repair at later stages of manufacture has become known as the "ten times rule"[2][3] because it is estimated that testing costs rise by a factor of ten at each successive stage.

Each of the testing phases mentioned above can be broadly classified into one of two categories: high-volume testing or low-volume testing. High-volume testing is typically carried out by the manu-

facturer of devices or systems who must test a relatively large number of identical units during or after manufacture. In such cases the time taken to test a single unit is multiplied by the production volume and may be a limiting factor on the overall production rate[4]. Because large numbers are involved, and small improvements in unit test times can significantly improve production rates, high-volume testing almost invariably requires the use of automatic test equipment (ATE) which, although expensive, is cost-effective in testing large numbers of identical units[3][5].

Low-volume testing is typified by the process of field service - the repair of a system which has failed in the field. In this case, the system is usually tested by one technician who must systematically test each section of the system, using general purpose instruements, until the fault is isolated and repaired. Because only one system of a given type is serviced at a time, the expense of ATE is clearly not justifiable (quite apart from the fact that most ATE is large and not portable). The process of manually testing a system is inherently a very slow, and therefore expensive one and for that reason field service is at the top of the "ten times rule" as the most expensive testing phase.

It is also important to note that in the field service environment the objective is usually to isolate and repair a fault in the system under test, whereas in many cases (notably in the testing of individual devices) ATE is used simply to detect faults. As faulty devices are discarded, the matter of fault isolation - of determining the nature of the fault - is of little importance. High and low volume testing therefore often differ not only in economics and method, but also in objective.

Over the past ten years increasing difficulty has been experienced in all stages of testing digital devices and systems as a result of the introduction of large scale integration (LSI). To appreciate and place into context the problems which the advent of LSI has created, it is first necessary to briefly review testing practices for digital systems prior to the introduction of LSI. Because of the vastly different approaches used in high and low volume testing, the ATE and field service environments will be considered separately.

1.2 Testing Practice Before the Introduction of LSI

1.2.1 Characteristics of digital systems before LSI

Before the introduction and acceptance of large scale integration in the early 1970's, digital systems were most commonly implemented in small scale integration (SSI) and medium scale integration (MSI) integrated circuits. Devices in the SSI family are characterised by having fewer than one hundred transistors (approximately) integrated onto each chip, while MSI devices contain one thousand transistors or fewer, this being the limit of technology in the late 1960's. Logic functions implemented in SSI and MSI devices were therefore limited by the number of transistors available on each chip, and constrained by the need to package useful, general purpose logic elements. Consequently, functions packaged in SSI/MSI tended to be simple - combinational logic gates, flip-flops, counters, multiplexers and so on. Almost invariably the operation of these devices chould be described by a simple truth table or characteristic table.

Sequential devices of the SSI/MSI families possessed a property which might be called "low sequential depth" - all sequential logic elements contained in the devices were not deeply buried, but had outputs taken either directly or through simple combinational logic to device output pins. Consequently, the operation of each sequential logic element in a device could be easily observed without requiring that logic values be propagated through multiple levels of sequential logic. This property results from the fact that more sequential functions which could have been implemented with less than one thousand transistors were not general enough to be commercially worthwhile[6].

This functional simplicity of SSI/MSI devices meant that individual devices were easy to test. To completely [1] test a device it was sufficient to simply verify that it behaved as its truth table specified, while if the internal logic circuit of the device was known (and this was usually the case) it could often be even more simply tested for the presence of a restricted set of faults. The success of ATE and field service practices in testing SSI/MSI based systems were a result of this property of SSI/MSI devices.

1.2.2 High-volume testing of SSI/MSI logic

Automatic test equipment for SSI/MSI devices and systems was typically based on a minicomputer which would apply a test stimulus to the unit under test (UUT) and monitor its response[7]. The test stimulus was a pre-determined sequence of logic values (often referred to as a "test set") which was applied to the device or board inputs to stimulate the internal logic. It is the different means of gen-

1 *A "complete" test in this context refers to a complete functional test, rather than parametric test* [7].

erating the test set and of analysing the UUT response which distinguish the various ATE methods employed.

While individual devices were quite easily tested, it was much more difficult to test and isolate faults on boards containing many devices. ATE employed one of two techniques to stimulate and test logic on a board. The first of these, functional testing, involved the straightforward application of the test stimulus to the available inputs (usually at an edge connector) of the UUT, and observation of the response of the UUT at its available outputs. The test set would be designed to exercise the board as a whole in such a way that any faults present in the internal logic could be identified from the response observed at the board outputs.

The second technique, known as in-circuit testing, avoided the difficulties of generating a test-set for the board as a whole by testing devices on the board individually. The input and output pins of each device was accessed using either an integrated circuit (IC) clip or a bed-of-nails fixture which allowed access to any node on the board, rather than just the input and output connections. The generation of test sets for in-circuit testing of boards therefore essentially consisted of developing tests for individual SSI/MSI devices which, as we have seen, was relatively simple. However this ease of developing test sets was achieved at the expense of effectiveness in fault detection. In-circuit testers typically detected faults in approximately 80% of faulty boards tested, whereas functional testers achieved fault detection rates of 95% or more[3][5]. The greater effort (and expense) of generating test sets for functional testing was therefore often worthwhile.

Two general methods of generating test sets for functional testing were employed. For certain types of combinational systems, methods were developed for analytically deriving minimal test sets guaranteed to detect certain faults. Two of the better known methods, the D-Algorithm and the Boolean Difference technique are described by several authors[7][8][9]. The methods were initially applied to detect stuck-at (s-a) faults[7][8][10] in irredundant combinational logic networks, but there have been many developments of these and other analytic techniques, extending fault coverage to other fault classes and more complex systems. The literature also contains many papers on the development of minimal test sets for certain classes of combinational logic networks and on the design of combinational networks to minimise test sets[7][8][11]. However little success has been achieved along similar lines for sequential networks[9].

The second method of test set generation used for functional testing was pseudo-random generation, often used for systems to which analytic methods could not be applied. This involved the generation, by the ATE, of pseudo-random test vectors which were applied as stimulus to the UUT. The existence of any faults in the UUT would be detected by comparing its response (possibly in real time) to that of a unit which was known to be good ("comparison testing"[7][12]) or to that of a computer simulation model of the system[5][7]. In either case the response data for a fault free system could be stored on-line by the ATE for comparison with the response of the UUT ("stored response testing" or "stored truth table testing"[7][12]).

With a computer simulation model of the system to be tested it was possible to introduce a known fault into the model and record the

response date for the system with that fault. The procedure could be repeated for many different faults, thereby constructing a "fault dictionary" containing response data for the system under each of the fault conditions simulated[5][7]. If a fault were to be detected in the UUT, its actual response could be compared with the contents of the fault dictionary to determine the likely nature of the fault. Simulators also allowed a degree of interaction, enabling the test programmer to manually modify the test set so that it would detect certain faults in the system which might otherwise go undetected[7][9][13]. Similarly, a simulator could be used to assess the fault coverage of a given test set.

ATE manufacturers developed various software aids to increase the flexibility and power of their equipment. These aids included simulators and automatic test pattern generators, such as that described by Thurman[14], to allow on-line generation and development of test patterns. Various forms of diagnostic software were also developed for ATE, including "guided probe diagnostics"[7][13] which allowed the ATE, with the aid of a manually operated logic probe, to examine nodes interval to a board and track down a faulty device on the board.

The inevitable growth in the size and complexity of SSI/MSI based systems made fault detection and isolation in these systems by ATE increasingly difficult. This gave rise to a discussion in the literature of the principle of "design for testability", particularly as applied to the logical and physical design of SSI/MSI boards so that they could be easily tested by ATE[11][15][16][17][18][19]. The common theme in these papers was the specification of design guidelines which,

if followed in design of a system would ensure that ATE could quickly and effectively test the system.

Typical guidelines were:

(i) provide reset lines, accessible by the tester, for all sequential logic;

(ii) provide means of access by the tester to the system clock (for synchronization);

(iii) provide an adequate number of test points, to allow internal logic elements to be easily controlled and observed (that is, to improve system controllability and observability);

(iv) avoid the use of monostables and asynchronous logic elements;

(v) avoid the use of wire-AND and wire-OR logic;

(vi) avoid the use of redundant logic.

In most cases, the implementation of these guidelines involved an increase in the design and manufacturing costs of the system, but it was considered that this would be more than offset by savings in testing costs over the life of the system[5][11][20].

It is important to notice that the success with which ATE was applied to SSI/MSI systems was due to the functional simplicity of SSI/MSI devices. It was only because the function of individual combinational devices could be described by a simple mathematical equation that the D-Algorithm and Boolean Difference technique could be used to derive test sets for networks composed of these devices. Similarly, simulators for SSI/MSI networks could only be developed because the behaviour of individual devices could be described by simple truth tables. However, it must also be noted that as the size of SSI/MSI networks increases, both the complexity of the analytic calculations

and the computer simulation time increase dramatically and there is a practical limit upon the size of SSI/MSI networks which can be dealt with using these methods[8].

1.2.3 Low-volume testing of SSI/MSI logic

Low volume testing (that is, field service) of SSI/MSI based systems was, as discussed above, a slow process usually performed by one technician using general purpose instruments. The usual approach to isolating a fault during field service was based on the observation of the behaviour of individual devices. Unlike automatic testing in which access to individual devices on a board is limited, the field service technician can gain access to, and observe, any device, at will.

Because SSI/MSI devices were characterized by simple behaviour, it was a relatively simple matter for the technician to observe the inputs and outputs of a device, and to decide whether it was functioning correctly or not. The means of observation varied from device to device, but in all cases the principle applied that because the devices had simple truth tables they were individually easy to test.

The size of the system being tested was, as for automatic testing, an important consideration. However, because individual devices could be observed directly, the size of the system affected not the ability to test the system overall, but the ease with which faults could be diagnosed from observed system behaviour. Individual devices could be tested just as effectively in a large system as in a small one (although adherence to the "design for testability" guidelines for ATE could make the process a little easier), but a technician would be expected to find greater difficulty in deciding which device to test in a large system.

For some systems long periods of down-time, while field service procedures such as described above were performed, are not tolerable. In such cases the practice of "board swapping" was, and still is, used to repair the system. Board swapping involves the immediate replacement of any board or boards in a system which are suspected of containing faults, instead of attempting to isolate and repair the fault on the board in the field. The suspect boards are returned to a central repair site where they are tested (possibly by ATE) and repaired before being returned to the field to be swapped for other suspect boards. Board swapping is an expensive repair method, because large stocks of spare boards must be held in the field[21], but it is cost-effective in large systems for which down-time is expensive. It also serves as a final solution for other systems at times when the traditional field service approach of component-level repair in the field, fails.

Despite the relative expensiveness of field service, it is apparent that the problems experienced dealing with SSI/MSI based systems were few, even as system complexity increased. This is confirmed by the observation that the literature contains virtually no discussion of any problems of field servicing SSI/MSI systems. This may be contrasted with the increasing concern expressed in the mid 1970's about the testability of complex SSI/MSI based systems by ATE, as discussed above.

1.3 The Effects of LSI on Testing Practice

1.3.1 The characteristics of LSI

The introduction of the 4004 microprocessor in 1971 by the Intel Corporation heralded the start of the era of LSI[22][23]. With 2250 transistors on one chip[24], the 4004 was the first of a family of devices which currently contain up to 70000 transistors on one chip[23].

The introduction of LSI increased the number of logic gates integrated in one device by a factor of ten to one hundred over SSI and MSI devices. While this simple fact created problems in testing LSI devices and systems, other characteristics of LSI (obviously a product of the higher gate count per device) have more directly affected testing practices. For some time before the introduction of LSI, device designers had found that it was no longer satisfactory to place more and more complex logic building blocks on a single chip, as had been the custom with SSI and MSI. Building blocks which were sufficiently general in function did not use the full capabilities of the integration technology[6]. Consequently, the first significant use of LSI was to produce completely different types of devices - in particular memories and, with the 4004, microprocessors.

LSI devices were therefore, from the outset, architecturally very different to their predecessors and did not possess the functional simplicity of SSI and MSI devices. With the exception of memory devices which, because of their very regular structure, fall into a class of their own, the new LSI devices were highly sequential. They contained very large numbers of sequential logic elements which were

no longer easily observable, but were "buried" within the device[25]. Not only did LSI devices have very many more states than their SSI/MSI predecessors, (because of the greater number of sequential logic elements they contained), but the accessibility of internal logic elements was much reduced, which made the devices immeasurably harder to test[26].

The intuitive concept of accessibility of logic elements in a network from its test points (the device pins in the case of LSI devices) has been put on a more formal basis by Goldstein[27]. He presents a quantitative measure of the ease of testing a logic network in terms of combinational and sequential "controllability and observability numbers". These numbers are an increasing function of the number of combinational and sequential logic elements between a given logic element and the primary inputs and outputs of the network. They indicate the ease with which the inputs of such a logic element can be controlled ("controllability") and the outputs observed ("observability") from the network inputs and outputs, through surrounding logic. Goldstein proposes that the higher the controllability and observability numbers of a network are, the harder it is to test, because internal logic is harder to exercise. The greater logic depth in LSI devices than in SSI/MSI devices means that LSI devices would have higher controllability and observability numbers and would, therefore, exhibit much poorer testability.

From a different point of view, the reduced accessability to internal logic in LSI devices can be viewed as a reduction in the test point-to-gate ratio over SSI/MSI systems of equivalent complexity. It is accepted that on a SSI/MSI based board, there should be one to two test points for every package on the board[16]. If it is assumed

that gate integration levels for LSI devices are one hundred times greater than typical SSI/MSI devices, by this principle a typical LSI device should have 100 to 200 test points, whereas few devices have more than forty pins.

Apart from the significant internal architectural differences between LSI and SSI/MSI devices, and the consequent problems of device testability, LSI devices (and the microprocessor in particular) differ from SSI/MSI devices in that the architecture of the device to a large extent defines the architecture of the system of which it is to be part. The microprocessor, unlike a flip-flop, multiplexer or counter, is designed in such a way that it can only be used in, or designed into, a system in one way, with a specific set of peripheral components (ROM, RAM, etc.). Thus the design of the microprocessor itself largely dictates that most LSI-based systems must be bus-structured systems, in which the busses are very often multiplexed and bidirectional. SSI/MSI based systems, on the other hand, have widely varying architectures and no similar generalisations can be made about their design or structure.

Finally, new failure modes in LSI devices have created difficulties in testing. The much smaller device geometries and line widths in LSI devices have increased the susceptibility of devices to dynamic (or "soft") failures and pattern sensitivity[28][29][30][31]. It is therefore no longer sufficient to test for stuck-at and bridging faults when testing LSI devices. New fault classes exist and test procedures must be designed to detect them.

1.3.2 The effects of LSI on automatic testing

The effects which the introduction of LSI has had on the field of automatic testing arise out of three of the characteristics of LSI discussed above. They are device complexity, susceptibility to dynamic failures and LSI system architecture.

The simple fact that LSI devices contain many thousands of transistors, forming a logic network of thousands of gates, means that the analytic methods employed to generate test sets for SSI/MSI based systems may be impractical for LSI devices and systems. Williams and Parker[10] have stated that the ability to generate tests automatically for general sequential networks decreases to unacceptable levels as the networks grow to the region of 1000 to 2000 gates. A typical LSI device now contains more than 1000 gates and therefore falls into this category. Lyman[32] reports that automatic test generators for LSI based systems have been found wanting, requiring expensive manual assistance, while Breuer and Friedman[33] state that automatic test generators are "grossly ineffective" for VLSI devices such as microprocessors. Even if the problems of computational feasibility were overcome, the ability to automatically generate test sets for LSI devices would be limited by the facts that device manufacturers seldom supply gate-level equivalent circuits for their devices[10][30][34][35][36] and that methods of generating tests for large sequential networks are not well developed in any case[37].

Given that these problems of automatically generating tests for LSI devices and systems exist, other approaches must be considered. However, it is apparent that any form of "complete" test is impractical. Jefferies[38] has estimated that a typical 8 bit microprocessor could

be tested "exhaustively", by forcing it to execute all possible instructions, with all possible data values, in all possible sequences, using 2^{608} test vectors which would take an impossibly long time. Bilton[35] estimates that to test a microprocessor fully, checking for pattern sensitivities, would take over 10^{23000} years! It is clear, then, that any practical test for LSI devices must be less thorough than tests which could be performed on SSI/MSI based boards, and must be developed on the basis of less complete, less rigid fault models.

The use of simulators to develop test sets for LSI based systems has been found to be difficult, for reasons similar to those behind the problems of automatic test generation. LSI device complexity, and the lack of gate level information about the devices make the modelling of LSI devices (and hence, of LSI-based systems) at the gate level virtually impossible[39][40][41]. Consequently, interactive manual development and evaluation of test sets, and the construction of fault dictionaries are difficult to achieve for LSI.

Whichever technique of generating test sets for LSI devices and systems may be used, the test sets and response sequences are invariably very long, because of device complexity. This means that for stored-pattern testing, ATE required a very large on-line storage capacity[4][42][43][44]. Furthermore, test times for individual devices and boards can become unacceptably large unless the ATE is very fast[4]. The effective testing of LSI in a production environment therefore requires the use of ATE which is fast and has a large on-line storage capacity and which is, therefore, very expensive.

Memory devices are somewhat exceptional LSI devices in that their complexity does not present such serious problems for ATE. Because the logic in memory devices is arranged in a regular structure, it is possible, knowing that structure and likely failure modes at the gate level, to develop test algorithms which will detect the most important faults, including pattern sensitivities and dynamic faults. Chen et al.[45] describe a procedure for generating a RAM test based on known failure modes. There exists a wide range of "standard" and ad hoc RAM test algorithms, which are variously described in the literature[7][46][47][48][49] although, as noted by Hayes and McCluskey[31] the underlying models on which many of these algorithms are based, are unclear. Purkis[26] remarks that "the success that has been achieved with efficient memory testing is largely due to clever software", whereas LSI devices which are much less regularly structured do not lend themselves to being tested by "clever software".

The susceptibility of LSI devices to dynamic failures and the consequent need to test them at their rated clock speed[50], reinforces the need (discussed above) for very fast ATE. With some LSI devices now working at clock frequencies greater than 10MHz, ATE must supply test vectors and either analyse or store responses at speeds approaching that figure, if the devices are to be tested in real time. Although the need to test in real time is not universally accepted[32], several manufacturers are committed to the principle, and the motives of those ATE manufacturers who deny that it is necessary must be suspected.

The use of tri-state bus structures in LSI-based systems has

created the need for special interface "pin" electronics in ATE. Together with the need to test systems at high speed, the use of bi-directional busses means that the input/output test pins of the ATE must be capable of changing direction at high speed[1][43][51]. This problem was almost unknown in SSI/MSI systems testing because it was generally true that the pins of SSI/MSI devices were unidirectional.

The use of bus structured architectures has also meant that in LSI based systems there are generally more devices connected to bus nodes than would normally be connected to a single node of an SSI/MSI based system, and, furthermore, that most of those devices are capable of driving the node. Therefore, if a stuck-at fault is detected on a bus node, some form of current tracing technique must be used to isolate the device which is the source of the fault[1]. Various current tracing tools are available for ATE[30][43][52], but they invariably require that a probe is manually moved over the board, possibly under the guidance of software running in the ATE. This is a cumbersome, slow, and therefore very expensive process.

1.3.3 The effects of LSI on field service

It was noted earlier in this chapter that the process of field servicing SSI/MSI based systems relied on the observation of the behaviour of individual devices and, because device behaviour was quite simple, presented few problems. The problems experienced in field servicing LSI based systems are not a result of device complexity as such, but arise because the perceived behaviour of LSI devices is, in contrast to SSI/MSI devices, very complex.

To view the highly sequential nature of LSI devices in another light, it is evident that the behaviour or activity of an LSI device

at any given time can depend on inputs applied to the device millions of clock cycles earlier. For example, the behaviour of a microprocessor at any time can depend on data stored in its internal registers at some arbitrary earlier time. This means that it is not possible to determine whether the microprocessor is functioning correctly if it is only observed over a short period of time. Thus the approach used to field-service SSI/MSI based systems is generally not suitable for LSI based systems.

Given that faults in an LSI based system cannot, in general, be diagnosed through the observation of individual devices, it is logical to consider the observation of overall system behaviour as an approach to diagnosing the fault. Several architectural properties of LSI (in particular, microprocessor) based systems make this approach difficult[53].

The principal path of communication in a microprocessor system is the data bus, which is multiplexed and, in general, has several devices which could drive it at various times. Consequently, it is difficult to interpret the data present on the data bus at a given time and, therefore, to determine what the system is doing. Waveforms present on system busses also tend to be non-repetitive, which makes the observation of data with conventional test instruments (such as an oscilloscope) extremely difficult[54].

Bus structured system architecture creates the same problems during field service in isolating stuck-at and bridging faults on system busses as during automatic testing. Here again a current tracing device must be used to locate the source of the fault. However, this is a less serious problem in the field service enviroment than in the

ATE environment because field service is itself a slow, manual process. The time taken to perform current tracing therefore does not significantly increase overall system test time during field service whereas it certainly does during automatic testing.

The architecture of a microprocessor system is such that the activity of the various devices in the system is very dependent upon the behaviour of other devices in the system. This, again, is in contrast with SSI/MSI based systems in which it is generally possible to determine whether a device is functioning correctly or not, irrespective of the behaviour of other devices in the system. The close interdependence of devices in microprocessor systems is a result of the multiple feedback paths (notably the data bus) which exist in such systems[55].

It has long been recognised that the presence of feedback loops in SSI/MSI based systems makes the systems difficult to test[19] and this is equally true for LSI based systems. If a system contains feedback loops, a fault in one device will create errors which may propagate around the loops and affect the behaviour of devices which provide inputs to the faulty device. Thus, a short time after the fault first occurs, several devices will be behaving incorrectly, with no obvious indication of which is the faulty one. If the fault is to be isolated, the device which misbehaves first must be found and this will, in general, be a tedious and time consuming task because it requires that system activity be followed in detail.

The final, and perhaps most serious, problem which microprocessor based systems present during field service is that they are software driven. To know what should be happening within a system and what

each device should be doing at a particular time, and to follow activity in the system, the field service technician must be able to understand and follow the system software, which is generally an extremely difficult task. Only if the technician is very familiar with both the hardware and software of a system is he in a position to deal with all types of fault in the system. This means, in practice, that before attempting to debug a system the technician must spend some time becoming familiar with the system, which makes the field service process even slower and more expensive.

1.4 Summary

It may be concluded from the preceding discussion that the introduction of LSI has seriously affected the ability to test digital devices and systems at all levels. Test techniques developed for SSI and MSI based systems have proven to be inadequate for LSI based systems, although it is apparent that automatic testing methods have proven less deficient than field service methods. The problems presented to automatic testing have, more than anything else, created a demand for very fast, powerful and expensive ATE. For field servicing, however, a completely new approach is clearly necessary.

The test and instrumentation industry has been far from static in the past ten years and, pressed by the need to contain testing costs at all levels, has refined existing instruments and techniques, and developed new ones. Some of these developments in both ATE and field service will be discussed in Chapter II, together with those problems which still exist in spite of the developments.

CHAPTER II

RECENT DEVELOPMENTS AND CURRENT TESTING PRACTICES

2.1 The Response to LSI

A variety of techniques to deal with the problems described in Chapter I have been developed since the introduction of LSI. Although the problems encountered in automatic testing and field service arise from the same basic properties of LSI devices, because of the different methods used and economic considerations involved in the two environments, those problems are quite different and have prompted very different solutions. For this reason it will once again be convenient to consider the automatic test and field service environments separately.

In this chapter, and throughout the remaining chapters, the microprocessor and related LSI devices will be considered in particular. They are the most widely used LSI devices, are among the most complex, and certainly have received the most attention in the literature. Consideration of the problems and methods of testing this particular family of LSI devices will cover most of the problems and methods of testing LSI devices in general.

2.2 Developments in Automatic Testing

2.2.1 Practical developments

The traditional techniques for automatic testing of SSI/MSI based systems have, as discussed in Chapter I, proven to be inadequate for dealing with LSI based systems. However, most of the developments in

the practice of automatic testing to enable LSI devices and systems to be effectively tested have been refinements or adaptations of those traditional techniques.

One method which was used initially to test boards containing an LSI device and some SSI/MSI logic was to place the LSI device in a socket on the board, to be removed from the board during testing[16][17][18][56]. The board would then be tested by traditional methods, while the LSI device would be tested separately. While this approach effectively avoided the problems of testing LSI-laden boards, the need to physically remove and replace the LSI device increased both handling costs and test times. It therefore only served as a stop-gap method[51]. As the use of LSI devices became more widespread, it became common to find several on one board, and the costs of placing several devices in sockets, and of removing them during testing, made the approach unattractive. In many current systems if all LSI devices were to be removed for testing, there would be little left on the board to test.

By about 1977 methods of modelling LSI devices for the purpose of simulation had been developed. As discussed in Chapter I, it is practically impossible to model LSI devices at the gate level. However, several authors have reported the development of "block box" or "high level functional" models for individual LSI devices[39][51] or groups of devices[57], which could be incorporated into a simulator to generate test sets for LSI-based boards as a whole (with the LSI devices in place). These high level models simulate the behaviour of the device as seen by the rest of the circuit at the device pins. No attempt is made to accurately model logic internal to the device, or to develop detailed tests for the LSI device[39] which would

typically be tested separately, prior to insertion into the board[57].

Although simulators continue to be used in various forms, several problems exist. The generation of device models, even at the "black box" level, is a slow process which must be performed manually and is therefore expensive. It has been found that new LSI devices are released at a greater rate than simulation models can be developed[34][52]. Furthermore, the documentation supplied for LSI devices is often not sufficient to allow accurate models to be developed, even at the "black box" level[1][30][32]. For example, it may be necessary, simulating a microprocessor system under fault conditions, to model the behaviour of the microprocessor itself in response to an "illegal" instruction code. This behaviour is rarely documented by device manufacturers [1], so the behaviour of the device under these conditions must be regarded as unpredictable, and variable from one chip to another. This means that no model of the device could be both complete and accurate.

Teradyne have very recently announced a software tool called TML[58][59] which accepts a functional description of an LSI device and compiles a gate level equivalent circuit for the device, which may be used by the LASAR simulator to simulate a complete system containing the device. While this does overcome the problems of manual generation of device models, and allows for the generation

1 *In an extreme example of this lack of documentation, Intel chose not to announce several working and useful instructions designed into the 8085 microprocessor* [22].

of tests for the LSI device based on the gate level equivalent circuit, the problem of incomplete documentation of the functional behaviour of devices remains. There also appears to be no guarantee that tests developed to detect classical (stuck-at) faults in the gate-level equivalent circuit will detect faults which are likely to physically occur in the device[60][61][62]. In other words, the equivalent circuit may not accurately represent the physical implementation of the device.

Comparison testing - particularly in real time, with a fixed reference device - has proven to be one of the most widely used techniques for testing LSI devices and systems. It is particularly popular for testing individual devices (for which diagnosis of any observed fault is usually not required) by device manufacturers, and at incoming inspection by board manufacturers[29][38][44].

Comparison testing, of course, is not without its problems. The use of a fixed reference device limits the flexibility of ATE[4], while difficulties exist in comparing the response of two devices to undefined instruction codes[38]. The main problem with comparison testing, however, is, as discussed in Chapter I, the generation of the stimulus test set. The simplest means of stimulating the unit under test (UUT) and the reference device, is by applying a pseudo-random data sequence to their inputs, which can be done in real time and requires no storage of stimulus data. However, it is difficult to assess the effectiveness of pseudo-random test sets in testing LSI devices[61] and McLeod[63] reports that the method is not particularly successful because it is difficult to propagate pseudo-random data through members of the current generation of programmable LSI devices. Manual generation of stored test sets is an alternative, but an

expensive one[44].

Algorithmic pattern generation (A.P.G.) is another real time test vector generation method[29][42][64], in which the test set is a sequence of microprocessor instructions generated on-line by a microprogrammed sequencer, rather than a pseudo-random data sequence. It is usually used in conjunction with a technique known as "module senorialisation" which allows suitable test instruction sequences to be derived from the architecture of the device. The major advantage of A.P.G. is that, like pseudo-random testing, it removes the need to store large quantities of stimulus data and is, therefore, relatively inexpensive.

Bisset[65] and Bluestone[12] describe two similar methods of implementing comparison testing in which the device under test operates in its "natural environment" - a system of the type in which it is typically used. Thus the stimulus is provided from within the system, so the test set for the device is, in effect, generated by programming the system to perform typical operations.

In-circuit testing has become increasingly popular as a means of testing LSI-based boards[66][67][68] because it allows the very complex LSI devices on the boards to be tested essentially in isolation[69]. The problem here, as in the testing of isolated devices, is the generation of suitable test sets for the devices. Therefore, while in-circuit testing reduces the problems of testing LSI based boards, it does not solve them.

There is a wide range of commercial ATE currently available and in development, which use a variety of the methods described above,

and some others[44][52][66][67][70][71][72]. These systems vary in their intended application, speed, software support, test effectiveness and, of course, price. While the literature contains frequent announcements of new ATE, with new capabilities, the full details of the test methods used by new equipment are very rarely divulged - a consequence of the intense competition between ATE manufacturers. It appears that the current generation of ATE is coping satisfactorily with most LSI devices, but concern is being expressed about the ability of ATE to deal with the upcoming range of very large scale integration (VLSI) devices[32][73]. In any case, it is clear that no single test method is in itself a complete solution and, notwithstanding the ability of the different methods to variously deal with LSI at present, problems still do exist.

Faced with these problems, ATE manufacturers (and others) have begun to demand that device manufacturers not only provide more complete documentation for their products, but also design them so that they can be more easily tested[32][73][74][75]. Some authors have gone as far as to suggest that the growth in the use of LSI and VLSI will be limited by problems of testability unless devices are designed with testability in mind[9][10]. A number of system manufacturers have recently provided testability features in devices being built into their systems, the best known example being IBM's use of Level Sensitive Scan Design (LSSD) in chip design[10][74][76][77]. There has been some preliminary independent research into other methods of enhancing device testability by improving controllability and observability of internal logic[78]. However, Lyman[32] points out that until the concept of on-chip testability aids is accepted by independent device manufacturers, it is unlikely to have a significant impact on the general testing scene.

2.2.2 Theoretical developments

It is significant that the test techniques being used in practice, and outlined above, have little basis in theory and involve little analytic or automatic test pattern generation. In fact, there has been very little theoretical work published which is at all directly relevant to testing LSI in practice. Much of the ongoing theoretical research on testing digital systems is still concerned with purely combinational networks, with a little work on sequential networks at the gate level. Bennetts[37] remarks upon the lack of work on sequential systems at a practical level and identifies some of the areas in which research is needed to meet the problems of testing LSI.

One of the few developments in the area of analytic test generation, at other than the gate level, is presented by Breuer and Friedman[33]. They describe a method, analogous to the D-Algorithm, of generating tests for "high level functional primitives" - that is, register level components such as counters and shift-registers. It is clearly not directly applicable to LSI devices, but does address test generation at a higher level than earlier analytic methods and is a step in the right direction.

Thatte and Abraham[36] describe a method of generating test instruction sequences for microprocessors, based on a register-transfer level (RTL) model of the microprocessor which can be derived from its instruction set description. The method is not fully analytic, in the sense that the algorithms presented involve many heuristic decisions. Nevertheless it, too, is a step in the right direction from the use of pseudo-random test sets, or purely heuristic test instruction sequences.

Apart from these two methods there have been few developments which are immediately relevant to the practice of testing LSI. In the absence of any sound and generally applicable theoretical basis for generating tests for LSI devices, the use of ad hoc and empirical test methods in ATE will continue. However, there is no guarantee that such methods will be adequate for testing devices in the future, and research on the topics discussed by Bennetts[37] is clearly necessary.

2.3 Developments in Field Service

The development of field service techniques to deal with LSI based systems has been somewhat more revolutionary than the development of automatic testing, with several entirely new instruments and techniques having been developed. The more significant field service methods which have been developed since the introduction of LSI will be outlined in the following paragraphs.

2.3.1 Board swapping

One of the first solutions to the problems of servicing LSI-based systems in the field to be adopted was the greater use of board swapping. This, in effect, allowed the LSI test problem to be removed from the field into a service centre where it could be dealt with by ATE, or, at least, by a technician who was familiar with the faulty board.

While board swapping was a satisfactory method of maintaining a small range of LSI-based products, the increasing use of microprocessors in a variety of products meant that many manufacturers found

that the stock of spare boards required in the field to maintain their products became uneconomically large[1][5][21][52][79]. Other problems experiences with board swapping programs were that the turn-around times for boards returned to be repaired were often very long, which further increased the necessary size of the spare board stocks which were held in the field, and that a large number of boards which were returned for repair showed no fault when tested at the central repair site[1][52][79]. The latter problem was considered to be partly due to marginal interface characteristics between boards in the faulty system, and partly due to the unnecessary replacement of good boards in the system by field service technicians, in an effort to effect a quick repair.

In some applications, particularly in the case of manufacturers who only produce a small range of LSI based products, and have a small field service staff, board swapping is still an attractive solution. In general, however, it is not a satisfactory solution and many manufacturers have been forced to find alternative methods.

2.3.2 New test instruments

The first instrument which was developed specifically for the diagnosis of LSI based systems was the logic state analyser, introduced by Hewlett-Packard in the early 1970's[80]. Early logic analysers could record and display up to sixteen words of sixteen bit data, sampled on a clock signal which was derived from the system under test, and had the facility to commence recording the data after the occurrence of a selectable data pattern, or "trigger word". In a microprocessor system, the execution sequence of the microprocessor could be displayed if the sixteen data lines were connected to the data or address lines, and the clock signal were derived from a

suitable microprocessor status or strobe line. This solved the problems of observing and interpreting nonrepetitive bus waveforms.

The current range of logic analysers[66][80][81][82] have the ability to record up to 1000 words of data up to 64 bits wide, and to display the data in binary, octal or hexadecimal notation. Other models can record data asynchronously at clock speeds up to 400MHz, and produce a timing display of up to 16 data channels. Another group of logic analysers have "personality modules" for a range of different microprocessors which allow the analyser to be connected directly to the microprocessor through a forty pin test clip so that the instruction execution sequence can be traced. The modules often enable the instruction sequence to be displayed in disassembled (that is, mnemonic) form.

Logic analysers are, without doubt, the most powerful and versatile instruments available for diagnosing LSI based systems, but there are a number of disadvantages involved in their use for field service[79][80]. The most serious of these is that the diagnosis of faults using logic analysers is necessarily a very slow process. This is because, in the first instance, unless a personality probe is available for the microprocessor being tested, the connection of the multitude of data and clock probes to the system under test is a very slow process, as is changing the probes in the course of diagnosis. Secondly, and more significantly, the interpretation and analysis of the data displayed is a slow, detailed process for which a thorough knowledge of the system and how it should behave is required[53][83].

These characteristics of the use of logic analysers have two consequences in practice. The first is that, because tracing system activity instruction by instruction is such a slow process, field service using logic analysers is very expensive. The second is that, because the technician must be very familiar with the system under test, he must spend a lot of time becoming familiar with both hardware and software before testing the system. If he is required to service several different systems, a lot of time will be spent purely on familiarisation, effectively increasing the repair time for each system, which makes the field service process even more expensive. Logic analysers, therefore, are not an attractive solution to the problems of field service and are not widely used in that application except to deal with particularly difficult problems. The main area of application for logic analysers is the development of LSI based systems, for which they are invaluable.

It is appropriate to consider single-stepping as a means of debugging microprocessor systems in this section because, although it is a new technique rather than a field service instrument, it is one which has much in common with the use of logic analysers. The implementation of single stepping requires that simple hardware be included in the microprocessor system so that the system can be switched into a mode in which execution by the microprocessor proceeds step-by-step (usually one instruction or clock cycle at a time). In this single-step mode, logic levels on the address, data and control busses are held constant at each step and a logic probe may be used to observe them. It is a very simple, yet often very effective means of enhancing the testability of a microprocessor system.

Like logic analysis, single-stepping allows execution by the microprocessor to be traced step-by-step, but the hardware provided usually does not include triggering facilities to allow execution to be traced through only specific sections of a program. Moyer[84] describes an unusually elaborate single step unit which does provide such facilities. Single-stepping also shares the characteristics of logic analysis that it is a very slow process and requires familiarity with system hardware and software. It has the additional disadvantage that it obviously does not test the system at full speed. Nevertheless it can be as effective as the use of a logic analyser in detecting "hard" faults (stuck-at nodes, open circuits, bridging faults etc.) without requiring expensive test hardware.

Another technique related to the use of logic analysers which was developed specifically as an aid for the development and diagnosis of microprocessor systems is in circuit emulation (ICE). An in circuit emulator is a piece of hardware which emulates the behaviour of a particular microprocessor, usually under the control of software running in a microprocessor development system. The emulator is connected to the system under test through a plug inserted into the microprocessor socket (in place of the microprocessor) and appears, to the system, to be a functional microprocessor. The emulator hardware usually provides the facilities to trace and record "processor" activity in real time, to start recording after certain trigger data occurs, and to cause a break in the execution sequence on the occurrence of "breakpoint" data. These facilities can be controlled through commands issued from the console terminal of the development system, and trace data can be displayed on the console terminal. Other facilities often provided include the ability to access development system memory as if it were memory contained in the target system, and the ability to perform

emulation at the end of a software development process (assembly or compilation) and use symbolic address and variable names, as defined in the program.

In circuit emulators are generally more flexible and powerful in debugging microprocessor systems than logic analysers, because they share the power and resources of the development system. However, they exhibit the disadvantage that the system under test is usually only accessed through the microprocessor socket, which means that essentially only the system busses are directly visible. Activity in logic which is not connected directly to the busses cannot easily be monitored, as it can with a logic analyser. Because they are used with expensive development systems, in circuit emulators are also more expensive than logic analysers. Furthermore, they share the disadvantages of logic analysers that they are slow to use and require a thorough knowledge of the system under test[79]. For these reasons, in circuit emulators as just described are rarely used in field service applications.

In recent years, however, several portable in circuit emulators have been developed for use as field service instruments[85][86]. These instruments provide similar facilities to the development system based emulators, but with commands issued and data observed through switches and displays on the front panel of the instrument itself. They are generally less flexible than the original emulators because they lack the resources of a full development system, but they are correspondingly less expensive. Once again, they share the disadvantages of other instruments in this class that they are slow to use, require detailed knowledge of the system under test and therefore, are expensive to use.

A more recent group of field service instruments combine ICE with the facility to download test routines for the system under test from a remote computer through a modem and inbuilt serial data interface, which enables the system to be tested almost automatically. These instruments will be discussed in a little more detail later in this chapter.

2.3.3 Self-testing

A technique which has, in recent years, become very popular as a means of enhancing the testability of microprocessor systems is self-testing. This has been encouraged to some extent by continually decreasing memory prices which have meant that it is possible to include a ROM containing a self-test program in a system, without significantly increasing its cost.

Self-test programs in typical microprocessor based equipment are executed when power is first applied to the equipment, or when initiated by an operator[87]. In the execution of a typical self-test routine the microprocessor would calculate a checksum on the contents of all ROM in the system, perform a simple read/write test on system RAM to detect any obvious faults, perform simple tests on input/output (I/O) devices such as switches and displays, and, possibly, perform some form of test on the CPU itself [31][35][88][89][90]. If any errors were detected while performing any of these tests, some indication would be given to the operator that the system contained a fault. The complexities and thoroughness of tests performed during self-test routines vary, there being a tradeoff in the sense that better fault coverage can only be achieved at the expense of more ROM to hold longer test routines[89]. In systems which are provided with extensive self-test routines, some indication of the nature of any fault which may

be detected is usually given to the operator. An even more extensive system of self-test and self-diagnostic routines for a microprocessor system is described by Srini[91].

The popularity of self-test routines in microprocessor systems is largely due to the ease with which self-test programs can be written which, in turn, is a consequence of the inherent modularity of microprocessor systems. The fact that the major system components - ROM, RAM and I/O - can be addressed and tested almost independently because of the modular architecture of microprocessor systems, means that test routines can be written independently to test each module as thoroughly as is convenient. In fact, this modularity means that self-test programs are usually designed to test one module at a time and are therefore, to a large extent, self-diagnostic.

There are three significant limitations on the usefulness of self-testing as an aid to field service. The first of these is that self-test routines commonly only give a "go/no go" indication after the system is tested. Even the more extensive self-test routines only diagnose faults down to the module level and cannot identify which component is faulty, which is what is ultimately required. Indeed, it is difficult to test, in a self-test routine, any devices which are not directly connected to the system busses, so complete fault resolution could not be expected from any self-test routine alone.

The second limitation is that certain faults in the system can prevent execution of the self-test program. Each microprocessor system possesses a "kernel" - that section of the hardware which must

be fault-free for the system to execute programs stored in ROM. If a fault develops in the kernel the self-test program will not be executed and will be of no use at all[91][92].

Finally, when a system performs a test upon itself there exists a possibility that a fault in the system will go undetected because of a second fault in the system which prevents the self-test routine from working correctly. Bilton[35] and Ballard[93] discuss this phenomenon of "fault masking" in self-testing systems and suggest an approach to writing self-test routines in such a way that it is least likely to occur.

Despite these limitations, which lead to the obvious conclusion that self-testing is not a complete solution to the problems of field service, self-testing is a useful technique, particularly for notifying an operator that the equipment he is about to use is faulty. It is because self-testing provides this "fail-safe" feature that it is so widely used in microprocessor based equipment.

2.3.4 Signature analysis

In 1977 Hewlett-Packard announced the development of signature analysis, the first really new approach to field servicing microprocessor based equipment[54][94][95]. It was developed as an alternative field service method to board swapping which had promised to become an extremely expensive process for Hewlett-Packard, a manufacturer of a large range of microprocessor-based equipment.

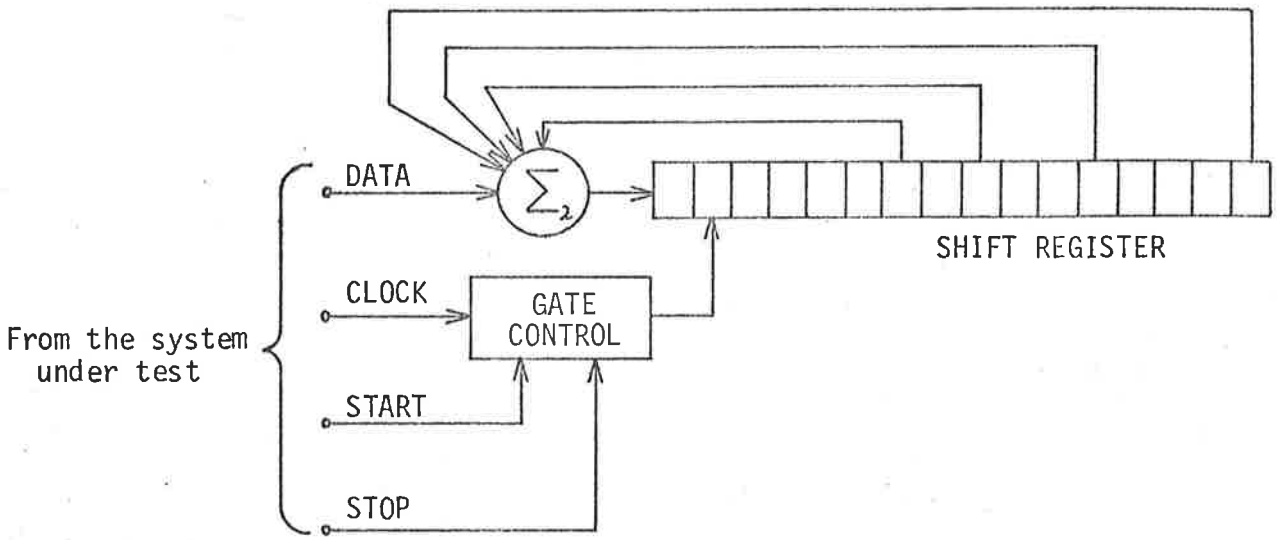
Signature analysis (SA) was a completely new approach to the diagnosis of faults in digital systems which, it was claimed, would overcome most of the limitations of the methods previously used (and

discussed above).

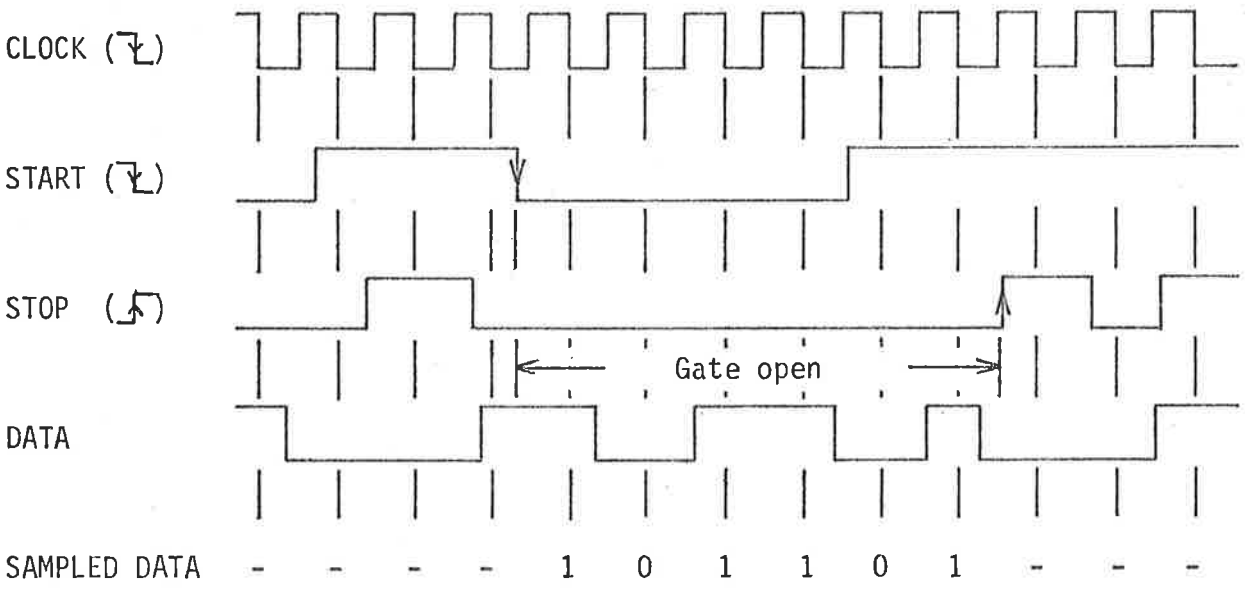
2.3.4.1 The principles of signature analysis

SA is based on the technique of data compression by a linear feedback shift register[54][94][96]. Data sequences in the system under test are sampled and compressed by the feedback shift register to form a sixteen bit "signature" which characterises the data sequence. The data is sampled from the system synchronously with a clock signal which is derived from the system, over an interval defined by "START" and "STOP" pulses, also derived from the system under test (Figure 2.1). The sampling and compression of the data are performed by an instrument known as a signature analyser[94]. Thus, instead of a data sequence being identified by a timing trace on an oscilloscope or logic analyser, it is identified by a four hexadecimal digit signature. Because of the feedback, the signature depends on the length of the sampled data sequence and on the value of each bit in the sequence. If two data sequences produce different signatures the sequences must be different.

With a sixteen bit shift register there are only 65536 different signatures. However, there is an infinite number of possible data sequences which may be observed at a given node, only one of which will be "correct" - that is, the data sequence which would appear at that node in a fault-free system. Therefore, there exists a possibility that an incorrect data sequence at a node would produce the same signature as the correct sequence and would be interpreted as being correct, if it were observed by the signature analyser. Frohwerk[96] has shown that the probability of such an error is less than 2^{-16} or .002% and that if a data sequence differs from the correct one by exactly one bit, it will produce an incorrect signature.



(a) Feedback Shift Register.



(b) Example Data Sequence.

Figure 2.1. Operation of the Signature Analyser

Data is clocked into the feedback shift register between the specified edges of the START and STOP pulses (that is, while the "gate" is open).

SA is used to test a component in a system in the following manner. Signatures at the inputs and outputs of the component are observed while the component is being exercised or stimulated within the system. If the device produces correct output signatures in response to the correct input stimulus (that is, if all input and output signatures are correct) then it is assumed to be fault-free. If, however, its input signatures are all correct but one or more output signatures are not, it is assumed to be faulty.

To enable components to be tested in this way there are two requirements. The first is that the component must be stimulated within the system to repetitively perform some meaningful operation which will indicate whether or not the device is functioning correctly. Appropriate START, STOP and CLOCK signals must be provided to enable the input and output data to be sampled synchronously. The operation must be performed repetitively so that signatures may be taken in turn for each of the input and output data streams, over the one sample interval delimited by the START and STOP pulses[97].

The second requirement is that all feedback to the inputs of the device under test (DUT), from its own outputs and from other untested logic in the system, must be removed. Otherwise if the DUT were faulty, or if a fault existed elsewhere in the system, the feedback may result in an incorrect input data sequence to the device, which would produce incorrect input and output signatures. Such a result is inconclusive.

Because of these two requirements SA is usually applied to test microprocessor systems as a two stage procedure[53]. The microprocessor (CPU) itself is the first device to be tested as it is the

central controlling element in the system and will subsequently be used to stimulate other devices in the system. However, it receives feedback (primarily on the data bus) from other devices in the system. In particular, if the ROM containing the program code were faulty, it would supply the CPU with incorrect instruction codes, which would cause the CPU to perform incorrect operations and would result in an inconclusive test of the CPU. While the CPU is tested the data bus must therefore be disconnected from the rest of the system. The usual way of achieving this, while still stimulating the CPU so that it can be tested, is to permanently force the "no operation" (NOP) opcode onto the CPU data bus. This is sufficient stimulus for it to continually attempt to fetch an instruction (the NOP) and increment the address placed on the address bus. Other inputs to the CPU - typically bus request, interrupt and "wait" lines - must also be disconnected from other components in the system and tied to inactive levels.

While the CPU is continually executing NOP instructions ("free-running") it should produce the correct status and strobe line outputs to read data from memory, while the address bus contents are repetitively incremented through the range from 0 to 2^{16}-1 (assuming a sixteen bit address bus). Signatures can be taken of the data sequences which appear on the address and control busses and if these are all correct the CPU is assumed to be functional. During this "free-run" stage it is also possible to check the contents of the ROMs in the system. While free-running, the CPU produces the correct status and addresses to read the ROMs, which therefore place data on their data bus outputs even though the CPU does not actually see the data read from the ROMs. If signatures are taken at the "ROM side" of the data bus, the contents of each ROM location are sampled and

can be verified.

After the free-run stage it will have been verified that the CPU can fetch instructions and that the ROM's can supply the correct data when properly addressed. The data bus is then reconnected and the CPU is forced to execute a program which is stored in ROM. This program is designed to stimulate the remaining components in the system so that the signature analyser can be used to verify their behaviour. The tests (which must be performed repetitively) would typically include a simple RAM test, a "walking bit" test on parallel ports and other sundry tests designed to exercise random logic in the system.

This two stage application of signature analysis is described in more detail in the introductory literature on SA[54][94][95]. It is what might be called the "standard approach" to the application of SA to microprocessor systems, being universally advocated as the procedure to follow for all such systems.

A number of advantages are claimed for SA over other field service methods[54][94][95]. The free-run stage of the standard procedure allows the system kernel (CPU, ROM, busses and address decoding logic) to be quickly and conveniently tested before the "software driven" second stage is started. Therefore it can be reasonably expected that if no errors are found during the first stage the test program for the second stage will be executed correctly. This overcomes one of the more serious problems of self-testing. SA offers the additional advantage over self-testing of being more thorough because each device in the system can be tested explicitly and directly.

SA can, in general, be applied to test a system much more quickly than logic analysis. The signature anlayser only has four leads which must be connected to the system under test, so the setup time is much shorter than for a logic analyser. The signatures do not require any interpretation - they are either right or wrong - so diagnosis of the systems need not take a long time, nor does it require detailed knowledge of system operation. If the set of signatures expected in a fault-free system is documented in enough detail a technician with virtually no knowledge of the system can test it, taking signatures as instructed by the documentation and replacing the device specified by the documentation when an incorrect signature is found. To summarise, SA held the promise to be a thorough, fast (and therefore inexpensive) field service method which could be applied to any system designed to accommodate it, and for which only a single inexpensive test instrument was required (the HP5004A signature analyser costs around one thousand dollars).

The price which must be paid for the benefits offered by SA is that each system must be designed in the first instance to be tested by the method[95]. Provision must be made, in the form of jumpers or buffers, to open-circuit the data bus and other inputs to the CPU, and to force the NOP instruction code onto the CPU data bus during free-running[10][79][98]. The test stimulus routine for the software-driven stage of SA must be written and a ROM to hold the test routine must be designed into the system. Connectors must also be provided to allow the signature analyser control lines to be connected to suitable START, STOP and CLOCK signals (which may need to be generated explicitly with otherwise unnecessary logic). Finally, and very importantly, the signatures for a fault-free system must be documented in a form which allows fast fault isolation by a relatively unskilled

technician.

While a substantial effort must be put into designing a system to accommodate signature analysis, Hewlett-Packard estimate that the increment in design time and hardware costs involved is only about one per cent[54][99] and this is more than offset by savings in field service costs. It is possible to design SA into an existing microprocessor system ("retrofitting"),[54][100] and Stefanski[101] describes how this can be done for systems based on one of several different microprocessors. However, it is to be expected that retrofitting will require some modification to existing system hardware and, in general, will be more difficult than incorporating SA into a new design.

Since 1977 SA has been slowly gaining acceptance as a field service technique[99][102][103]. Hewlett Packard was, of course, the first company to use the method in a range of products[53][54][92][104], but recent reports[10] indicate that over 300 companies now use the method for field service. It has also been adopted for use in ATE, employing essentially the same approach for testing microprocessor systems as in field service[50][105][106] but with the additional facilities of automatic test generation and control line switching. Its data compression properties have also been used to aid in-circuit testing of LSI devices[69].

SA is widely considered to be the best field service method currently available. Hutcheson[70] states that is is "the most promising of all [field service methods]" while Stephen[98] considers it to be "the only reasonable alternative". Nicholson[79], reviewing several field service methods, states that "SA seems to offer most promise".

Support for the technique has not been unqualified, however. Riezenman[107] notes that it is the most popular technique, but solves only five to ten per cent of problems and Bodt[108] expresses reservations that it "doesn't give 100% answers". A limited range of unsupported figures on the fault coverage of SA in practice have been variously published, but no consistent trend is evident.

2.3.4.2 Developments of signature analysis

Since the introduction of signature analysis and the HP5004A signature analyser, several field service instruments have been developed which either offer improvements on the basic technique, or employ it in a slightly different manner than originally intended[80][82][109].

One of the earliest and most significant developments was the Millenium Microsystem Analyzer (μSA)[98]. As well as performing the function of a basic signature analyser, this instrument incorporates in circuit emulation facilities similar to those described in Section 2.3.2, which allow the signature analysis free-run and software-driven stages to be run under the control of the μSA. Currently, in-circuit emulation of the Zilog Z80, Intel 8080 and 8085, and Motorola 6800 microprocessors is supported. The μSA has complete control over the CPU and can intercept and redirect data appearing at the CPU's data bus pins. Thus the CPU can be free-run without any hardware in the system under test being changed, and a test routine for the software driven stage can be executed by the CPU out of a ROM which plugs into the μSA. The μSA therefore largely removes the requirement for extra hardware to be included in the system under test for it to be tested by SA. In particular, this makes retrofitting SA to an existing system much easier than when using only the HP5004A.

The µSA has an RS232C compatible serial port through which system test routines may be downloaded via a modem from a remote computer. This facility, coupled with an automatic test generation software package called "Fastprobe"[110] allows for semi-automatic testing, with guided probe diagnosis, in the field, The µSA also has the facilities to measure frequency, time intervals and other circuit parameters. It costs about five times as much as the HP5004A[79].

Several other instruments are available which combine the function of a signature analyser with other capabilities. The Tektronix 308 logic analyser, for example, can take signatures in the same manner as the HP5004A and display them on its CRT display, as well as perform functions normally expected of a logic analyser. Spector[83] describes the Paratronics 532 logic analyser, which employs a parallel data compression technique, similar in principle to signature analysis, to generate a characteristic signature for its stored data. This signature may be compared to that of a set of prestored data (usually response data of a fault-free system), which enables a technician to very quickly see whether the data recorded from the system under test is as it should be. The value of data compression in this application, as in "traditional" signature analysis is that differences between two large sets of data can be very easily detected. The 532 also includes a non-volatile memory, in which response data for a known-good system and test setups can be stored for reference while servicing a system in the field. Like the µSA, it has a serial port through which test data can be downloaded from a remote computer. In his discussion of the features of the 532, Spector concedes that the flexibility offered by the logic analyser is obtained at the expense of the ease of use of the single channel signature analyser.

In mid-1980 Solartron announced an instrument which they called the "Locator" which can be used to, in effect, perform signature analysis on systems with their feedback loops intact[79][107]. It employs a technique known as "Trace Analysis", in which all bits of a data sequence (up to 32,000 bits long) are sampled and stored internally, and signatures can be displayed for selected sections of the stored data. If the signature for a data sequence is incorrect (because the data sequence contains errors) the operator can, by examining signatures for selected sub-sections of the data sequence, determine where in the sequence an error first occurs. Data sequences are sampled at various nodes around the system until the device is found which has an error at its outputs earliest in the sequence. This device is considered to be the source of the error and therefore, faulty.

Because the Locator can isolate faulty devices in a system with feedback loops intact, many of the design overheads required to implement traditional signature analysis are no longer necessary. Furthermore, the Locator, like the μSA can stimulate the system under test by in circuit emulation, which removes the need to design a ROM containing test routines into the system. It is a very new instrument which has not yet had time to penetrate the field service market, and acceptance of the technique to data is hard to assess.

At about the same time as the Solartron Locator was released, Hewlett Packard announced the development of the HP5001A microprocessor exerciser[111][112], which stimulates a microprocessor system for testing by signature analysis in a similar manner to the Millenium μSA. The 5001A operates similarly to an in circuit emulator. It plugs into the microprocessor socket of the system under test, while the microprocessor is inserted into a socket on the 5001A. In response to codes

entered on a small keyboard the device can cause the microprocessor to free-run (without any changes to the hardware of the system under test), or execute one of several simple stored test routines for RAM, ROM, I/O or the CPU itself. It also includes a feature whereby data appearing on the eight data bus lines may, in some tests, be serialised to produce a single signature, instead of the eight signatures otherwise required to characterise data on the data bus. As in the case of the μSA, the inbuilt free-run hardware and the stored test routines of the 5001A mean that retrofitting of SA to a system is now much simpler, and that design overheads in a new system are considerably reduced. Currently only a Motorola 6800 version of the 5001A is available.

Finally, in late 1980 Hewlett Packard announced the HP5005A signature multimeter[113]. It offers improved performance over the HP5004A signature analyser and incorporates voltmeter and counting functions. It does not represent any development of the basic SA technique and is only interesting from the point of view that Hewlett Packard appear to be supporting the trend towards multi-function field service instruments such as the μSA, Tektronix 308 and the Solartron Locator.

2.3.5 Portable ATE

The development of instruments such as the Millenium μSA and Paratronics 532 reflects a trend towards more automatic testing in the field. Several manufacturers of large ATE also produce smaller, less expensive "portable" ATE for use in the field[13][21][109]. These testers use a variety of the ATE techniques discussed earlier to test both boards and individual devices, but differ from larger ATE primarily in respect of having little or no test sequence development facilities. Typically, test sequences are initially generated on compatible large ATE and then stored in some form in the portable ATE for later use in

the field. The performance of portable ATE, in terms of both test speed and the ability to perform on-line diagnosis is, of course, poorer than large ATE. The ability to download test sequences through a serial port is a common feature of this type of instrument.

Equipment of this type is still relatively expensive, with typical prices ranging up to thirty thousand dollars. In general it is too expensive to be used by individual technicians in one-off field service tasks, so portable ATE is typically used in local field service depots[66], possibly to provide a more local test and repair site for boards replaced under board swapping programs.

2.4 Conclusions

The various instruments and techniques which have been discussed in this chapter, developed by the test industry to meet the challenge of testing LSI components and systems, have proven to be satisfactory in many applications. In the field of automatic testing in particular, many of the techniques which had been used to test SSI/MSI based systems have been successfully adapted to test LSI. However, it is clear that there is a need for ongoing development in all fields of testing so that the more complex LSI and approaching VLSI devices and systems may be effectively tested. Existing methods will not cope indefinitely as device complexity continues to increase.

Steady increases in device integration levels will, in the foreseeable future, give rise to a constant demand for new, faster, and more powerful ATE. The ATE market is a very competitive one, in which

there is currently a great deal of activity. The commercial manufacturers are clearly heavily involved in research and development on improving ATE techniques but (presumably because the field is so competitive) very little of this work is published. For this reason, and because of the strong market pressures which motivate ATE manufacturers, it is doubtful whether any research outside of the market could be of any immediate or practical significance. The areas of research identified by Bennetts[37] would appear to be the most suitable for independent investigation.

The problems of field service seem to be somewhat more immediate. Signature analysis and its derivatives appear to have overcome many of the problems of earlier field service methods. As discussed earlier in this chapter, SA is widely considered to be the most promising field service technique but support for it is by no means universal. While there have been many claims about the advantages of SA and the insignificance of the extra design effort required to implement it, there has been very little published in the way of documented implementations which might support these claims or the claims of the critics of SA. Without a fully documented assessment of SA in practice,it is impossible to identify its strengths and weaknesses or to undertake any further development of the technique. Indeed, because SA is widely accepted as the best field service method, without such an assessment it is not possible to identify which field service problems remain and how they might be increased by the advent of VLSI. Hewlett Packard have very recently added two case studies to their set of application notes on SA[114][115], (one of them is a reprint of the article by Rhodes-Burke [112]), but these serve more as a guide to the detailed implementation of SA than an assessment of the difficulties of the implementation,or of SA overall.

Therefore, as a means of assessing the effectiveness of SA in practice in isolating faults in typical microprocessor systems, and to identify any specific problems in the implementation of the technique with a view to an overall assessment of the current problems of field service, it was decided to implement SA in a microprocessor system. The details of this implementation will be presented in Chapter III. Chapter IV will contain a detailed assessment of SA based on the trial implementation.

CHAPTER III

AN IMPLEMENTATION OF SIGNATURE ANALYSIS

3.1 Aim of the Implementation

The aim of performing an implementation of signature analysis was to determine how effective the technique is in isolating faults in LSI based systems, and to reveal any deficiencies or disadvantages of the technique which are not documented elsewhere. In this way it was hoped to determine to what extent SA is a solution to the problems of field servicing LSI and which of these problems remains to be solved.

Ideally the performance of SA would be assessed on the basis of fully documented implementations on a wide range of LSI based systems. However, it was clearly not possible to perform a variety of implementations in the time available for this study and, in fact, this time restriction meant that it would be possible to perform only one implementation. It was therefore important that this one implementation should be a "typical" application of SA to an LSI based system, to ensure that the results obtained would be representative of other applications, and would allow a meaningful assessment of SA and its capabilities.

This requirement meant that the implementation of SA must be carried out on a system which is typical of most LSI based systems, and yet has features which would be likely to show up problems which could occur in larger and more complex systems. Furthermore, the implementation must follow the most commonly used procedure for applying SA, as closely as possible. In practice, this meant that the "standard

approach" to applying SA described in the early SA literature [53][54][94][95] must be followed.

3.2 The Target System

3.2.1 General requirements

As discussed at the beginning of Chapter II, microprocessor systems are the most widely used and documented LSI based systems, so it was logical to choose a microprocessor system as the target of the SA implementation. These systems typically include several LSI devices (CPU, RAM, ROM and I/O devices) and, for reasons discussed in Chapters I and II, have the distinction of being one of the most difficult classes of systems to test.

Given that the target system was to be a microprocessor system, two approaches to the SA trial implementation were possible: either a new system could be designed and built, incorporating SA; or SA could be retro-fitted to an existing system. The second of these alternatives was chosen for several reasons. As there was no immediate application for a microprocessor system of typical size and complexity at the time of commencing the implementation, a new system would have been designed without any specific design goals other than testability. This would have resulted in an atypical, perhaps even unrealistic, system on which a fair assessment of SA would not have been possible.

Secondly, when SA is retrofitted to a system it is necessary to make changes to the system hardware. The process of explicitly changing system hardware, rather than incorporating hardware into a new

system, emphasizes the differences between the original "untestable" system and the modified "testable" system. That is, the process highlights the steps which must be taken (or the price which must be paid) to include SA in a system.

Finally, it was expected that the benefits offered by the HP5001A, the Millenium μSA, and other instruments which can externally stimulate a microprocessor system could be better judged if the exercise of retrofitting SA to a system was actually performed. A major selling point of these instruments is that they greatly simplify retrofitting of SA. The importance of this advantage would be better appreciated after the problems of retrofitting had been experienced in practice.

3.2.2 The Intel SDK-85

3.2.2.1 Description of the system

The system which was chosen as the target of the implementation of SA is the Intel MCS-85 System Design Kit (or SDK-85). This is a single board microcomputer based on the Intel 8085 eight bit microprocessor. In its basic configuration it includes a 24 key keyboard, a six digit display, an 8355 (or 8755) 2K × 8 bit ROM with two I/O ports, an 8155 256 × 8 bit RAM with three I/O ports and an 8279 keyboard/display controller, A description of each of these devices may be found in the Intel Component Data Catalog 1980[116].

The system can be expanded by the addition of a second 8355 ROM or 8755/8755A EPROM, a second 8155 RAM and buffers for all data, address and control bus lines which are then accessible at connectors on the board. There is a wire-wrap area on the board on which other logic

may be added to the system. Figure 3.1 is a block diagram of the SDK-85.

The 8355 ROM supplied with the basic SDK-85 contains a monitor program which allows programs or data to be entered into RAM and tested. The monitor provides for either the on-board keyboard and display or a 110 baud serial terminal to be used for the entry and display of commands and data. A more detailed description of the SDK-85 is contained in the SDK-85 User's Manual[117] and circuit diagrams for the system are presented in Appendix A.

There were, again, several reasons for choosing the SDK-85 as the target system. Not the least of these was that it was considered to be typical of systems commonly in use, in respect of the use of an eight bit microprocessor, the ROM and RAM capacities, and the I/O facilities provided. It is a readily available commercial system which means that any experiments performed on the system could be readily repeated and independently verified.

The system possesses certain architectural features which appeared to make it a particularly suitable target for the trial implementation of SA. One of the most important of these is that it contains several LSI devices with integration levels which were typical of state-of-the-art devices in the late 1970's. The 8279, in particular, is one of a family of intelligent peripheral controller chips which are coming into more popular use[118][119]. Since the problems of testing digital systems so obviously depend on the integration level of the devices they contain, the SDK-85, with several state-of-the-art LSI devices, was expected to illustrate well many of the problems of testing complex LSI devices. However, it was con-

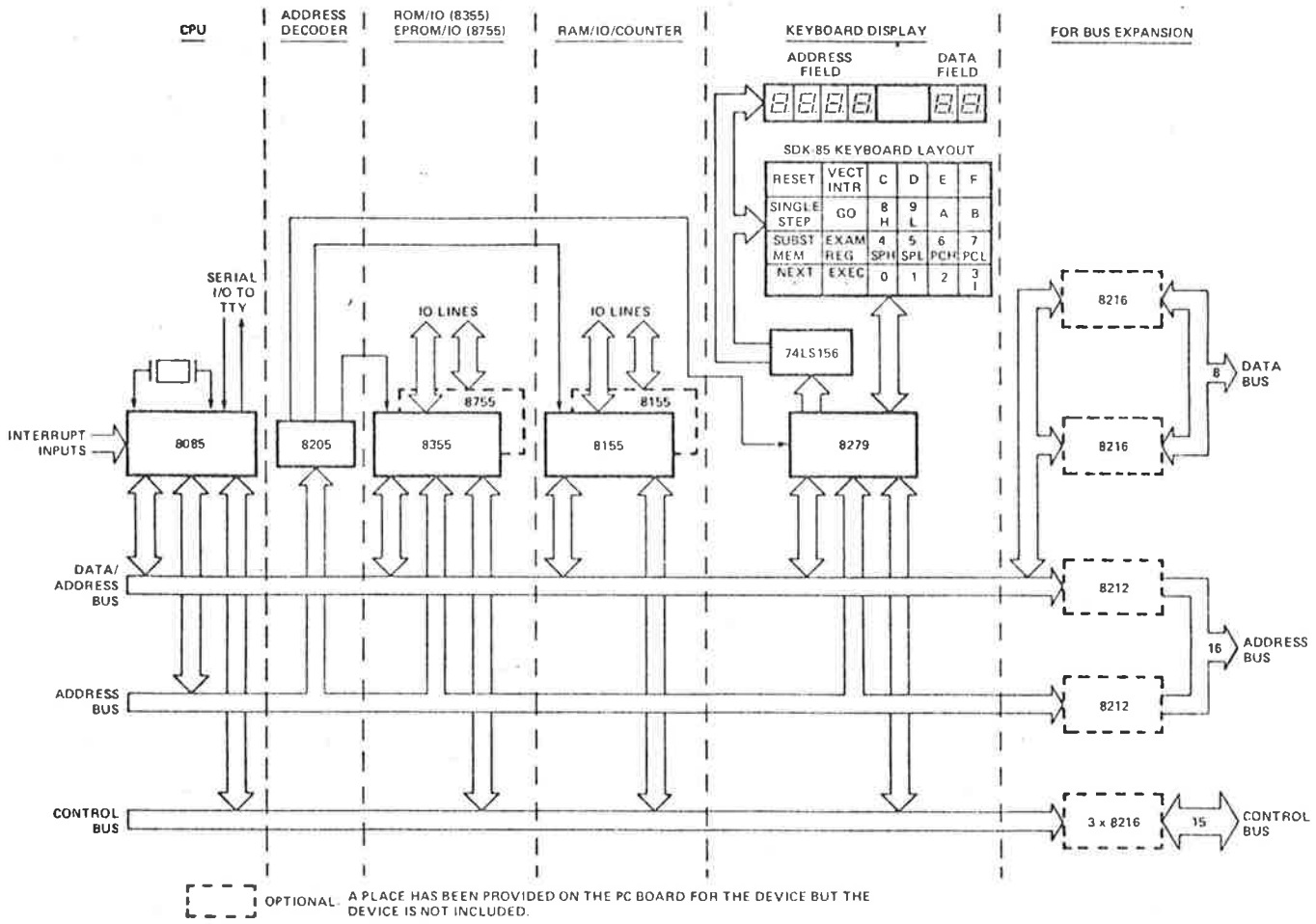


Figure 3.1. SDK-85 Block Diagram.

(Reprinted by permission of Intel Corporation, Copyright 1981.)

sidered to be a sufficiently small system that a thorough application of SA and a detailed study of all problems arising in the system would be possible in the available time. A larger or more complex system would certainly have introduced extra difficulties and highlighted more problems, but a detailed study of all of them would have taken much longer than the study of the SDK-85.

Finally, several SDK-85 kits had been used for several years in the Department of Electrical Engineering at the University of Adelaide. During this time a number of problems with the kits had been detected, and these promised to be suitable test cases which might be used to evaluate the SA procedure which was to be developed.

3.2.2.2 Local modifications

The particular SDK-85 which was used for the implementation of SA differed from the standard commercial system in several minor details. These differences were:

(i) The optional expansion 8155 RAM and all buffers in the system had been installed.

(ii) A 24 pin socket had been added to the wire-wrap area of the board to accommodate a 2708 (1K × 8 bit) EPROM. Simple address decoding logic, which mapped the EPROM into the address range 8000H to FFFFH, was included, A -5 volt series regulator was also added. In the implementation of SA all of this "external" hardware was ignored and the SA procedure was developed as if the address range 8000H to FFFFH were unoccupied.

(iii) A -12 volt, rather than -10 volt, power supply was used (for compatibility with the 2708 EPROM). Two resistors in the twenty milliamp serial interface were replaced with resistors of a higher value to compensate for this change.

(iv) A 10kΩ resistor was connected from pin 29 of the 8085 (the WR/ output) to the +5 volt supply rail, to tie the WR/ line high during system reset. The 8085 WR/ output goes into a high impedance state during reset, and it had been found that, in all SDK-85's which had been expanded as described in (i), RAM locations were overwritten during reset. With WR/ tied high this problem no longer occurred. The remedy is recommended in the MCS-85 User's Manual[120] but, surprisingly, is not implemented in the standard SDK-85 kit.

With the exception of the installation of the expansion RAM and buffers, none of these alterations was considered to be significant for the purposes of evaluating signature analysis on the SDK-85.

3.3 The Signature Analyser

Primarily because of problems with the availability of the commercial unit (the HP5004A), a signature analyser was designed and constructed for the purposes of implementing SA on the SDK-85. A locally built unit offered the additional advantages that modifications to the function of the unit could be easily achieved if they proved to be desirable, and that the limitations of the unit could be better understood.

For example, with an intimate knowledge of the operation of the

instrument, the reason for an incorrect or unstable signature with a particular control line setup could be more easily appreciated and an alternative setup found.

3.3.1 Signature analyser design

The objective in designing the signature analyser was to achieve a level of performance as near as possible to that of the HP5004A[94] subject to the constraint that readily available components must be used. Thus, the analyser was designed to accept a maximum clock frequency of 10MHz, require data and control signal setup times of about 15ns, and have high impedance inputs with standard TTL input logic levels.

The design uses predominantly Schottky and low power Schottky TTL devices. It is based on four 4 bit shift registers with feedback (from bits 7, 9, 12 and 16, as in the HP5004A) through exclusive-or gates. The remaining logic is associated with the START and STOP gating controls, the display, and the unstable-signature detector. A bipolar PROM, programmed to produce the modified hexadecimal character set of the HP5004A, is used as the display decoder. The four input signals (DATA, CLOCK, START and STOP) are buffered by LM360 high speed comparators with feedback to produce the nominal 0.8V and 2.0V input threshold levels. Input to the buffers is through a 51KΩ resistor, shunted by a capacitance of approximately 3pF.

A simple self test facility was provided in the form of a "divide by forty" counter clocked by the display multiplexing clock. The clock and "clock ÷ 40" signals are brought out to a connector on the front panel of the instrument, to which the CLOCK, START and STOP leads can be connected, with the DATA probe connected to a con-

stant '1' input. If the correct signature is then displayed, the instrument is assumed to be operating correctly.

The logic probe feature of the HP5004A was not implemented because it was considered to be unnecessary for the purposes of this study. Similarly, the HOLD mode of operation was not implemented, although if it had proven to be desirable, only minor changes to the hardware would have been required to implement it.

Figure 3.2 is a block diagram of the signature analyser.

3.3.2 Signature analyser performance

It was possible to meet the design goals for the signature analyser using readily available components only by assuming that the propagation delays of the four input signals through their respective buffers would be equal. In practice these delays varied by up to 15ns, which meant that it could not be guaranteed that the instrument would perform satisfactorily at frequencies up to 10MHz with the specified input setup times. The unit was tested on various circuits and it was found that it performed satisfactorily in all cases at clock frequencies up to approximately 5MHz, but at higher frequencies unstable signatures were obtained in some cases.

Although the performance of the unit is certainly inferior to that of the HP5004A, it was considered to be adequate for use on the SDK-85, in which the highest clock frequency is 3MHz, and the shortest valid pulse width is greater than 100ns. In practice the unit did perform satisfactorily on the SDK-85, with unstable signatures being very rare. Certainly, its performance was adequate for the purposes

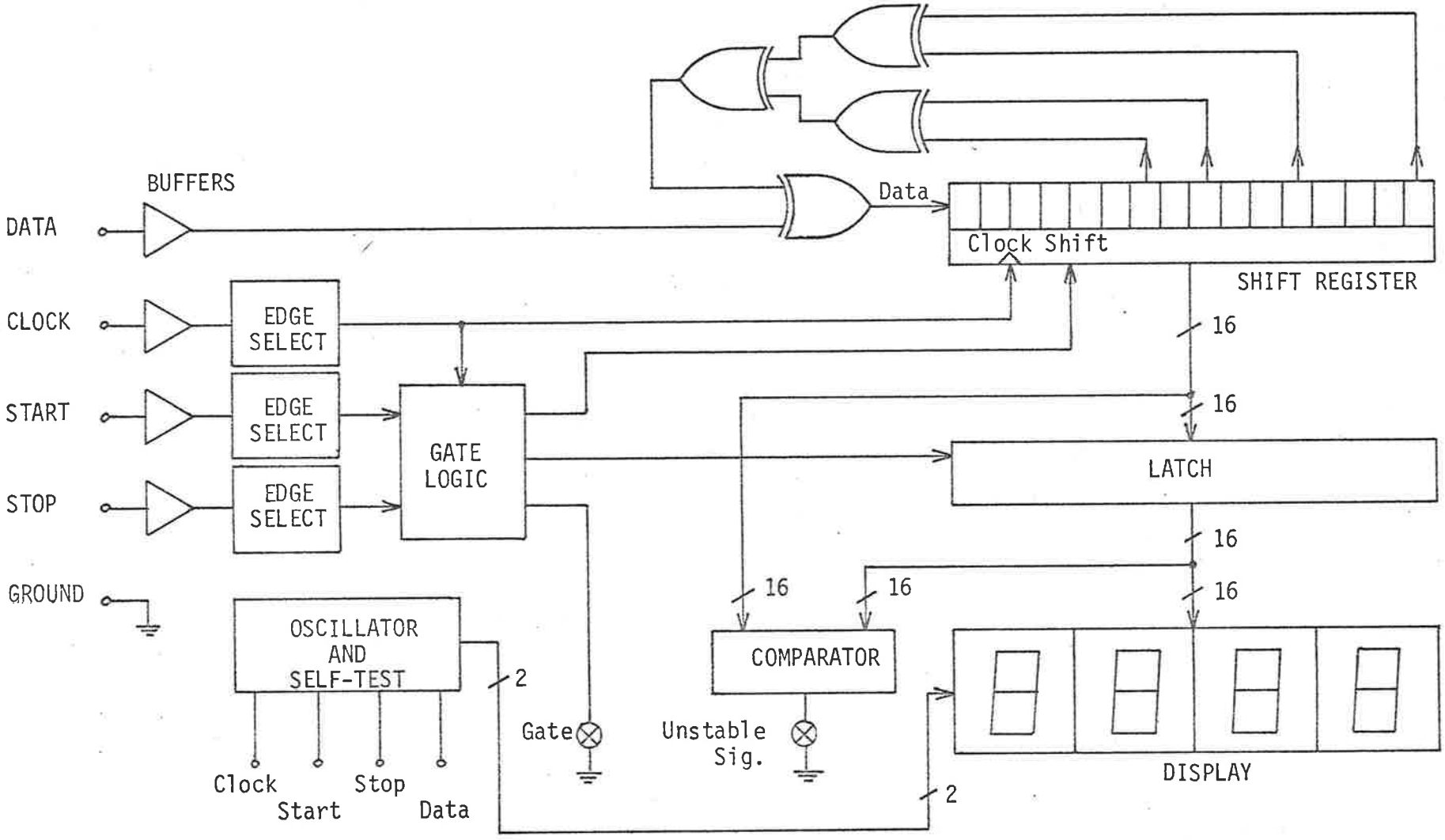


Figure 3.2. Signature Analyser Block Diagram.

of this investigation.

The correctness of the signatures produced by the analyser was verified by two methods. A computer program was written to simulate the operation of the analyser, and this was used to confirm signatures obtained from various counting circuits. This program, written in Pascal to run on a Cyber 173 computer, is listed in Appendix B. Secondly, the signatures observed on the 8085 address bus during free-run were checked against the expected signatures, tabulated by Stefanski[101].

3.4 Development of the SA Procedure

3.4.1 Design philosophy

As discussed at the beginning of this chapter, because the trial implementation of SA must be a "typical" one, the standard approach to the applicaton of SA must be followed. The standard procedure described in the introductory literature on SA[53][54][95] consists of the free-run and software driven stages of SA, in which the major components of the system (CPU, ROM and RAM) are tested in a prescribed manner. However, the methods of testing other components in the system are not prescribed (presumably because there is so much variation in minor components from system to system) and the designer is left with no rigid guidelines on how to proceed in designing a test for these other components. Some suggestions are made in the application notes on SA[95][114][115] but, clearly, much must be left to the discretion of the individual designer.

A fundamental decision which must be made at the outset concerns the degree of fault resolution which will be attempted with the SA procedure. If only the basic steps of testing CPU, ROM, RAM and I/O as prescribed in the literature were carried out, faults in the system could be isolated to one of these major system components, or to the remaining (untested) minor components as a group. If, on the other hand, the test procedures were to be extended to explicitly test all components then, ideally, any fault could be resolved down to a single component, which could then be replaced. The penalty which must be paid for the greater fault resolution would be that the test procedure must be longer and more complex, and the documentation of the procedure must be more detailed.

For the SDK-85 implementation of SA it was decided to attempt fault resolution down to a single replaceable component, as this was expected to allow a fuller assessment of the capabilities of SA. Any inherent limitations on the ability of SA to isolate faults would only be fully revealed if complete fault isolation with the technique were attempted. If a less ambitious goal had been set, the capabilities of SA would not have been tested as fully.

Given this decision to test each component in the system it remained to be decided how, and in what sequence, minor components were to be tested. In this respect the only guidance provided by the SA literature is in the form of a suggested overall approach to testing components. The underlying philosophy of the prescribed tests for major components appears to be that they should be exercised in some way. This is referred to by Gordon and Nadig[54] and in Hewlett Packard's "Designer's guide to signature analysis"[95] as "node-wiggling". The emphasis is not on testing components exhaustively

so much as on stimulating them to perform some operation which might indicate whether they are working correctly. The tests for minor components in the SDK-85 were in general developed to be consistent with this approach and hence, with the tests performed on major components.

In fact, the details of exactly how and when each component was to be tested were determined by an evolutionary, or iterative process, rather than by a firm decision at any stage in the development of the procedure. The procedure was gradually developed from the basic prescribed sequence of testing major system components, to include tests for as many minor components as possible, in the most efficient sequence possible. The iterative approach was found to be necessary to develop an efficient testing sequence, which involves the smallest possible number of signature analyser control line changes, but which also tests components in the order dictated by their input and output signal interdependence.

In its final form the procedure consists of three stages. The first stage is the traditional free-run stage of SA in which the CPU, data and address busses, address decoder, ROM and buffers are tested. The second stage is the software driven stage in which the CPU executes a program to test RAM and the parallel ports. The test program for the second stage is stored in a 2K byte EPROM, installed as the expansion ROM (A15) in the SDK-85. The third and final stage of the procedure is a self-test stage in which the CPU executes a second program (also stored in A15) which was designed to exercise the more complex facilities of the SDK-85, which are not tested in the first two stages.

In the following sections the sequence of tests performed in

each stage will be discussed so that the problems encountered in the implementation and the factors which determined the degree of success of SA in the application might be appreciated.

3.4.2 Stage I

The Stage I test sequence closely follows the standard free-run test procedure described in the SA literature. At the first step the system hardware must be reconfigured to cause the CPU to free run.

The 8085 microprocessor has a multiplexed address and data (AD) bus[120]. In each memory cycle the eight least significant bits of the address are placed on the AD bus early in the cycle, and data is transferred over the AD bus late in the cycle. Therefore, if the processor is to be free-run with all sixteen bits of each address propagating throughout the system, it is not sufficient to simply open-circuit the AD bus and force the NOP instruction code onto these lines. Stefanski[101] describes a free-run adapter for the 8085 which incorporates buffers whose function is to isolate the eight address bits output on the AD bus from the NOP instruction code read in on the same lines. The 8085 must be physically removed from its socket in the system under test and placed in a socket on the adapter, while a forty pin plug from the adapter is inserted into the socket in the system under test. The socket on the adapter ties all CPU inputs (except the data bus and RESET) to inactive levels so that there is no feedback to the CPU from the rest of the system.

An adapter almost identical to the one described by Stefanski was constructed. The only change made was that a 74LS04 inverter was included to buffer the RD/output of the 8085 which has up to three Schottky TTL loads to drive on the SDK-85 board, and would otherwise

have had another four low power Schottky loads to drive on the adapter. A circuit diagram of the adapter is contained in Appendix C.

Before the Stage I test procedure is commenced, the various links on the SDK-85 board must be inserted to tie external inputs to inactive levels and to configure the data bus buffers so that external memory lies in the address range 8000H to FFFFH[117]. For the first test in Stage I, the 8085 is placed in its socket on the free-run adapter and the signature analyser START and STOP leads are connected to the most significant address line (A_{15})[1] and the CLOCK lead to the address latch enable (ALE) line. After the power supplies, RESET and other inputs to the CPU have been verified as being at their correct levels, the signature on the sixteen address lines are observed, to verify that the 8085 is free-running correctly.

It should be noted that the signatures of the eight least significant address lines are observed at the outputs of the buffers on the adapter, which indicates an underlying assumption that the adapter itself is fault free. This assumption applies throughout all stages of the SA procedure to all external test hardware (including, of course, the signature analyser).

A second important point which should be noted with regard to the free-run test of the CPU is that it is possible for an incorrect signature to be observed on one or more address lines as the result

1. *In Intel circuit diagrams and literature, integrated circuits on the SDK-85 are denoted A1 to A17, while address lines are similarly denoted A8 to A15. To avoid confusion, in any discussion of the SDK-85 the integrated circuits will be referred to as A1 to A17, as in the Intel literature, but references to address lines will be subscripted (as A_8 to A_{15}).*

of a fault on the address bus, rather than a faulty CPU. Such a fault may occur at the input of any other device connected to the address bus, or on the printed circuit board as a bridging fault between conductors and,unless a current tracer were used,would be indistinguishable from a fault in the CPU. Therefore, if an incorrect signature is observed on the CPU address lines and replacement of the CPU does not correct the problem, a current tracer must be used to isolate the fault on the bus. The same procedure must apply in all cases in which the output signature of a device is observed to be incorrect.

While the CPU is free-running and incrementing its address values, it provides an excellent stimulus by which the continuity of the address bus may be checked. Signatures are observed at the address inputs of all devices connected to the address bus. Since it has already been verified that the CPU is placing correct addresses on the address bus, an incorrect signature at any point must be the result of a faulty connection from the CPU socket to that point.

It may be noticed that this practice of veryifying only the address inputs to all devices on the address bus at one time differs from the practice recommended in much of the SA literature of checking all inputs and outputs of a single device at one time. It was necessary to use this approach to keep the number of times that the control lines of the signature analyser had to be changed during Stage I down to a reasonable value.

For most devices in the SDK-85, all input and output data cannot be observed with a single control line setup for the signature analyser. In fact, this is true of many components in microprocessor systems,

particularly those in systems which use multiplexed busses. Therefore, if all inputs and outputs of a device were to be checked at once, it would be necessary to change the signature analyser START, STOP and CLOCK inputs at least once for most devices, which would result in an unacceptably slow test procedure. In order that the number of control line changes would be minimised, it was decided, instead, to plan the procedure so that for a given control line setup, all possible signals would be verified before the setup was changed. Thus, all address line inputs are verified together.

The 8205 address decoder (A10) is somewhat exceptional in that, because it is a purely combinational device, its inputs and outputs can be observed on the same clock. The outputs of the 8205 are observed first and then, if any of the output signatures are incorrect, its inputs are checked. If all outputs are correct its inputs are assumed to be also correct. In the most likely case that the 8205 is fault-free and its input data is correct, it will only be necessary to observe the outputs of the device to check it, so this practice reduces the number of signatures which must be observed.

This technique of "half-splitting"[95] was used in the tests for all of the few devices in the SDK-85 which have inputs and outputs which can all be observed with a single setup for the signature analyser control lines. It was also used in some tests designed to verify device interconnections and connections to output connectors. It was not more widely used because of the many different signature analyser setups which are required for the observation of the various signals in the SDK-85. For example it was decided not to verify the the outputs of the 8355 ROM before its inputs because, if an error had been observed, it would have been necessary to change control

lines to observe its inputs. Similarly, half-splitting was not employed on a system-wide scale (as recommended in the "Designer's Guide to Signature Analysis"[95]) because this would have virtually required a new control line setup for each set of signatures observed. Thus the Stage I test procedure evolved as an expanding kernel test in which (in most cases) all device inputs are verified before their outputs.

After the address decoder is tested the chip select inputs to the various devices on the data bus are checked, which verifies the connections from the address decoder to those devices. This is the last test to be performed with ALE as the signature analyser clock.

The signature analyser CLOCK input is next connected to the RD/ output of the 8085, with data being sampled on the trailing (positive-going) edge of the RD/ pulse. This edge occurs late in every memory read cycle, at a time when all status and control outputs from the CPU (SO, S1, INTA/ IO/$\overline{M}$, HLDA and WR/) are valid. With RD/ as clock, these signals are checked at the processor output pins and at the corresponding inputs of all devices to which they should be connected, in a similar manner to that in which the address lines were tested. For reasons which will be explained in the discussion of Stage II, SO and S1 are not included in this test.

While the processor is free-running, it is continually executing an "instruction fetch" memory cycle, so the status and control outputs (as observed on the trailing edge of RD/) should be static and produce stuck-at-one or stuck-at-zero signatures (0001 or 0000 respectively). Therefore, this part of the test procedure serves more to verify that the status and control inputs to the various devices in the SDK-85 are at the correct levels than to verify that the lines are not stuck-at or

open. Each of these lines must be tested at a later stage to ensure that it can assume either logic level.

The path of the RD/ signal throughout the system cannot be fully checked simply by the observation of signatures at RD/ nodes, clocked in the positive edge of RD/. If there were no faults on the RD/ line, this would result in a stuck-at-zero signature in all cases, which (naturally) would also result if the node were stuck-at-zero. It is therefore necessary to verify that the signal at each point which should be connected to RD/ is high when RD/ is high and low when RD/ is low. If a HP5004A signature analyser were being used its inbuilt logic probe would be used to verify that each node is not stuck-at, which, together with the correct signature, would be sufficient to verify that the signal is correct. As the signature analyser being used on the SDK-85 did not have an inbuilt logic probe, the RD/ line is instead verified by the observation of two signatures at each point - one with the positive edge of RD/ used as the clock, and one with the negative edge of RD/ used as the clock. These signatures should respectively be the stuck-at-zero and the stuck-at-one signatures.

The RD/ clock is also used to check the small amount of combinational logic which enables the external data bus input buffers, A4 and A7. This section of the circuit has RD/ as one of its inputs so it is again necessary to observe two signatures at each node, one clocked on the positive edge of RD/ and one on the negative edge.

The system clock signal, CLK, and the small amount of logic associated with it must be tested with CLK itself used as the signature analyser CLOCK input because CLK is the highest frequency signal present in the system. Once again signatures must be observed at each point

with both positive and negative CLOCK edges. For the purpose of verifying the CLK signal path the START and STOP inputs of the signature analyser are connected to RD/, which produces a signature gate interval of four clock periods (the time taken by the 8085 to execute a NOP instruction). RD/ is used for the START and STOP inputs rather than the most significant address line (A_{15}) because the 8085 data sheet[116] does not specify an absolute timing relationship between CLK and A_{15}, and signature instability proved to be a problem when A_{15} was used. There was much less difficulty with signature instability when RD/ was used.

The CLK signal is traced from its source (the 8085) to each device to which it is an input, and from the output of buffer A5 (which, by this stage has had all of its inputs verified) to the external connector J1. CLK is also an input to the hold acknowledge synchronization flip-flop (A9), and signatures are taken at its outputs to verify that they are at the correct (static) levels.

In the next step of the Stage I procedure the path of ALE through the system is checked. As in the cases of RD/ and CLK, there is no signal in the SDK-85 which could be used as the signature analyser CLOCK input for the purpose of verifying the ALE signal. Therefore, it is once again necessary to use the signal itself as the CLOCK input, with signatures observed on both clock edges. Address line A_{15} is again used as the START and STOP input to the signature analyser. Although A_{15} is not guaranteed to be valid on the leading edge of ALE it was found that stable signatures are obtained on either edge of ALE if the gate interval is started on the positive edge of A_{15}, and stopped on the negative edge.

After the ALE input to address latch A6 has been verified, all of the inputs to all buffers to the external connectors have been verified, so the outputs to the connectors can be checked. The outputs of the data bus buffers (A4 and A7) are observed to see that they contain the eight most significant address bits during the early part of each machine cycle (that is, on the negative edge of ALE). The address buffers are tested by the observations of their outputs later in each cycle (on the positive edge of RD/), which also verifies that the latch, A6, retains the data placed on the data bus while ALE is active.

For some of these buffers, as for the 8205 address decoder, A10, it proved to be possible to observe both inputs and outputs with a single signature analyser control line setup. In such cases signatures are observed at the buffer outputs first, and input signatures are only observed if an error is detected. For this reason address inputs to buffer A1 are not verified earlier in Stage I when address inputs to all other devices are verified. In the more likely cases that the buffer is not faulty it will not be necessary to observe its inputs, so the total number of signatures which must be observed is reduced.

At the final step in Stage I, the contents of the two ROM's (A14 and A15) are checked. By this time all control and address inputs to the two devices have been verified, so any errors observed at the ROM outputs must indicate a faulty ROM. While the CPU is free-running, it strobes data sequentially from all locations in each ROM onto the data bus, as explained in Chapter II. For each ROM, the signature analyser START and STOP lines are connected to the chip select line of the ROM (starting on the leading, or negative, edge and stopping on the trailing, or positive, edge) with the trailing edge of RD/ used as the

clock. Signatures can then be taken on the data bus to verify the data read from the ROM. Because data is sampled only while the chip select line of one ROM is activated, only data from that ROM is included in the signature[97], so an incorrect signature can only be caused by incorrect data placed on the data bus by the selected ROM.

At the end of Stage I, all data and address bus connections have been verified, and the various control and status lines have been partially tested. It has been verified that the CPU can fetch instructions and execute at least one. Just as importantly, it has been verified that both ROM's can place the correct data from any of their 2K locations onto the data bus when properly addressed. It can therefore be reasonably assumed that a test program stored in ROM will be correctly executed by the CPU, which means that the software driven stage of signature analysis can be started.

3.4.3 Stage II

With the kernel having been verified in Stage I, the system is reconfigured in Stage II to allow the CPU to execute a test program which stimulates the system so that the remaining major components (RAM and I/O devices) may be tested. The CPU is therefore removed from its free-run adapter and is replaced into its socket (A11) in the SDK-85.

Clearly, some change to the SDK-85 hardware is required to force the CPU to execute the stimulus program instead of its normal "application program" - the SDK-85 monitor. The simplest approach would be to remove the SDK-85 monitor ROM from the A14 socket and replace it with the ROM containing the test program. However, this was considered to be an unsatisfactory approach because it would involve the removal of

a major section of the system, significantly changing its normal operating configuration. If the monitor ROM were to be removed, any faults in the operation of its I/O ports obviously could not be detected during Stage II.

The approach adopted was to store the stimulus program in the expansion ROM (A15) in the SDK-85, and, for the duration of Stage II, swap the chip select lines (CS0/ and CS1/) to the two ROMs, A14 and A15. Thus, throughout Stage II, A14 occupies the address range 0800H to 0FFFH, while A15 occupies 0000H to 07FFH, so the first instruction fetched by the CPU after reset is read from A15. In addition to the changes to the chip select lines, the address line A_{10} input to A15 is tied high during Stage II, so the first instruction fetch cycle after reset (from address 0000H) actually addresses location 400H in A15. In fact, the Stage II test program occupies locations 400H to 522H in A15, most of the remaining space being occupied by the Stage III program.

To implement these addressing changes, the CS0/, CS1/ and A_{10} lines on the SDK-85 board were cut, and connections were made to a socket on the wire-wrap area of the board. For normal operation of the system, (and during Stage I) a jumper plug is inserted which simply restores the broken connections. For execution of the Stage II program, a second plug, which changes the connections as described above, is inserted. A circuit diagram showing details of this addressing scheme is contained in Appendix C.

The Stage II test program stored in A15 (and listed in Appendix E) is designed to stimulate the system RAM, the parallel ports, and the 8279 keyboard/display controller. The parallel ports are first

initialised as output ports, and all output lines are set to zero. The 8279 is then initialised and a "walking ones and zeroes" pattern is written to its display RAM, to stimulate the display multiplexing and driving circuits. A loop is then entered in which three tests are performed repetitively. The first test writes a simple "walking bit" to each of the parallel ports in the system, to enable the connections from the ports to the I/O connectors (J3, J4 and J5) to be checked. The other two tests are independent tests on each of the 256 byte RAM's in the system.

Each of these tests is constructed in such a way that it requires that a set of signatures be observed over a gating interval which lasts for the duration of that test only. Thus in each pass through the test loop there are three separate gating intervals during which signatures are taken to test the output ports, the basic 256 byte RAM, and the expansion 256 byte RAM. The START and STOP pulses which delimit these three gating intervals are produced at the unused outputs of the 8205 address decoder by the execution of dummy input and output instructions which activate these outputs as appropriate at the beginning and end of each test. Thus, the output port test is performed between instructions which activate the CS2/ and CS7/ outputs, the first RAM test between CS7/ and CS6/, and the second RAM test between CS6/ and CS2/. In each case the decoder output is activated by both input and output instructions so that either RD/ or WR/ may be used as clock for the signature analyser, as convenient.

At the first step in Stage II all inputs to the 8085 (except the data bus) are checked to ensure that they are inactive, and the reset input and output lines of the CPU are tested with a logic probe. While the CPU is executing the Stage II test program its activity is

much more varied than in Stage I, in which all memory cycles are instruction fetches. During execution of the Stage II program, the processor performs input/output and memory reads and writes, "wiggling" the WR/, SO, S1 and IO/$\overline{M}$ lines which are static throughout Stage I. Although some of these lines are checked during Stage I, this is only done to verify that all inputs to devices which are tested during Stage I (the ROMs in particular) are correct. SO and S1 are not used as inputs to any devices other than buffers, and therefore need not be, and are not, tested in Stage I. All of these lines are therefore tested early in Stage II to verify that none of them is stuck at either logic level.

The WR/ signal path is checked first of all, with WR/ itself being used as the signature analyser clock, and signatures being observed on both clock edges. The negative edge of ALE is used as the clock to trace the other status outputs, with the negative edge of CS7/ being used as the START edge, and the positive edge of CS6/ as the STOP edge (that is, the status lines are monitored during execution of the first RAM test). The START and STOP edges were selected so that the state of the status lines during execution of the input and output instructions which generate the START and STOP pulses would be sampled, so that a logic '1' level would be observed on the IO/$\overline{M}$ line (which is otherwise always '0' during the RAM test).

The tests performed so far in Stage II are, in effect, leftovers from Stage I - those tests of system busses which could not be effectively performed while the CPU was only free-running. At the next step, the results of the first test performed explicitly by the Stage II program (the RAM test) are observed.

The RAM test consists of a simple read/write test which is performed independently on each of the two RAM's in the system, with data read from the RAM's being verified by the signature analyser. For this purpose, the signature analyser is clocked on the positive edge of RD/, with its START and STOP inputs connected to the 8205 CS7/ and CS6/ outputs (respectively) for the first RAM test, and to CS6/ and CS2/ for the second RAM test. With the signature analyser set up in this manner, signatures observed on the data bus depend on the data read from RAM during the RAM test. In fact, signatures observed reflect data read from ROM during the RAM test as well as data read from RAM, but since the ROM contents have been already verified, an incorrect signature must be the result of incorrect data being read back from RAM. It should also be noted that the signature analyser is configured to start sampling on the positive edge of the START pulse, and stop on the negative edge of the STOP pulse. Thus the (undefined) data present on the bus during execution of the input instructions which create the START and STOP pulses is not included in the signatures.

The RAM test is constructed such that each 256 byte RAM chip is tested separately and, because all temporary variables are maintained in CPU registers throughout the test, will execute correctly even if neither RAM chip is present. The test was not intended to be exhaustive nor to detect any particular form of pattern sensitivity. It was designed to that verify each 8155 RAM chip contains 2048 unique storage bits, organised as 256 eight bit bytes. As the first part of the algorithm a "marching ones and zeroes" test[7][47] is performed on the RAM, which verifies that there are 256 addressable locations, that writing to any one location does not overwrite any other, and that none of the RAM bits is "stuck-at". In the second part of the algorithm, the data sequence 55H, 33H, 0FH is written to, then read from each RAM

location, to verify that each bit in each byte is unique, and not stuck together with any other bit in the byte. It would have been possible to perform a more elaborate RAM test, based on a better defined fault model, but this algorithm performs a reasonable functional test on the two RAMs, occupies little ROM space and has an acceptably short execution time.

After the two RAMs have been tested, signatures are observed at the output lines of all parallel ports to verify that they are responding correctly to the walking bit test. The signature analyser gating interval for the parallel port test is delimited by a start pulse at the 8205 CS2/ output and a stop pulse at the CS7/ output. The stimulus program initially sets the outputs of all ports (there are eight in a fully expanded SDK-85) to zero and then walks a '1' bit across each port in turn. The output data sequence at each of the port lines (there are up to 76) is initially observed with the signature analyser at the external port connectors (J3, J4 and J5). If the signatures for a given port are correct, both the port itself and the connections between the chip and the port connector are assumed to be fault free. If an incorrect signature is observed at the connector, the signature at the corresponding output pin of the RAM or ROM device is observed to determine whether the fault is in the device or in the connection to the external port connector.

The CLOCK input to the signature analyser for the parallel port test was connected to the RD/ line, with data being sampled on the positive edge. The propagation delay through the 8755 and 8155 ports, from the trailing edge of the WR/ pulse which strobes data into the ports to the change of data at the output lines, can be up to 400ns. However, the trailing edge of the first RD/ pulse after data is written

to the ports (generated during the fetch of the instruction immediately following the OUT instruction to the port) occurs no less than 790ns after the trailing edge of the WR/ pulse, so data on the port output lines must be stable on the positive edge of RD/.

Finally in Stage II, a preliminary test is performed on the 8279 keyboard/display controller and its associated key scanning and display driving circuits. At the start of the Stage II test program the 8279 is initialised to display sixteen digits, with encoded scanning of the display and keyboard. Data is also written into the display RAM of the device, to generate a "walking one" and "walking zero" pattern on its eight segment latch output lines as the display is multiplexed. When the controller is operating in its sixteen digit encoded scan mode, a binary count from 0000 to 1111 should be produced at its scan line outputs (SL0 - SL3). During this count the display and keyboard are each scanned twice because the most significant scan line (SL3) is not connected to the 74LS156 demultiplexer. This should result in the top eight bytes and the bottom eight bytes of data in the display RAM being effectively superimposed when displayed on the seven segment displays. Thus, since the top eight bytes and bottom eight bytes of the display RAM are initialised to contain complementary data, all segments of each of the six display digits should be uniformly lit during Stage II.

The first step in testing the 8279 is verification of the scan line outputs. For this purpose, SL3 is used as the START and STOP inputs to the signature analyser, with both edges of SL0 used as the clock. Signatures are observed on each of the scan line outputs. The same control setup is used to check the outputs of the 74LS156 demultiplexer, A12. However, because they are open collector outputs, it is necessary to connect a 10kΩ resistor between the signature analyser

data probe and the +5V supply rail to pull the outputs up to a logic '1' level when they are inactive.

The same set of control inputs to the signature analyser is also used to test the keyboard. Signatures are observed at each of the eight return line (RL) inputs to the 8279, while each key is pressed in turn. At each return line if no key connected to that line is pressed, the signature should be the stuck-at-one signature because of the internal pullup resistor at each of the 8279 RL inputs. If a key connected to the line is pressed, the signature observed should be that of the corresponding scan output of the demultiplexer, A12. If an incorrect signature is observed at one of the RL inputs while a key is pressed the connection from the output of A12, through the switch, to the RL input, must be checked with an ohm-meter or similar instrument.

It was found to be impossible to check the segment outputs of the 8279 by signature analysis. Although the segment output data is synchronous with the scan line count, a blanking mechanism in the 8279 places the digit blanking code (FFH) on the segment outputs while the scan lines are changing. Therefore data sampled at any segment output on either edge of SL0 will always be '1', so if SL0 is used as the clock, the signature observed at the segment output will be the stuck-at-one signature, irrespective of the data appearing at the output between scan line changes.

The timing of all of the 8279 outputs is derived from the CLK input and so, in principle, it should be possible to use CLK as the CLOCK input to the signature analyser for the observation of any 8279 output. However, it was found in practice that when CLK was used as the signature analyser clock, signatures observed at the scan line and

segment outputs were all unstable. Changes in the scan line outputs (that is, on the SLO output) occur at a frequency which is $\frac{1}{128n}$ of the CLK frequency, where n is the value of the programmble prescale factor and is usually set to 31 in the SDK-85 to give the recommended 5.1 millisecond keyboard scan time. However, the 8279 data sheet[116] does not specify an absolute time relationship between CLK and SLO. The signature instability is therefore attributed to an accumulation of propagation delays (possibly exceeding one period of CLK) within the 8279 clock divider chain resulting in indeterminate values of the output lines on the clock edges. Observation of the CLK and SLO signals on an oscilloscope confirmed that changes in the SLO outputs did not occur at any consistent point within the CLK cycle.

Because there is no suitable clock signal which may be used to observe the 8279 segment outputs in particular, the data appearing at these outputs could not be verified directly. However, in spite of this limitation it is possible to detect some faults in the display driving circuitry (including the segment outputs of the 8279) by observation of the display. A display other than one in which all segments are uniformly lit indicates a fault in either the driving transistor and associated hardware or the 8279 itself. If an incorrect display pattern is observed, conventional methods (such as diagnosis of the display circuit with an oscilloscope) must be used to isolate the fault. Even if the 8279 segment outputs could be observed with the signature analyser, it would still be necessary to use an oscilloscope to isolate any faults in the display driving circuits because the voltage levels around the driver transistors are not standard TTL levels and could not be observed with the signature analyser. Nevertheless, the inability to directly verify the 8279 segment output data does complicate the process of isolating faults in the display circuit

considerably.

If no faults are apparent in either the keyboard or the display circuits after these preliminary tests it may be reasonably assumed that there are no serious faults in these sections of the circuit. If the 8279 is also free of serious faults, it is now possible to use the keyboard and display to interact with a program running in the SDK-85 which performs a series of more complex, automatic tests in Stage III.

3.4.4 Stage III

The purpose of Stage III of the signature analysis procedure is to test those facilities in the system which could not be conveniently tested in Stages I and II. These include facilities which cannot be tested repetetively or in such a way that they can be verified by signature analysis; which require human intervention during the test; or which can be more quickly and completely tested entirely under the control of a self-test program. The feature which distinguishes Stage III from the earlier stages is that it involves extensive interaction (through the keyboard and display) with the operator - the person conducting the test. Instead of a few simple tests being performed repetitively to allow observation of activity within the system with a signature analyser (as in Stage II), in Stage III several complex tests are performed once only, with the response to each test being monitored by the CPU itself under the control of the self-test program. The outcome of each test is indicated to the operator on the display.

The "standard approach" to SA, as described in the early SA literature, does not include the equivalent of Stage III. In fact, Stage III was only included in the SDK-85 SA procedure to test the system more thoroughly than a straightforward application of the

"standard approach" would have done, in an attempt to achieve the goal of fault resolution to a single component. The SDK-85 is a somewhat unique system in as much as it has facilities for human interaction and it is a general purpose system with many general purpose facilities (I/O ports, interrupts etc.) which should all be tested. Most microcomputer systems would not have all of these facilities, so the need for a final, automatic, interactive test stage would not be as great. Indeed, if a system does not have some means of human interaction, a series of tests such as performed in Stage III would not be possible.

The Stage III test program, which is listed in Appendix E, is stored in the 2K byte expansion ROM (A15) together with the Stage II program. It is much longer than the Stage II program and occupies locations 000H to 331H and 640H to 73FH in A15. Execution of the Stage III program on reset is arranged by the insertion of a third jumper plug into the address selection socket on the wire wrap area of the board. This plug interchanges the CS0/ and CS1/ line to ROMs A14 and A15, but leaves the A_{10} address line connection to A15 intact. Thus, on reset the CPU starts executing from location 000H in A15.

At the start of the Stage III program further tests are performed on the 8279 keyboard/display controller. It is initialised and then data is written to its display RAM so that a sequence of 24 characters should appear to be continually shifted across the six digit display. If any errors are observed in the display sequence the fault is assumed to lie in the 8279 since the display driving circuitry is assumed to be fault free after Stage II. When the operator is satisfied that the display sequence is correct he must press one of the keys on the keyboard and the test will be stopped.

While performing the display test the 8085 monitors the 8279 input buffer status and its interrupt line to determine whether any characters have been entered into the buffer (that is, whether any keys have been pressed). If the display sequence does not stop when the operator presses a key, this procedure for detecting a key closure has clearly failed and the 8279 is assumed to be faulty because the keyboard and all address data and control bus connections to the 8279 are assumed to be fault free.

When a key closure is detected the 8085 starts executing a routine which is designed to test both the 8279 interrupt generation logic and the operation of the 8085 RST5.5 interrupt. Tests are performed to verify that:

(a) the RST 5.5 interrupt of the 8085 is asserted if, and only if, the 8279 input buffer is not empty;
(b) when the RST 5.5 input is asserted, 8085 interrupts are enabled and RST 5.5 is unmasked, a RST 5.5 interrupt does occur; and
(c) the 8279 keyboard buffer becomes empty after one character is read from it.

If an error is detected in any of these tests the message "Err 1n" is written to the display, where "n" is a number which indicates which of the tests failed. If the first test fails the logic levels of the 8279 INT output pin and the 8085 RST 5.5 input pin must be examined to determine which of the two devices (or the interconnection between them) is at fault. If the second test fails the fault clearly lies in the 8085, which is replaced. If the third test fails a multiple key entry has occurred and the test is repeated to determine whether the error is persistent. If it does occur again the 8279 is assumed

to be faulty.

If all of these tests are executed without error a character, which corresponds to the key originally pressed, is displayed on the right hand digit of the display. Any key except "NEXT" may then be pressed and its corresponding character will be written to the display. The operator is required to press each key on the SDK-85 keyboard (except "RESET" and "VECT INTR") in turn and verify that the correct character appears on the display. The test is intended to check the key encoding logic of the 8279 to ensure that each of the 22 keys in the keyboard matrix can be uniquely identified by the controller. The "NEXT" key should be the last one to be pressed, as it will terminate the test and start execution of the next test in Stage III. If both the display and the keyboard tests have run without error it is assumed that the 8279 is fault free and may be confidently used in subsequent tests as a medium of communication between the Stage III test program and the operator.

It should be noted at this point that development of these first two tests in Stage III took far longer than any of the other tests in the entire procedure. This was principally because a number of difficulties were experiences in attempting to use the 8279 keyboard/display controller. In particular, it was found that if commands are written to the 8279 command register in arbitrary order a garbled display can result. It was found, for example, that a "clear keyboard FIFO" command could not be issued after data had been written to the display RAM without the display being corrupted. While this, in itself, is not an unreasonable restriction, it is one which is not documented in the 8279 data sheet[116]. Consequently, it was only by trial and error that the correct command sequence was found which would cause the 8279 to operate

as desired.

The third test to be performed in Stage III is the serial input/output test, for which a test plug must be inserted into the serial I/O socket (J7) to loop the serial output data back to the serial input. The test program first sets the 8085 serial output data line (SOD) then, after a short delay to allow the input filter capacitor (C5) to charge, checks that the serial input data line (SID) is high. SOD is then set to '0' and SID is read again to verify that it follows. If the test fails an error message is displayed and a loop is entered in which SOD is toggled every millisecond to allow an oscilloscope to be used to isolate the fault in the serial I/O circuit. If a square wave is not present at SOD or is present at both SID and SOD the 8085 itself is assumed to be faulty. When the "NEXT" key is pressed execution of the test loop stops and the next test is started.

Two tests are performed next on the I/O facilities of the 8355/8755 and 8155 devices. The first of these is a simple write/read test on the parallel I/O ports which requires that test connectors be inserted into the sockets J3, J4 and J5 to connect corresponding bits of the A, B and C ports on each device together. Thus data written to port A of any of the chips can be read back through port B (and port C in the case of the 8155s) of the same chip. The test program writes a walking bit pattern to port A of each device, reading back through port B (and C), and then writes the walking bit pattern to the B ports, reading back through the A ports. Thus the input capabilities of each port are checked. If any errors are detected they must be due to a fault in the port because connections from the ports to J3, J4 and J5 were verified during Stage II, and the test connectors are assumed to be fault free.

The second of the two tests is for the counter/timer on the basic 8155 (A16) and the 8085 TRAP input, which is connected to the timer output of A16 on the SDK-85 printed circuit board. The timer is initialised to produce a pulse at its output after a short delay, then a delay routine is executed. If a TRAP interrupt has not occurred on exit from the delay routine an error flag is set.

If any errors are detected in either of the parallel I/O or TRAP tests an appropriate error message is displayed and both tests are repeated at one millisecond intervals. Repetition of the tests allows the timer output of A16 to be traced, to determine whether the cause of the TRAP failure was the timer in A16 or the 8085. As before, repetition of the tests stops when the "NEXT" key is pressed. The parallel I/O and TRAP tests are performed together, with results of both being displayed at once, so that A16 is "fully" tested at one time and the tests for its I/O ports and counter/timer are not separated unnecessarily.

After the parallel port and TRAP tests the second part of the Stage III procedure, in which the SDK-85 external interrupt and hold facilities are tested, commences. Up to this point in the SA procedure links 3-4, 7-8 and 20-21 on the SDK-85 board must have been in place, tying the RST 6.5, INTR and HOLD inputs to their inactive (low) levels. In the second part of Stage III the links are to be removed and these inputs are to be connected to an output port, so that they may be asserted under software control. These input lines are connected to inputs of an 8216 buffer which, being TTL compatible, float to the "high" state. Therefore if, in the process of removing the links and connecting the inputs to the appropriate output port, the inputs were allowed to float, the RST 6.5, INTR and HOLD Inputs to

the CPU would all be asserted and the SDK-85 would hang until the inputs were taken low again. In order that this does not occur the following procedure is adopted:

As port A of A15 is the one which is to be used to control the RST 6.5, INTR and HOLD inputs, at the end of the TRAP test the relevant output bits of this port are set to zero. A message is then written to the display informing the operator that the next test is about to be performed and prompting him to change the test connectors. He must first remove the connectors currently in sockets J3, J4 and J5, then insert all connectors for the remaining tests in Stage III (and, in particular, the one which connects the RST 6.5, INTR and HOLD inputs to Port A of A15) and finally remove links 3-4, 7-8 and 20-21. He must do this without removing power from the system or pressing "RESET". Only in this way will a continuous "low" level at the RST 6.5, INTR and HOLD inputs be maintained.

It may be noted that these precautions would not have been necessary if the three inputs concerned had been active low inputs, which would float to the inactive state when left disconnected.

The first test to be performed in the second part of Stage III is on the RST 6.5 facility and consists of the following steps:

(i) The RST 6.5 input to the 8085 is read to check that it is initially low.

(ii) Data is written to port A of A15 to take the external RST 6.5 input high and the RST 6.5 input to the 8085 is again read to verify that it then goes high.

(iii) 8085 interrupts are enabled and RST 6.5 is unmasked. A flag in RAM is then read to verify that a RST 6.5 interrupt has occurred.

If any of these tests fail an error message is displayed and the test is repeated continually until the "NEXT" key is pressed. Repetition of the test allows the pulse train generated at the external RST 6.5 input to be traced (with a CRO or logic probe) through to the 8085, so that the fault may be isolated to the 8085, the input buffer (A5), or an interconnection.

After the RST 6.5 test a message is written to the display which indicates the start of the vectored interrupt (RST 7.5) test. In response to this prompt the operator must press the "VECT INTR" key, which should produce an active transition at the 8085 RST 7.5 input. The test program checks for a transition at this input and, when one is detected, unmasks the RST 7.5 interrupt and enables 8085 interrupts. If a RST 7.5 interrupt does not then occur the 8085 must be faulty so an error message is displayed. If the active transition on RST 7.5 is not detected (that is, if nothing appears to happen when the "VECT INTR" key is pressed) the operator must press the "NEXT" key. An error message is then written to the display to emphasize that the test was illegally terminated, probably due to failure of the test. An oscilloscope or logic probe can then be used to determine whether the fault is in the "VECT INTR" key and its associated components or, failing that, in the 8085. When the "NEXT" key is pressed again the next test routine is entered.

The next test in Stage III is the INTR test. When the 8085 receives an INTR interrupt it issues the interrupt acknowledge strobe

(INTA/) and expects to read an instruction code (usually a restart instruction) back from the data bus. For the INTR test, a short interrupt service routine is stored in the test ROM at location 0018H. Therefore external buffers which place the RST 3 instruction code (DFH) onto the data bus when INTA/ is issued are required for this test. Appendix C contains the circuit diagram of a suitable adapter which is connected to the SDK-85 data bus at connector J1 and to INTA/ at connector J2. These connections are made to the board at the beginning of the second part of the Stage III procedure. If the RST 3 buffers are not available the "NEXT" key can be pressed in response to the display which marks the start of the test and the test will be skipped.

Execution of the INTR test commences when the "EXEC" key is pressed. Data is written to port A of A15 to take the INTR input high and 8085 interrupts are enabled. If a RST 3 interrupt does not then occur an error message is displayed and the test is repeated until the "NEXT" key is pressed. If a fault were to prevent the RST 3 instruction being placed on the data bus in response to INTA/, the most likely occurrence is that the 8085 would read all ones (FFH) from the bus and execute a RST 7 instruction. As a precaution against this causing the test program to fail, a RET instruction is stored in the test ROM as a RST 7 service routine, to allow an orderly recovery from failure of the test.

There is a relatively large amount of untested logic - including the data bus input buffers - involved in passing INTR to the 8085, INTA/ to the external connector and the RST 3 instruction back to the 8085. If the INTR test fails all of this logic must be tested with an oscilloscope while the test is being repeated so that the

fault may be isolated.

The INTR test is followed by a test for the counter/timer on the expansion RAM chip, A17. As A17 is an optional component the operator has the option of skipping the test by pressing the "NEXT" key in response to the message on the display which marks the start of the test. For this test a plug must be inserted into connector J5 to connect the timer output of A17 to bit 7 of its A port. Link 17-18, which connects CLK to the timer input of A17 must also be in place.

The test commences when the "EXEC" key is pressed, and consists of the following steps:

(i) The timer output is read (through port A of A17) and tested to verify that it is initially high.

(ii) The counter is initialised and a short delay routine is entered, during the execution of which the timer output must go low.

(iii) A second delay routine is entered, during which the timer output must return to the high level.

If any of these tests fail an error message is displayed and the test is repeated at one millisecond intervals until the "NEXT" key is pressed. A logic probe or oscilloscope may be used to trace the pulse train which should appear at the timer output. If the pulse train does not appear at the timer output A17 is assumed to be faulty.

The next step in the Stage III procedure is not so much a specific test as a routine designed to stimulate any external memory connected to the SDK-85 through the data and address buffers. With link 25-27 in place on the SDK-85 board the data bus input buffers are enabled whenever the CPU performs a read operation from memory addresses 8000H to FFFFH, or from I/O addresses 80H to FFH. Thus any memory or I/O devices in this address range are assumed to be external to the SDK-85. Since the nature of any such external memory or I/O will vary from system to system, no specific test could be performed to check it. Instead a general purpose stimulus routine is executed, in which the data bytes 00H and FFH are repetetively written to, then read back from each memory location in the range 8000H to FFFFH.

The response of external memory (if any) to this stimulus will, of course, depend on what type of memory it is, so the stimulus routine ignores the data read back. Some other means of verifying the response of the memory (such as signature analysis) must be used, and for this reason START and STOP pulses for the signature analyser are generated at unused outputs of the 8205 address decoder (A10) each time the routine is executed. No stimulus is provided explicitly for external I/O devices since stimulus data for I/O devices generally must be quite specific to elicit any meaningful response. A general purpose stimulus would therefore be unlikely to be very useful.

In an SDK-85 with no attachments this test can be ignored. It is stopped when the "NEXT" key is pressed.

The last test performed in Stage III is the HOLD test. A message which indicates that the test is about to be performed is

written to the display, then data is written to port A of A15 to assert the external HOLD Input. If the hold mechanism works correctly the 8085 should then enter a HOLD state from which it cannot exit. With the system hung in this way it is possible to test the logic associated with the hold acknowledge status line (HLDA) while it is in its active state. This section of the circuit is tested in Stage I, but only while HLDA is inactive so it is necessary to test it again while HLDA is active to ensure that there are no stuck-at-zero faults present (particularly around the HLDA synchronization flip-flop, A9).

If the hold mechanism fails and the system does not hang, execution of the test program continues and an error message is written to the display to indicate a HOLD fault. The CPU then halts. Once again a logic probe or oscilloscope may be used to trace the path of the HOLD input signal so that the source of the fault may be found. If the HOLD input pin of the 8085 is found to be asserted then the fault must lie in the 8085 itself.

This test completes the signature analysis procedure for the SDK-85.

3.4.5 Documentation

The documentation of a signature analysis test procedure for any system will clearly have a critical effect on the success with which the procedure is applied to the system. The SA literature presents several alternative methods of documentation which the designer may choose to use[92][95].

In the simplest method of documentation the "correct" signatures which are expected at each node in the system are printed adjacent to their respective nodes on the circuit diagram. It is then the task of the field service technician, while servicing the system, to observe signatures at whichever nodes he considers appropriate based on his knowledge of the system, and to identify the faulty component as the one with good input signatures and bad output signatures. This obviously requires some knowledge at least of the operation of each component in the system. Thus, to some extent it nullifies one of the major advantages claimed for signature analysis - that the service technician does not need to be highly trained or very familiar with the system to apply SA to it.

In complex systems, in which many signatures must be taken, possibly with many different control line setups, the method becomes impractical[92][95]. In such cases there is simply too much information to be legibly included on the circuit diagrams. Furthermore, it would be virtually impossible for a technician to approach the diagnosis of a large system systematically and efficiently without some overall guidelines on the order in which signatures should be taken. Therefore some more extensive form of documentation is required.

In a second method of documentation signatures are tabulated in the service manual of the system, while signals paths within the system are shown as arrows printed on the printed circuit board. In servicing the system the technician must follow the arrows, verifying signatures along the signal paths until the faulty device is found. This method also is not well suited to application in large complex systems because only a limited amount of information can be intell-

igibly printed on a printed circuit board. It is certainly not suitable as a means of documenting a retrofitted SA procedure because of the difficuoty of printing the required information on the printed circuit board.

In the third method of documentation a flow-chart is constructed which gives explicit directions to the service technician, specifying which signatures should be observed, how the signature analyser should be set up to observe them, and what action should be taken when an incorrect signature is observed. This method allows the designer to include much more detail in the documentation of the SA procedure than either of the other two methods. He can thus, with the benefit of an overview of the system, plan the SA procedure in detail so that diagnosis of the system will, in each case, proceed as quickly and efficiently as possible with little demand on the diagnostic skills of the service technician. Indeed, if the flow-chart is sufficiently detailed the diagnosis of a quite complex system can, in principle, be performed by a technician with absolutely no knowledge of the system. It must be acknowledged, however, that a flow-chart containing such detailed instructions must be quite complex. It is also clear that such a rigidly defined test procedure cannot be expected to deal with all possible fault conditions and will, in some cases, fail to correctly isolate the fault. Thus Sharritt[92] remarks that the flow-chart approach can be "risky and cumbersome" when a too rigid specification of the test procedure is attempted.

Nevertheless, in the case of the SDK-85 the flow-chart method of documentation is the only practical alternative. Because the procedure was designed to isolate faults to a single component and consequently involves the observation of a large number of signatures

with various control line setups, the documentation of the procedure must be quite detailed. In several places in the procedure it was necessary to include unconventional directions (such as to observe signatures with certain keys pressed) which could not easily be documented by one of the other methods. In Stage III, because the tests are so complex and varied, a flow-chart form of documentation is virtually mandatory. All of these constraints were evident at the start of the development of the SA procedure and it was therefore developed with the intention of documenting the procedure in flow-chart form.

The final form of documentation for the SDK-85 SA procedure is, in fact, based on an adaptation of the flow-chart method. The documentation consists of a numbered sequence of explicit instructions for each step of the procedure, listed in the general order in which the tests are to be performed. The instructions are generally followed in numerical sequence although depending on the outcome of the tests, the technician may be instructed to skip one or more steps or to replace a component and start the procedure again. This format was adopted in preference to the more conventional flow-chart because it allowed greater flexibility in the description of each step of the procedure, particularly in Stage III, in which the instructions for some tests are quite long. The principle disadvantage of this format is that it results in a very long set of instructions which a service technician may be reluctant to follow in detail from beginning to end. Nevertheless it was considered to be necessary, given that the test procedure itself, being intended to isolate faults to a single component, is quite long and detailed.

The main section of the documentation is divided into three sections corresponding to Stages I, II and III. This is preceded by a set of general notes which must be read before diagnosis of the system is started. These notes give general instructions on how the technician should proceed and how he should interpret the instructions for each of the three stages. They also contain specific notes concerning unstable signatures; the observation of signatures at device outputs and the possibility of bus faults; the need to observe signatures on both edges of some clock signals; and the fact that the procedure is not infallible.

The complete set of instructions for the procedure is contained in Appendix F.

3.5 Testing the SA Procedure

3.5.1 Verification

It was verified that each of the three stages in the SA procedure do test the systems as they were intended to during the development of the various individual tests, primarily with the aid of a logic analyser and an oscilloscope. For Stages II and III in particular the logic analyser was used to monitor activity in the system at critical sections of the test programs to ensure that the test software was correct.

During development of the three stages, and when the final set of "good" signatures was recorded, the procedure was performed several times, with particular attention being paid to signature consistency

and stability. As all signatures were found to be repeatable and (except in one case) stable, it was concluded that all instructions for the observation of signatures are valid. The one case of signature instability was attributed to violation of the data setup and hold times of the signature analyser due to propagation delays through a buffer. No suitable alternative method could be found for the observation of the signature at the node in question.

It was found that it initially took approximately 90 minutes to perform the entire SA procedure on the fault-free system. As might be expected, as familiarity with the procedure increased the time taken to perform the procedure decreased. However, it was not found to be possible to reduce the total time taken to less than 80 minutes - comprising 45 minutes for Stage I, 30 minutes for Stage II, and 5 minutes for Stage III (with no test performed at step 9 - the external memory test).

3.5.2 Application of the procedure to faulty systems

The only way in which a method of isolating faults in a system can be assessed in practice is by application of the method to a faulty system to see how quickly it does isolate the fault (if at all). Thus the effectiveness of the SDK-85 SA procedure was assessed by trial application to systems which were known to be faulty.

After the modifications described in Section 3.2.2.2 had been made to all available SDK-85's, only one faulty system was available. Therefore, so that the SA procedure could be more fully tested, it was necessary to introduce various faults into a working system and then apply the SA procedure to that. The procedure was designed explicitly to detect almost all possible stuck-at and open-circuit faults

faults on the SDK-85 board and, for any given fault in this class, it could be easily predicted whether the procedure would isolate the fault and, if so, at what stage. Therefore it was not considered to be worthwhile to introduce stuck-at and open-circuit faults into the system to test the effectiveness of the SA procedure.

The approach adopted instead was to selectively replace good components in the system with components which were known to be faulty (although the nature of the faults were not known) and then apply the SA procedure to try to isolate the faulty devices.

The results of the various trials of the SA procedure are described in the following sections.

3.5.2.1 Faulty SDK-85

The first trial application of the SA procedure was to an SDK-85 which contained an unknown fault. In normal operation the system was found to run satisfactorily for short periods but it was observed that the monitor would display its error message at apparently random times in response to valid key sequences.

Examination of the faulty system revealed that it was different from the one on which the SA procedure was developed. In particular, the display scanning logic was obviously different, employing an 8205 one-of-eight decoder instead of a 74LS156. Some of the other minor components were also different. The fact that these differences between the systems existed meant that the extent to which the developed SA procedure could be applied to the faulty system was limited. The procedure is obviously very specific to a particular circuit and even minor changes in the circuit may mean that a large part of the

SA procedure must be changed for it to be applicable. It was therefore considered to be impractical to revise the procedure to suit the faulty system. In any case, a circuit diagram for the system was not available so a revision of the procedure would have been virtually impossible.

It was possible, however, to perform the free-run test on the 8085 CPU because it only involves the CPU and is independent of system configuration. At step 3 in Stage I, pin 3 of the 8085 was found to be pulsing and, when monitored on an oscilloscope, proved to be producing a 3MHz square wave whereas it was expected to produce the static RESET OUT signal. Closer examination of the behaviour of the chip, and the printed circuit board, revealed that this particular 8085 had its CLK output at pin 3 and its RESET OUT output on pin 37 (that is, pins 3 and 37 were swapped around from the usual pin configuration for an 8085). The printed circuit board had obviously been designed to accept this particular version of the 8085. It can only be assumed that the 8085 was an early version of the device, while the system was a correspondingly early version of the SDK-85. No information could be found in the available Intel literature about a version of the 8085 with the functions of pins 3 and 37 interchanged.

The printed circuit board was modified to accept a "standard" 8085 and the system then appeared to work faultlessly. Because of the circuit differences between this and the original SDK-85 it was only possible to run the free-run test on the CPU and Stage III of the SA procedure. None of the tests showed any errors, although during the external memory test in Stage III the display flashed on and off, presumably as a result of the differences in the display scanning logic which were mentioned above. It appeared, then, that the fault

in the system was cured when the "old" 8085 was replaced with a "new" one, which suggests that the fault either was due to the layout of the printed circuit board (which was changed slightly for the new 8085) or was in the old 8085.

While this faulty SDK-85 proved to be a rather unsatisfactory test case for the SA procedure, it did serve to illustrate two important points. The first is that the SA procedure can detect "faults" in the system (in this case, a quite drastic "fault" in the CPU). The second is that the procedure, as developed, can only be applied to the exact system for which it was developed; even minor changes to the circuit may mean that the SA procedure must be completely revised. At the very least it would be necessary to revise the documentation and record an updated set of "good" signatures after a circuit change.

3.5.2.2 Faulty 8085

For the second trial application of the SA procedure, the CPU chip in the original SDK-85 (the one on which the procedure had been developed) was replaced with one which was known to contain a fault, although the nature of the fault was unknown. The only error then apparent in the operation of the system was that when it was first turned on the display was garbled, instead of showing the expected "- 8085" monitor sign-on message. After the RESET key was pressed once, the display would be correct and the system would operate normally.

It was observed that the problem was not overcome by holding the RESET key down as the system was powered up, so it appeared that the fault was not related to the length of time for which RESET was active. It was also observed that, although the display was initially

garbled, the monitor was apparently running correctly as RAM locations could be modified by the usual key sequences.

Application of Stages I and II of the SA procedure to the system revealed no errors at all. At the start of Stage III the display was again garbled immediately after power-up, instead of showing characters shifting across the display. If the instructions were to be followed at that point (step 2 of Stage III) the 8279 keyboard/display controller would be replaced because the display was incorrect. However, the fault was known to lie in the 8085 and not the 8279, so the SA procedure clearly failed to isolate the fault. If the 8279 had been replaced according to the instructions the fault would not have been cured and there would have been no indication of any likely alternative source of the fault.

After the RESET key was pressed the correct display sequence was produced and all remaining tests in Stage III executed without error.

It was decided to attempt to determine the nature of the fault in the 8085 so that the reason for the failure of the SA procedure to isolate the fault might be discovered. A logic state analyser was used to monitor the behaviour of the CPU (that is, to trace the sequence of addresses referenced by the CPU) immediately after power-on. This revealed that the CPU started executing code at address 0024H in the monitor before the instruction at address 0000H was fetched and executed. The TRAP interrupt of the 8085 forces the CPU to execute from address 0024H onwards, so it was apparent that the 8085 was servicing a TRAP interrupt as soon as it commenced execution after being reset at power-on. In the course of execution of the TRAP routine in the monitor, data is written to the 8279 control and data registers, so

in this case data was being written to the 8279 before it was properly initialised. This premature writing of data to the 8279 apparently caused the garbled display when the device was properly initialised and data was placed in its display RAM, after execution of the TRAP routine had been completed.

It was therefore concluded that the nature of the fault in the 8085 was that it was sensing a TRAP interrupt immediately after power-on reset. The fault was clearly not due to the timer in A16 because its TIMER OUT pin was observed to go high while RESET was active, and stay high while the 8085 was coming out of reset. It should be noted that, apart from the problem at power-on reset, the TRAP mechanism appeared to be functioning correctly, as implied by the error-free execution of the TRAP test at step 7 of Stage III, and by the correct operation of the monitor single-step function.

A possible interpretation of this fault is evident from the circuit diagram of the TRAP logic given in the 8085 data sheet and reproduced here as Figure 3.3. If the connection from the 8085 RESET input to the TRAP flip-flop clear input were open-circuited the flip-flop would not be cleared on reset and may, in fact, be set after power is first applied to the 8085. A TRAP interrupt would then be sensed by the CPU and serviced as soon as possible after reset, thereby clearing the TRAP flip-flop. A second reset would result in the correct execution sequence by the CPU, as the TRAP flip-flop would be clear.

It is interesting to note that this fault would only result in incorrect behaviour by the CPU immediately after power-up, or during normal execution if a reset were to occur in the short interval

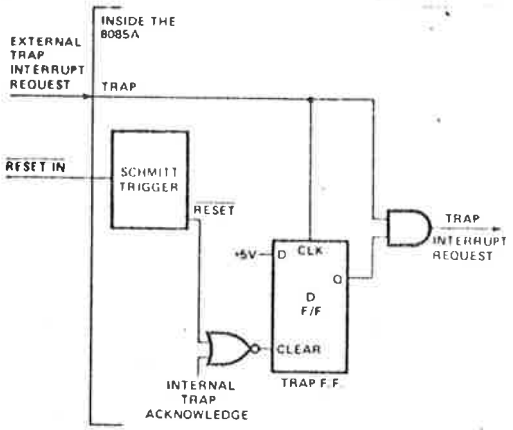


Figure 3.3. Internal TRAP Logic of the 8085.

between the TRAP flip-flop being set and the CPU servicing the interrupt. This explanation of the fault accounts for the observed behaviour of the system during Stage III of the SA procedure and, in particular, for the inability of the procedure to correctly isolate the fault. It is significant that a physical explanation of the fault could only be postulated because the circuit diagram of the relevant section of the 8085 happened to be available.

3.5.2.3 Faulty 8355

The second "known faulty" device to be used to test the SA procedure was an 8355 ROM containing version 1.2 of the SDK-85 monitor. Once again the nature of the fault was not known. The device was placed in socket A14 in the SDK-85, and no errors were observed in normal system operation.

The SA procedure was applied to the system with the result that no errors were found in any of the three stages of the procedure. It was therefore assumed both that the contents of each location in the ROM were correct and that the operation of the two I/O ports in both input and output modes was correct. However, it is possible that the device did contain a fault which only occurred when the I/O ports were configured in a particular way, or after a particular sequence of input data to the chip. If the device was indeed faulty, the fault was most likely to have been some such form of pattern sensitivity rather than a simple stuck-at, bridging, or open-circuit fault of the type tested for in Stages II and III. A second, seemingly less likely, possibility is that an error occurred during the tests but was not detected by the signature analyser because it produced the "good" signature. The probability of such an occurrence (less than .002%[96]) is small enough to be negligible.

3.5.2.4 Faulty 8155

An 8155 RAM device - also assumed to be faulty, but with the nature of the fault unknown - was placed in socket A16 on the SDK-85 board, in place of the "basic" RAM chip, with no apparent errors during normal system operation. The SA procedure was once again applied, with no errors being detected.

Like the 8355, then, if the 8155 was faulty, the fault was most likely to have been one which only occurred in a mode in which the 8155 was not tested, or which was some form of RAM pattern sensitivity. It should be noted that the 8155 is functionally more complex than the 8355, having several possible operating modes for its I/O ports and counters and yet its I/O section is no more thoroughly tested than that of the 8355. The possibility of the SA procedure not detecting a fault in an 8155 is correspondingly greater than for an 8355.

3.5.2.5 Faulty 8279

A faulty 8279 was inserted in place of the good 8279, with no errors evident during normal operation of the system. The SA procedure was applied, with only one error being observed. This occurred at step 2 of Stage III, in which characters are shifted from right to left across the display. It was observed that the bottom ("d") segment of the left-most display digit was turned on at certain times when it should not have been, so that a "4", for example, would appear as a "y". The error did not appear on any other digit and there was no obvious correlation between the occurrence of the error on this digit and the turning on of any other segment in any of the digits.

The SA procedure requires that the 8279 be replaced when an incorrect display is observed, so in this case the faulty device was correctly identified.

3.5.2.6 P.C.B. bridging fault

It is appropriate to consider at this point a fault which was not deliberately introduced to test the SA procedure, but which was discovered during development of the procedure. The effects of the fault were first observed during development of the Stage III test program (which was developed before the Stage II program), as unexpected behaviour of the system after the serial I/O test had been performed.

A logic state analyser was used to trace the activity of the CPU, monitoring the sequence of addresses placed on its address bus, with the negative edge of ALE used as clock. It was found that after port A of A15 was enabled as an output port for the first time (at the start of the parallel I/O test) the sequence of addresses output by the 8085 bore no relationship to the following instruction sequence. This implied that the 8085 had stopped executing instructions out of the ROM.

After some experimentation it was found that the cause of this behaviour was a short circuit on the printed circuit board between the ALE line and bit 6 of port A on A15. In the presence of the short circuit, with this bit of the port acting as an output, the 8085 could not drive its ALE output to valid logic levels and therefore could not fetch instructions from the ROM. The fault had no effect before the parallel I/O test because until that point in the test port A of A15 had only been enabled as an input port and bit 6 therefore would not have affected the ALE signal.

It is interesting to consider the effects which the fault would have had during normal execution of the SA procedure. Stage I would have revealed no errors because all ports remain in their initial (input) mode throughout Stage I. In Stage II, however, all ports are enabled as output ports by the first few instructions executed after reset, after which the system would start to misbehave. Since none of the Stage II test program would be correctly executed, START and STOP pulses for the

signature analyser would not be generated and no signatures could be observed. This would first be noticed at step 4 of Stage II, with a consequent direction to replace the 8085.

The SA procedure would therefore have failed to correctly isolate the fault and it would then be necessary to use conventional techniques to try to isolate it. This procedure would be made more difficult by the facts that the error only occurs when the port is programmed as an output port, and that the fault involves the ALE line, which was not directly involved in the observation of signatures at step 4 in Stage II. A technician would need a logic state analyser and some familiarity with the program being executed when the error first appeared (the Stage II test program) to isolate the fault.

3.6 Summary

The implementation of signature analysis on the SDK-85 described in this chapter was a task which required a great deal of attention to detail and many decisions and compromises. Some of this was foreshadowed in the SA literature, but much of it could only be appreciated after performing a detailed implementation. The compromises, which were mostly made necessary by practical considerations of the length of test programs and test execution time, clearly affected the effectiveness of the final procedure and tests carried out on the procedure have shown that there is some room for improvement.

The implications of the observations made in this chapter about the implementation and effectiveness of SA on the SDK-85 will be discussed in Chapter IV. In particular, the extent to which SA is seen to be the complete field service solution and the more serious areas of deficiency will be discussed.

CHAPTER IV

ASSESSMENT OF SIGNATURE ANALYSIS

4.1 Unique Properties of the SDK-85 Implementation

Before the effectiveness of signature analysis (SA) can be assessed on the basis of the implementation described in Chapter III, it is necessary to identify any unique characteristics of that implementation which may influence the assessment. In particular, the effects of any peculiarities of the target system and the approach to the implementation must be considered.

4.1.1 Peculiarities of the SDK-85

Although the SDK-85 was chosen as the target system for the implementation of SA largely because it was considered to be a typical microprocessor system, it does have the distinction of being a general purpose system with a wide range of I/O and interrupt facilities. A complete test for the system should test all of these facilities in all possible operating modes. In a dedicated system there would typically be fewer facilities, which would only ever be used in a restricted set of operating modes and which, therefore, would only need to be tested in those modes.

The need to test the various I/O facilities of the SDK-85 complicated the SA procedure to the extent of requiring a third stage in the procedure. However, Stage III and the various tests it contains have only served to illustrate,in one implementation, many of the problems which may be variously encountered in attempting to test the I/O facilities of a range of different systems.

A second unique characteristic of the SDK-85 is its cost. Being

a "bare-bones" system, supplied without power supply, elaborate packaging or front panel controls, it is somewhat cheaper than a fully packaged system of equivalent functionality would be. The SA development costs, extra hardware costs and overall repair costs will be a greater proportion of the value of the SDK-85 than of an equivalent packaged system. Therefore, the fact that the various pieces of test hardware required for the signature analysis of the SDK-85 are worth about twenty per cent of its total value (approximately $350) should not be taken as being significant. For a packaged system the proportionate cost would be significantly less.

4.1.2 Peculiarities of the SA procedure

Apart from the inclusion of Stage III the most important unique characteristic of the SDK-85 SA procedure is that it was designed to isolate faults down to a single replaceable component. This resulted in a procedure which is quite long (particularly Stages I and II) because it involves a lot of signal tracing from chip to chip. If a less ambitious goal for fault resolution had been set it would not have been necessary to test each component separately and the procedure could have been significantly shorter. Therefore it is fair to say that the SDK-85 implementation must exaggerate the problems and tedium of tracing signals from chip to chip and the slowness of the procedure.

The aim to achieve fault resolution to a single component also had the effect of reducing the extent to which half-splitting could be used, as discussed in Chapter III. Once again this resulted in a procedure which is longer than it would otherwise be. It can be said, however, that the restricted use of half-splitting in favour of an "expanding kernel" approach made the organisation of the documentation for the procedure much simpler. If half-splitting had been more ex-

tensively used there would have been many more branches, or decision points, in the documented procedure, which may have meant that the already lengthy documentation would have become unmanageably complex. It is likely that half-splitting could have been used to greater advantage on the SDK-85 than it is in the procedure described in Chapter III, but it is not expected that this would have significantly reduced the total time taken for the procedure.

A third characteristic of the procedure is that the thoroughness of the tests performed on individual devices in the system varies considerably. Whereas almost all nodes in the system are "wiggled" to test for stuck-at and bridging faults and are thereby tested quite thoroughly, some devices receive only a cursory test. It proved to be quite easy to test most nodes in this way, particularly during the free-run stage in which most system nodes are "wiggled" by the CPU, and yet the technique provides quite a substantial fault coverage. Node-wiggling also provides a very convenient and effective stimulus by which some devices, including the 8205 address decoder and the two ROMs, can be tested.

In contrast, it proved to be very difficult to fit a thorough test for the 8085 CPU and 8279 keyboard/display controller into the context of node-wiggling and, in the case of the 8279, it was necessary to devise an explicit test. Even more extensive tests for the 8085, 8279 and other devices could have been included in the procedure, but this would have been inconsistent with the "node-wiggling" approach to stimulating devices, and the suggestion in the "Designer's Guide to Signature Analysis"[95] that to test LSI devices one should simply "apply whatever stimulus is required to exercise them". The important point to note is that the "node-wiggling" form of stimulus advocated

in the SA literature does test some devices very thoroughly, but is completely unsuitable for others and therefore results in device tests which are not uniformly thorough.

It is clear, then, that there are several characteristics of the SDK-85 SA Implementation in respect of which it differs from what might be considered a truly typical implementation. However, it is not considered that any of these would prevent a fair assessment of SA in general based on the SDK-85 implementation, provided that their effects on the procedure are kept in mind.

4.2 Implications of the SDK-85 Implementation

It is now possible to consider the developments and tests described in Chapter III and their implications about SA and its general effectiveness as a field service technique for microprocessor systems. To do this, three aspects of the SDK-85 procedure will be considered. They are the problems involved in the implementation of the procedure on the SDK-85; the ease with which the procedure is applied; and the effectiveness of the procedure in isolating faults in the system.

4.2.1 Implementation problems

The outstanding feature of the process of designing the SA procedure was that it proved to be a much longer and more demanding task than expected. As discussed in Chapter III, it was necessary to perform several refinements of the procedure, resequencing tests to ensure that each device was tested as easily and efficiently as possible. A great deal of attention to detail (particularly signal timing) was required and in many cases it was necessary to exercise a degree of

inventiveness, to devise suitable tests for devices or to find suitable control signals for the signature analyser. In short, the design process was not straightforward.

However, in the more usual day-to-day implementation of SA two factors would act to make the design process faster and easier:

(i) Experience. As a designer gains experience in the application of SA he will be able to anticipate, to some extent, the best time and method for testing each component in the system. He will therefore create, at the first attempt, a SA procedure which is close to the optimum in terms of the use of half-splitting, the number of control line changes, and the number of signatures which must be observed. This "learning curve principle" is noted in[95].

(ii) Initial design. When SA is incorporated into a new system it is the system designer who is most likely to plan the SA procedure. His knowledge of system operation is likely to make it much easier for him to determine when and how each component should be tested. Secondly, in designing the system, he can incorporate features which will make the implementation of SA easier than it was for the SDK-85.

Because of these two factors and the unusual economics of the SDK-85 noted earlier in this chapter, the SDK-85 implementation of SA provides no reason to disagree with claims by Hewlett-Packard[54][99] that incorporation of SA into a product increases its development costs by only one per cent.

The second problem which was encountered during development of

the SA procedure was that there are several features of the SDK-85 design and architecture which made the use of SA to test some devices difficult or impossible. The first of these, and one which the SDK-85 shares will all microcomputers is the use of busses, as discussed in earlier chapters. Unless some form of current tracing is used it is impossible to isolate a bus fault to a single component on the bus. This is an inherent limitation on the ability of SA, or any other voltage sensing technique, to isolate faults. It means that if an incorrect signature is observed on a device output as the result of a fault elsewhere on the bus, the field service technician must use some other means of isolating the fault. To do this he needs a relatively high level of technical knowledge and some familiarity with the system under test. This problem therefore represents a significant restriction on the ability of SA to quickly and cheaply isolate faults, albeit one which is shared by most other techniques and is virtually inevitable.

The second feature of the SDK-85 design which made the implementation of SA difficult was the inclusion of circuitry which exhibits non-standard (that is, non-TTL) logic levels. Voltage levels present in the serial interface and display driving circuits are incompatible with the input logic levels of the signature analyser, so signatures simply cannot be observed in those sections of the circuit. An oscilloscope must be used to trace any faults which may occur there. Although the circuits involved are quite simple and can be satisfactorily (perhaps even more easily) diagnosed with an oscilloscope, it does mean that once again a skilled technician with some knowledge of the system must use an alternative method to SA.

If these sections of the circuit had contained any sequential logic or feedback, then diagnosis with an oscilloscope would be much less straightforward and signature analysis may have offered some advantages. The implication for system design is clear: for the most straightforward application of SA to a system, all system nodes should exhibit standard logic levels and, in interface circuits, in which this is not possible, feedback should be avoided.

An obvious basic requirement for the application of SA to any system is that there be suitable START, STOP and CLOCK signals available for the observation of the signatures of all signals in the system. These control signals proved to be quite easy to find for most of the signals present in the SDK-85, most data in the system being synchronous with ALE, RD/ or WR/. However for some lines (notably ALE and CLK) there was no suitable signal of higher frequency available which could be used as a clock. In these cases it was necessary to use the signal being observed as clock, and observe signatures on both positive and negative edges of the clock - an inconvenient but satisfactory method. The need to use several different clock signals at various times and the consequent need to change signature analyser control lines also proved to be inconvenient and both slowed down the procedure and restricted the use of half-splitting, as discussed in Chapter III. Clearly, the multiplicity of clocks, or strobe signals, in the SDK-85 was, in itself, a problem.

Ideally, all signatures in a system would be observable with just one clock signal so that few control line changes would be required. Signatures could then be observed throughout the system in any convenient order and a faster SA procedure would result. Although

the SDK-85 is a synchronous system, with all signals being derived ultimately from the 6.144MHz oscillator, the relevent data sheets[116] do not specify a fixed time relationship between this clock (or its derivative, CLK) and other signals in the system. Therefore, while the 6.144MHz clock could, in principle, be used to sample data at all nodes in the system, it is not possible in practice and several different clocks must be used.

A related problem was the inability to observe signatures at the segment outputs of the 8279 keyboard/display controller because a suitable clock signal was not available. Again, in principle, CLK could have been used for this purpose, but in practice unstable signatures resulted. The segment latch outputs change synchronously with the SLO scan line output, which is derived from CLK, but the signature instability would suggest that there is a long and somewhat variable propagation delay in the clock divider chain. In fact, the "Designer's Guide to Signature Analysis"[95] remarks specifically on the difficulties of observing signatures around I.C. keyboard encoders (such as the 8279) and the possibility of signature instability when testing long ripple counter chains. The inability to observe all outputs of the 8279 meant that the device could not be directly tested as fully as it should have been. This type of restriction clearly compromises the effectiveness of SA although in this case there is some compensation in the fact that the display can be examined to detect errors in the segment output data of the 8279.

Finally, two features of the SDK-85 design created minor problems which could easily have been avoided and would have, had the system been designed with SA in mind. When observing the outputs of

the 74LS156 demultiplexer it is necessary to use a pull-up resistor on the data probe of the signature analyser, so that the open collector outputs appear to be in the high state when not active. This is only a minor inconvenience, but it is one which would have been easily overcome by the inclusion of pull-up resistors on the 74LS156 outputs in the original design.

Secondly, the fact that the external RST 6.5, INTR and HOLD inputs to the system are active high proved to be very inconvenient when it was necessary to change test connectors during Stage III. Had these inputs been active low they would have floated to their inactive (high) level when disconnected and there would have been no need for special precautions when the connectors were changed. Indeed, if the inputs were active low there would have been no need for the links on the board which tie the inputs to their inactive level when not in use.

These last two points serve to illustrate that for successful, trouble-free implementation of SA in a system the design of the system to accommodate SA must be considered in great detail. This is particularly so if fault resolution to a single component is to be attempted. Certainly it is necessary to consider design of the system for SA in greater detail than is implied in the SA application literature.

4.2.2 Ease of use of the SA procedure

The most striking aspect of the actual use of the SA procedure is the time taken to complete it. Application of the procedure to a fault-free SDK-85 takes at least 80 minutes, of which 45 minutes is occupied by Stage I, 30 minutes by Stage II and 5 minutes by Stage III.

If a fault were to be found in the system this time would obviously increase, particularly if it were one which had to be isolated with an oscilloscope or current probe.

The impression which was gained while performing Stages I and II of the procedure was that it involves a lot of slow, tedious tracing of signals from one chip to another, with a lot of time being spent simply locating specific device pins. There seemed to be few changes in signature analyser control line setups, and these did not seem to occupy a significant amount of time overall. The amount of device-to-device probing could be reduced either by redesign of the procedure so that all signatures for each device are observed at once or by the greater use of half-splitting. However, as discussed earlier, either of these changes to the procedure would dramatically increase the number of control line changes required and, consequently, the time taken for the procedure would not be reduced significantly, if at all. Only if most signatures in the system could be observed with a single control line setup would either of these changes offer any significant improvement over the existing "expanding kernel" procedure. Half-splitting, as noted in [95], is most suitable for systems in which there is a clear signal propagation path, with little feedback, which is certainly not the case in the SDK-85.

It should be noted, once again, that the slowness of the procedure and the amount of device-to-device probing involved are largely a result of the fact that the procedure was designed to isolate faults to a single component. In other implementations they may be less of a problem.

It is significant that Stage III, in which several relatively complex tests are conducted, takes much less time to perform than either of the other two stages. This is entirely a result of the fact that in Stage III the response to each test is monitored under software control by the system itself. This is, of course, much faster than manual observation of the test results by signature analysis (or any other method). It is clear, then, that as a general practice as much of the system as possible should be tested by self-diagnosis, with signature analysis used only to test those sections of the system (CPU, ROM, RAM, I/O) which must be tested before self-diagnosis can take over. This means that if a system does not have some form of output display or input switches which can be used for interaction with the operator during self-diagnosis, they should be designed into the system specifically for this purpose.

Because Stage III executes so quickly it might be considered worthwhile to use it as a quick (but not conclusive) system verification routine. However if any errors were to be detected during Stage III it would then be necessary to apply the full SA procedure, starting with Stage I, to isolate the fault because the results of the Stage III tests depend on the error-free execution of Stages I and II. That is, an error during Stsge III may be the result of a fault which would be detected in one of the earlier stages. Therefore, while Stage III may be used as a convenient self-check routine it provides little useful diagnostic information if executed alone.

The eighty minutes taken for the execution of the SA procedure, while seeming to be a long time, should be considered in the context of almost complete fault resolution. During those eighty minutes almost every component and node in the system is tested in some way, so

that if no errors are detected the level of confidence in the system under test should be quite high. If a less thorough test were to be performed (for example, only a major system components) the time taken for the test would be much less, but at the cost of reduced fault resolution.

It is unlikely that the SDK-85 could be tested by traditional field service techniques as thoroughly as by the SA procedure in less than eighty minutes. For example, it would probably take well in excess of eighty minutes (not including the time taken to become familiar with the system) to use a logic stage analyser to isolate an obscure but commonplace fault such as a stuck-at bit in a RAM. In such a case the time actually taken to isolate the fault would depend on several factors, including luck. However it would certainly take a very long time to test the system such that all faults which the SA procedure can detect, would be detected. This speed advantage of signature analysis is due to the fact that the compression of data and extensive documentation of "good" signatures make verification of even complex data sequences very easy and fast.

A second aspect of the ease with which the SA procedure may be used is the degree of technical skill and system familiarity required. One of the major advantages claimed for SA is that the operator does not need a high level of technical knowledge or familiarity with the system under test to diagnose faults in the system. In fact, because the documentation for the SDK-85 SA procedure is quite detailed, virtually no knowledge of the system is required to apply the procedure in most cases. Generally it is only necessary to be able to locate the various devices and connectors on the board and identify which pin must be probed to observe a signature. This clearly requires

very little technical knowledge.

However, if a fault exists in the system which the SA procedure cannot isolate then the demands made upon the field service technician are somewhat greater. If it becomes necessary to use a current probe to isolate a bus fault, or to use an oscilloscope to trace a fault in the serial interface or display circuits, or if the procedure identifies a wrong component as being faulty, then the technician must use some technical skill to isolate the fault. Only in the last case - the complete failure of the SA procedure to correctly isolate the fault - is a significant degree of familiarity with the system required, because the technician must then use traditional techniques to isolate the fault "from scratch". In such a case it may, in fact, prove to be more economical to simply replace the board and have it repaired at a central site - that is, employ board-swapping on a limited scale for cases in which the SA procedure fails.

For the most part, then, it is true that neither technical skill nor familiarity with the system are required to apply the SA procedure to the SDK-85. For some faults a degree of technical knowledge is necessary, and in extreme (and hopefully rare) cases familiarity with the details of operation of the system may be required. It is significant that an understanding of system software would only be necessary if the SA procedure failed completely.

It can be concluded, then, that although the application of SA to the SDK-85 is not a fast process, it will in general isolate faults faster than traditional methods. Furthermore, SA generally requires that much less time be spent becoming familiar with the system and can be used by less skilled personnel than traditional

methods. However, it is still a labour-intensive and therefore expensive process.

4.2.3 Effectiveness of the SA procedure

The effectiveness of the SA procedure in the SDK-85 can only be assessed in terms of the ability of the procedure to isolate faults in the system. This assessment can only be based on the few trials of the procedure which were described in Chapter III, and on any general difficulties or limitations which may be foreseen.

The difficulties presented by the bus oriented architecture of the SDK-85, the use of discrete components, and the inability to observe signatures at certain nodes, were all discussed in Section 4.2.1. These problems obviously place a limitation on the ability of SA to isolate faults to the component level and, in that sense, limit the effectiveness of SA in the SDK-85. However, the limitations of the procedure are probably better illustrated by the trial applications of the procedure.

In the case of the 8085 in which the TRAP mechanism appeared to be faulty it is apparent that if the conclusion reached about the exact nature of the fault (that the TRAP flip-flop is not cleared on reset) is correct, the fault would only ever be evident after the CPU is reset. This fault, or any other fault only evident at reset, could not be directly detected by the SA procedure because the procedure only tests devices in what might be called "steady state" operation. Signature analysis is usually, and most conveniently, applied to observe the results of tests which are performed repetitively[95] and therefore would not detect errors which only occur during transient or "once-only" activity of the system, such as at reset.

It is possible to observe signatures of once only events using the "HOLD" facility of the signature analyser, and activity at all nodes in the system immediately after reset could be monitored if the SA procedure were so designed. The problem with this approach is that there are countless faults which could take effect at reset, and during other special states of the system and it would be extremely difficult to arrange the SA procedure to try to test for them all. Furthermore, it may be noted that the behaviour of the 8085 immediately after reset was found to be somewhat erratic and, in any case, is not fully documented by the manufacturers. Therefore while in principle it is possible to verify system behaviour after reset by signature analysis, it is not so in practice because the behaviour of the 8085 must be regarded as being variable from device to device and from time to time. The 8085 TRAP fault is therefore one which, by its very nature, could not be directly isolated by signature analysis.

With the faulty 8085 in the system the SA procedure ultimately identified the 8279 as being the faulty component. This constituted a complete failure of the SA procedure and, to isolate the fault, the only alternative left would have been to resort to traditional methods. This situation could have been avoided if for the given error, as well as identifying the component which was most likely to be faulty (the 8279), the documentation specified an alternative device which could be faulty and cause the error. Then, when replacement of the 8279 did not cure the fault, the alternative device (most likely to be the 8085 in this case) would have been replaced and the procedure would have then correctly isolated the fault. It seems desirable, then, that the documentation for a SA procedure include at least one alternative whenever a faulty device is identified after an error is observed, particularly when the reasons for selecting that device as the faulty one in

the first place are not very strong.

The fact that the 8279, rather than the 8085, was identified as being the faulty component reflects the assumption that, at that stage in the procedure (early in Stage III), the 8085 has been tested satisfactorily and is fault-free. The assumption that the CPU is fault-free after it has passed the free-run test is basic to the whole SA procedure, as described in the introductory literature. All tests performed after the free-run stage are software driven and therefore rely upon the integrity of the CPU. In the case in point the test for the 8279 failed simply because the 8085 did not initialise the device properly when it was assumed that it would. It is therefore very important that the CPU (along with other kernel components) is adequately tested before the software driven test stages. Any fault in the CPU which goes undetected may well prevent it from correctly performing software driven tests, in which case the SA procedure is likely to fail by identifying the worng component. In view of this conclusion it is doubtful whether the simple free-run test performed on the CPU, being the only explicit CPU test, is thorough enough.

In the SDK-85 SA procedure the free-run test is supplemented by several quick tests performed during Stage II. While the CPU is executing the loop to perform the RAM and output port tests the signatures which are taken to verify the WR/, SO and S1 outputs also verify that the CPU is executing the test program correctly - that is, peforming the correct number of the correct type of machine cycles. Similarly, when the stuck-at-one signature with RD/ as clock is observed during each RAM test, it is verified that the CPU is executing the correct number of read cycles during the test. In the sense that the CPU is

performing a variety of operations during these tests, they constitute a much better (although certainly not complete) test of the CPU than the free-run test. These simple tests therefore reduce the likelihood of a fault in the CPU going undetected. They do not, however, test the CPU in well defined or systematic way, so it is difficult to say by how much they reduce this likelihood.

The failure of the SA procedure to detect any faults in the 8155 and 8355 devices which were assumed to be faulty illustrates that the results obtained from the procedure can be inconclusive. While it is likely that neither of these devices was, in fact, faulty, there remains the possibility that one (or both) contained a fault was simply not detected by the tests performed. Faults which may not have been detected include:

(i) faults only evident in operating modes of the I/O ports which were not tested;

(ii) faults only evident under transient conditions (such as reset);

(iii) RAM pattern sensitivity (in the case of the 8155);

(iv) intermittent faults;

(v) a.c. or d.c. parametric faults.

There is the further possibility, that the only reason no faults were detected in the devices was that the faults happened to produce correct signatures. The important points arising out of this discussion are that neither the 8155 nor the 8355 is completely tested and that faults do exist which would not be detected by the SA procedure.

Although the 8155 and 8355 tests are incomplete and arguably unsatisfactory, it is important to realise that tests on the I/O sections of these devices, and I/O devices in general, are not as important as thorough tests on the CPU. As we have seen, the successful execution of tests in the later stages of the SA procedure depends on the integrity of the CPU, and any undetected faults in the CPU may cause the procedure to fail. Input/output devices, however, are not as important to the procedure. Although some tests in Stage III make use of I/O ports, there are only a few of these tests and they are documented in such a way that if any of them fails because an I/O port is faulty, the fault can be traced back to that port. Therefore, while incomplete testing of an I/O device may result in a fault in that device going undetected, it is unlikely that another device will be judged to be faulty because of it.

The one successful application of the SA procedure was in the case of the faulty 8279, which lit the bottom segment of the leftmost display digit at the wrong times. Although the procedure did correctly isolate the fault the success in this case was not particularly reassuring. The fault was clearly one which was pattern sensitive in some way, because it only occurred when certain characters were being displayed and did not show up at all during normal operation of the system. It must therefore be considered very fortunate that the fault happened to show up with the particular display sequence used for the test. If another sequence had been chosen the fault may not have been detected. If it had been possible to observe the segment outputs of the 8279 during Stage II the fault may have been detected then, but again there is no guarantee that the data in the display RAM during that test would have shown the fault up. This case once again suggests that thorough device tests, rather than

simple device exercise routines which might by chance show up a fault, are desirable.

The bridging fault between ALE and bit 6 of port A of the expansion ROM, described in Chapter III, is an example of a type of fault with which signature analysis cannot deal in practice. The fault would have caused complete failure of the SA procedure (if it had been applied) because it created a feedback path from port A to the CPU which eventually caused the CPU to stop executing the test program.

It has been noted several times that the successful application of SA to a system requires that all feedback paths from untested logic to a device under test be opened. For this reason all tests performed in Stages I, II and III were carefully sequenced so that no device is tested before devices which affect its inputs. However, in the design of a SA procedure it is not possible to allow for the existence of extra feedback paths, such as the one created by this bridging fault, simply because there are so many possibilities. When such faults occur it is therefore likely that they will invalidate the procedure. The property of bridging faults that they can create feedback paths is well known in the theory of testing combinational circuits, in which they can cause sequential behaviour and complicate test procedures considerably[7][121]. The principle is the same in the case of the bridging fault in the SDK-85.

The failure of the SA procedure in the presence of this fault would be complicated by the fact that it would first be noticed while performing a test which is nominally for the 8085 WR/ output. Replacement of the 8085, in accordance with the instructions for the procedure, would not cure the fault so it would be necessary to use tradit-

ional techniques to isolate the fault from that point. The fact that the fault would only be evident when port A of A15 was programmed as an output port would make this process much more difficult. Indeed, unless the technician was familiar with the Stage II test program he may even be unable to reproduce the error. Any fault like this, which would prevent the test procedure from being executed correctly and which therefore would produce misleading symptoms when the procedure was applied, would be particularly difficult to diagnose.

This fault illustrates that the only way to ensure that all possible faults in a system are detected and correctly diagnosed is to test the system in each of its possible operating states. If port A of A15 had been enabled as an output port during Stage I the fault would have been detected when the ALE signal path was checked, and could have been isolated with a current tracing tool. However, it is simply not practical to repeat all of the Stage I tests with the system in each of its possible operating configurations so that faults such as this would be detected. As discussed in Chapters I and II, it is not practical to test any LSI based system in all of its possible operating states, so faults such as this bridging fault, which cannot be allowed for in the design of the SA procedure, must be accepted as threats to the effectiveness of the procedure.

4.3 The Importance of the Deficiencies of SA

The strengths and shortcomings of SA observed in the SDK-85 implementation were discussed in the preceding section. Some of the observed deficiencies (such as the apparent tedium of performing

Stage I of the procedure) are not of great consequence, whereas others are more serious and significantly reduce the overall effectiveness of the technique. In this section the more serious deficiencies of SA which were evident from the SDK-85 implementation and their significance will be discussed, together with possible developments which may overcome the deficiencies.

It must be stressed that the properties which will be discussed are more concerned with the means of implementing SA than with SA *per se*. It is clear that SA will only be fully effective in a system if all implementation problems can be overcome, and it is apparent from the discussion in the first part of this chapter that implementation problems provide greatest limitation on the effectiveness of SA.

4.3.1 The observability of signatures at all nodes

The ability to observe signatures at all nodes in a system is an obvious requirement for the successful application of SA to the system, particularly when it is desired to test each component individually and thereby achieve complete fault resolution. It was apparent during development of the SDK-85 SA procedure that to achieve this it is necessary to pay a great deal of attention to details of system design and the characteristics of system components - more so than had been expected after reading the early SA application literature.

In the SDK-85 implementation two properties were found to limit the observability of signatures: voltage levels and timing. If signatures are to be observed at all nodes, logic levels at the nodes must be compatible with signature analyser input levels, so all nodes in a system should ideally exhibit standard TTL or CMOS logic levels.

While the new HP5005A signature multimeter[113] does have adjustable input threshold levels, it is obviously not practical to adjust the threshold to suit each node. For the reasons discussed in Section 4.2.1 the use of discrete components should be avoided, particularly in sections of a circuit which include feedback. Given the ongoing trend away from the use of discrete components, it seems likely that problems of this sort will become less common as a matter of course, rather than as a result of any trend towards design for testability.

The issue of timing and synchronization in systems is more complex. The basic requirement is that there be suitable start, stop and clock signals for the observation of data at every node in the system. In addition, it has been seen that it is desirable that there be as few different control line setups for the signature analyser as possible. In fact there would ideally be just one clock signal which could be used for the observation of all signatures in a system. In the SDK-85, as we have seen, this is possible in principle but not in practice because timing specifications for the system are not sufficiently precise. Thus it was necessary to use several different clock signals at various times, including CLK, ALE, RD/, WR/ and SLO.

Ideally, in every system there would be one signal which is twice the frequency of the highest-frequency signal of interest in the system. This could then be used as the signature analyser clock and either edge of that clock could be used to observe each signal on the board. Because its frequency is twice that of the highest-frequency signal of interest, either clock edge can be used to sample that signal in both logic states, removing the need to observe signatures on both clock edges, as was necessary for CLK, ALE, RD/ and WR/ in the SDK-85. Thus, in the SDK-85 the 6.144MHz clock would ideally be

used to observe all signatures in the system (if the timing relationship between it and the other signals in the system were well enough defined and propagation delays were not large enough to cause signature instability).

As an illustration of this principle, consider the segment outputs of the 8279 keyboard/display controller, which could not be observed because no suitable clock was available. These outputs change synchronously with the SLO output, which is derived from the CLK input by division by the programmable clock prescale factor (which is normally set to 31), then by a further factor of 128. The most suitable clock for the observation of both the segment and the scan line outputs would be one which is twice the frequency of SLO (or CLK ÷ (prescale factor) ÷ 64) as discussed above.

To verify that the availability of such a clock would enable the 8279 to be more easily tested, the clock prescaler for the 8279 during Stage II was changed to 16 (the nearest possible power of two to 31) as in the listing in Appendix E. Thus SLO became the CLK input divided by a factor of 2^{11}. A 4040 CMOS divider chip was connected to the system to provide a "CLK ÷ 2^{10}" clock signal, which was twice the frequency of SLO, but with a variable phase relationship to it, as shown in Figure 4.1. With this signal used as the signature analyser clock, and with SL3 used for the START and STOP inputs, it was found that stable signatures were observed at all scan line outputs, segment outputs, outputs of the 74LS156 and return line inputs to the 8279 with <u>either</u> clock edge (although it was sometimes necessary to reset the system to adjust the phase relationship between the clock and SLO). Thus it became much easier to test the keyboard and display scanning circuit, because it was only necessary to observe signatures

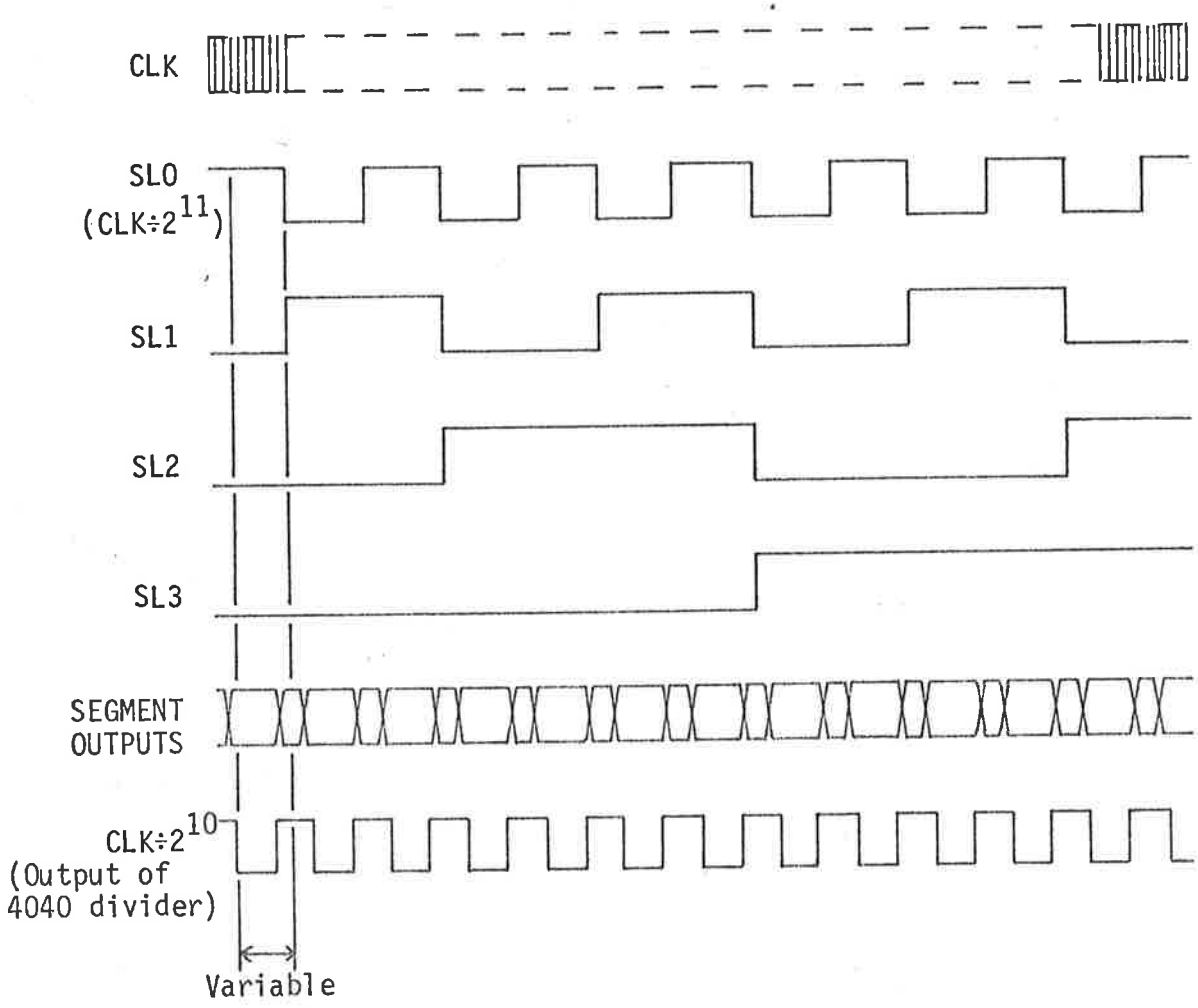


Figure 4.1. Provision of a Signature Analyser Clock for 8279 Scan Line and Segment Outputs.

on one clock edge and the 8279 segment outputs could be verified directly.

Although the extra hardware required to achieve this improvement in the testability of the system was quite simple, the principle it demonstrates is important. The existence of precisely defined synchronism in all sections of a system will considerably improve its testability by signature analysis. It is obviously preferable to be able to use a single clock for the observation of signatures at all nodes in the system, but in sections of the system in which prepagation delays may make this impractical, a "local" clock should be provided - with extra hardware if necessary. In the case of the SDK-85 this would involve the addition of one component (the 4040 divider - itself easily tested) to allow the 8279 to be more easily tested.

A much better general solution to the problems of testing the 8279 would be the provision by the manufacturer of the required clock signal at an output pin of the device itself. Then an 8279 in any system could be tested by signature analysis without any external hardware. However, I/O pins on LSI and VLSI devices are a precious resource[25] and manufacturers are understandably reluctant to dedicate any to improving the testability of the device, particularly if it only improves the testability for one specific method such as SA.

Nevertheless, complete synchronism within a system is essential if SA is to be easily applied and fully effective. The better the synchronism is defined (that is, the more complete the timing specifications are) the easier it will be to implement SA. While this would ideally be considered by device manufacturers, who would pro-

duce devices and specifications accordingly, it would be unrealistic to expect this to become common practice in the foreseeable future. The system designer will therefore be constrained to use devices which do present problems for SA and he must be prepared to include extra hardware in the system to overcome specific problems. This will require that he consider the problems of timing and synchronization in detail, and even (as was necessary with the 8279) experiment with devices to find suitable signature analyser setups with which they can be tested.

The problem of signature observability at all nodes in the system is therefore one which can be largely overcome with sufficient attention to design of the system in the first place. However, it is apparent that in some cases an inordinate amount of effort will be required to overcome specific problems. Certainly it is clear that the design of a system to include SA involves a good deal more than the provision of test programs and means of free-running.

4.3.2 Individual device tests

Perhaps the most outstanding result to come from the SDK-85 implementation of SA is that the tests performed during the SA procedure on individual LSI devices (the 8085, 8155, 8355 and 8279) are not thorough enough. The trials which were performed with the faulty 8085, 8155 and 8355 all demonstrate that the tests performed on these devices during the SA procedure are inadequate.

It was noted in Chapter III that the aim in developing the tests for each of these devices (except the 8085, for which the method of testing is prescribed in the SA literature) was to exercise or stimulate the device in some manner, consistent with the "node-wiggling"

philosophy underlying the approach to SA implementation. For simple SSI and MSI devices in the system, such as the 8205 address decoder, this produced a virtually complete test. It also proved to be possible to easily and thoroughly test the ROM section of the 8355/8755 and the RAM section of the 8155. However, the 8085 and 8279, being functionally more complex devices, were much less thoroughly tested. In other words, the "reasonably convenient" tests performed during the SA procedure tend to test devices up to a certain fixed level of functionality. This is a result of the fact that the effort which was put into deriving tests for each device was deliberately limited. It is clear, then, that node-wiggling" is an excellent stimulus for simple (especially combinational) devices, but is a poor stimulus for complex LSI devices.

As we have seen, it is particularly important that the CPU be adequately tested early in the SA procedure because later tests depend on it and assume that it is fault free. This was illustrated by the effect of the TRAP fault in the 8085. However, SA guidelines only provide for a very simple test of the CPU. The free-run test only tests the CPU during esecution of one instruction (NOP), which is probably the simplest instruction in its repertoire. Even if the free-run signatures are correct there is no guarantee that execution of the NOP instruction does not have any side effects within the CPU. There are countless possible CPU faults which would not be detected by the free-run test, of which the TRAP fault described earlier is just one. The supplementary tests performed on the CPU during Stage II would detect many of these faults, but still only constitute a partial test of the CPU. There is a clear need for a specific, systematic and thorough CPU test early in the SA procedure, which would test all or most of the CPU facilities in a uniform manner. Only

then could subsequent tests be performed with a high level of confidence.

The 8155 and 8355/8755 tests are certainly more complete than the free-run test for the 8085. Ignoring the possibility of an error going undetected because of the data compression performed by the signature analyser, the ROM section of each 8355/8755 is fully tested, and the 8155 RAM test is reasonably thorough. The I/O facilities of the devices are also tested quite extensively. However, the tests are not sufficiently thorough that it is not possible to postulate plausible faults which would not be detected by the tests. Here again there is a need for systematic and thorough tests for these devices which would test all of the facilities of the devices for the presence of any plausible fault.

The 8279 is functionally much more complex than either the 8155 or the 8355/8755 and the tests performed on it are the most extensive in the SDK-85 SA procedure. However the 8279 test is much less complete than the tests for the 8155 and 8355/8755. Although several tests are performed on the 8279 during Stages II and III, it is only operated in two of its several possible modes and many of its facilities are not tested at all. It is exercised, but by no means thoroughly. The mode in which it is tested most extensively is the one in which it is operated by the SDK-85 monitor and is therefore the only one in which it is likely to operate in this system. Thus, if the device were to be fully tested in that one mode it might be considered to be acceptably tested. This type of limited function testing has been used in the ATE environment to keep device tests down to a reasonable length[30][41]. Ideally, however, the device should be tested in all possible operating modes, particularly in a general purpose system

such as the SDK-85.

As noted in Chapter III the 8279 is a member of a rapidly growing family of intelligent microprocessor peripheral devices. In fact it is one of the simpler devices of this type. All of these devices must be pre-programmed, or initialised (with quite lengthy data sequences in some cases) before they will perform any meaningful function and therefore cannot be tested by a simple "node-wiggling" stimulus. An explicit programmed test for each device is necessary. The difficulty of testing these devices by simple node-wiggling may be compared with the difficulties, noted by McLeod[63] of testing LSI devices with random test sets. As these devices become more popular, and it becomes common to find several in one system, it will become essential to adequately test each device so that the system as a whole can be effectively diagnosed. Therefore, once again, guidelines for developing systematic and thorough programmed tests for peripheral devices are necessary.

The conclusion to be drawn from this discussion is clear. The effectiveness of SA in any system depends largely on the thoroughness of tests performed on individual devices and in this respect current SA guidelines are inadequate. The general node-wiggling approach to stimulating the system is satisfactory for simple devices, but not for complex programmable LSI devices. For these it is necessary to develop systematic programmed test routines which will test the devices thoroughly and uniformly. As yet, no guidelines for the development of such tests in the context of signature analysis exist.

4.3.3 Input/output device tests

The issue of how I/O devices and facilities in a microprocessor system can be tested is closely related to, but separate from, that of LSI device tests, discussed in the preceding section. The particular difficulty in testing I/O facilities is that a suitable external stimulus must be applied to the device under test, and its external outputs must be verified.

In the case of the SDK-85 it proved to be quite a simple matter to test most of the I/O facilities. For the parallel and serial I/O port tests external connectors are used to loop outputs back to inputs and it is simply verified that data written to the output ports can be read back correctly through the input ports. The parallel ports are tested in one mode only with a "walking bit" pattern but there is no reason (apart from test execution times and ROM space requirements) why the tests could not be more extensive. Even the more complex I/O facilities of the SDK-85, including timers and interrupts, are easily tested, with parallel ports used to simulate input signals and verify outputs. It should be noted, however, that each of these tests requires some external hardware, generally consisting of a simple loop-back connector, although external buffers are required for the INTR test. The 8279 test is an exception in that it does not require any external hardware. The "loop-back" of output data (the display) to inputs (the keyboard) is performed by the operator.

The ease with which tests were developed for the SDK-85 I/O facilities is largely a result of the fact that the CPU has simple, almost direct, control over the outputs and can almost directly observe inputs. The parallel and serial I/O ports in the SDK-85 do

not process the data passed between the CPU and the external world, so that data appearing at output lines is essentially identical to that written to the port by the CPU, and data read by the CPU is identical to that appearing at input lines. It is therefore a simple matter for the CPU to set output lines to specific logic levels and, with the aid of a loop-back plug to an input port, examine and verify the data present on those lines. The principle also applies to the 8279, although this device does perform some code conversion and multiplexing. The human element in the loop-back path helps overcome this difficulty by performing intelligent interpretation of the 8279 output data (that is, the displays).

Another important characteristic of the I/O facilities on the SDK-85 is that output line changes only occur when initiated by the CPU, and similarly, the CPU can observe all input line changes as they occur. Again, the multiplexing action of the 8279 is an exception to this rule.

These characteristics may be contrasted with those of I/O devices which more extensively process data passed between the CPU and the external world. Examples of such devices which have become available from several manufacturers in recent years are cathode ray tube (CRT) controllers, floppy disk controllers, serial communication controllers, and GPIB controllers[116][118][119]. These "intelligent" peripheral controllers have the common characteristic that data read from the device by the CPU depends not only on the current, instantaneous input value, but also on the past sequence of values over some period of time. Similarly, output lines may adopt an extended sequence of values in response to a single write operation by the CPU. Furthermore, the logical relationship between data written and read by the

CPU, and data appearing at output and input pins can be simple (as in the case of a serial line controller) or complex (as in the case of a floppy disk controller).

Consequently, for many I/O devices it would not be a simple matter to simulate realistic input data in real time with programmed writes by the CPU to a parallel output port. Neither would it be simple to monitor device outputs through an input port and check them under software control. Apart from the fact that the input and output data sequences may be at a high frequency, the CPU must simulate the encoding and decoding behaviour of the device under test, to provide it with realistic inputs and to check its outputs. This means that in the case of devices like a floppy disk controller, for which input and output data sequences are time critical, the CPU could not simulate inputs and monitor outputs in real time so it is simply not possible to test the device by this technique. An alternative to the self-test (loop-back) method used in the SDK-85 must be found.

One possible solution to this problem is the use of an external I/O emulator which would provide input stimulus data and check output data for a given device. Instruments which perform this function for serial I/O controllers have been available for some time[122]. Standard test sequences for each device could be defined so that the emulator would apply a standard input sequence to the device, and the CPU would then check its response. Similarly, the CPU would program the device to perform a standard sequence of operations and the emulator would check the data sequence produced at the device's output pins. The principal disadvantage of this approach is its likely cost. One emulator would be required for each different type of I/O device to be tested in this way, and in many cases the emulator itself would

be a very complex instrument. While there is no doubt that the technology is available to achieve the required complexity and speed, and it would be possible to justify the cost of such an instrument for high volume testing applications (some ATE provide such facilities), it is a much less practical proposition for low volume, field service testing.

Another possible approach is to test I/O devices in what has been called their "natural environment"[12] - that is, with the normal external I/O hardware connected. Thus a floppy disk controller could be tested by writing data to and reading data from a floppy disk, so there would be no need to provide "artificial" input/output. The difficulty with this approach is that unless the external hardware (the interface electronics, the floppy disk drive and the disk) is known to be fault-free, failure of the test would be inconclusive because the fault may lie in any of this hardware, rather than the device under test. While other I/O controllers have less complex I/O environments, the principle still applies that unless the external hardware is known to be completely fault free, it only serves to comlicate the diagnosis of the fault if an error occurs.

A variation of "natural environment testing" may provide a practical solution to this problem. If a "known good" device of the same type as the device under test is connected in parallel with it (so that the two devices receive identical inputs, but only the D.U.T. outputs are connected back into the system) a comparator may be used to detect any differences in their behaviour. This arrangement is illustrated in Figure 4.2. With suitable allowances for varying propagation delays and critical timing events, if the comparator detects any differences between the responses of the two devices to

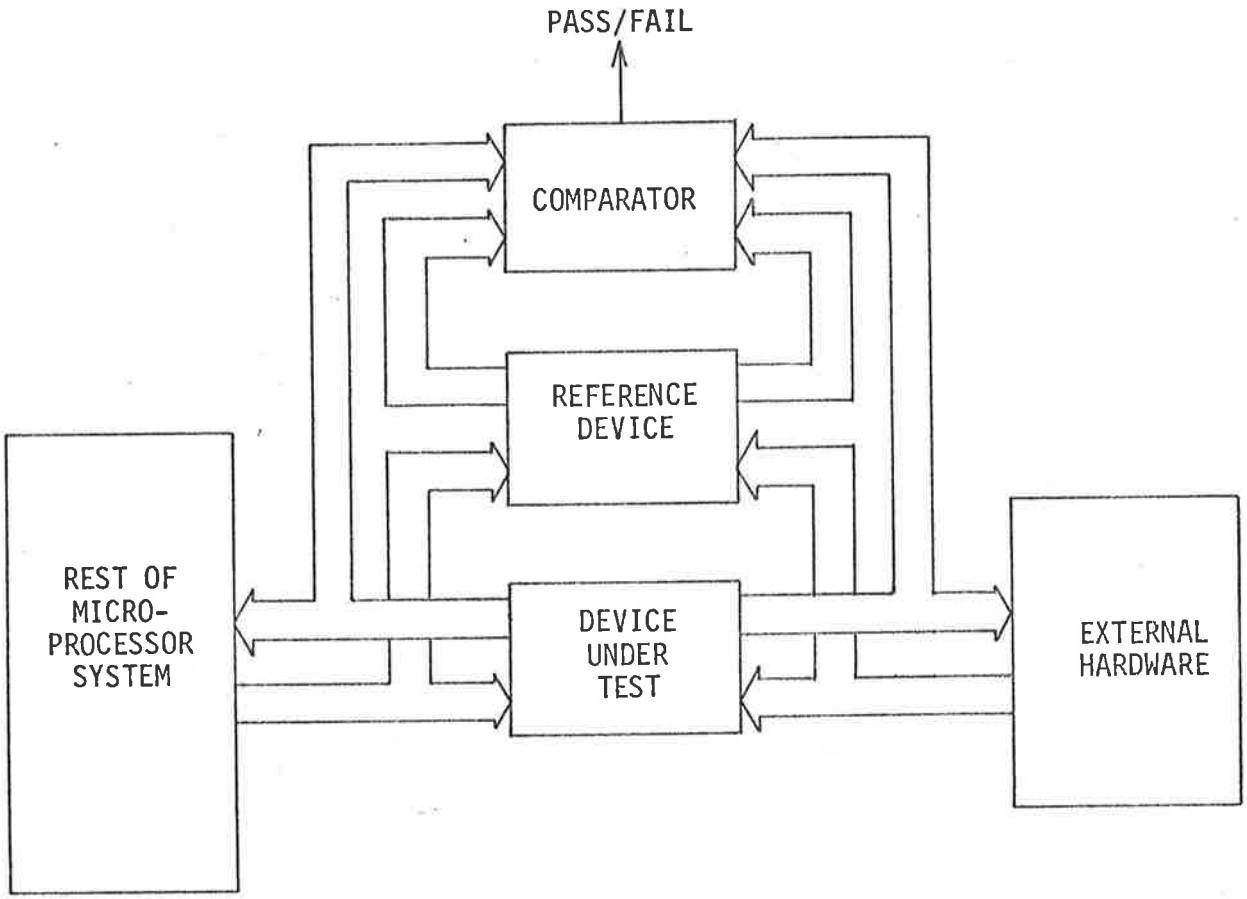


Figure 4.2. Natural Environment Comparison Testing of I/O Devices.

external input and CPU writes then the D.U.T. is assumed to be faulty.

This method is similar to that of comparison testing discussed in Chapters I and II, the difference being that the stimulus for the known good device and the D.U.T. is provided by the natural environment of the D.U.T. instead of a pseudo-random vector generator. Bisset[65] describes the use of this technique in commercial ATE, while in [12] Bluestone describes the similar use of "Conditioned Natural Environment" testing to perform parametric tests on LSI devices. It is interesting to observe that chips in the recently announced Intel iAPX 432 microprocessor chip set[123] have a "checker mode" which can be used in a very similar form of comparison testing.

The method suffers the disadvantage that it does require device-specific hardware. For each different type of device to be tested by this method a known-good device, the means to connect it in parallel with the D.U.T. and the comparator are required. Furthermore the design of the comparator may need to be quite complex to allow for variations in propagation delays and the response to critical timing events. There is no doubt that this hardware would be less expensive than the I/O emulator discussed above, but it may still prove to be an economically less attractive method for fault isolation than simple chip substitution.

Whichever method is used to provide the I/O stimulus to test an I/O device - and it is clear that simulation by the CPU is not always a practical approach - the problems, discussed in the preceding section, of developing an effective test for the device remain. The only way in which I/O devices differ from other LSI peripherals in this respect is that it is necessary to determine a suitable external stimulus

to test the device, as well as the stimulus which the CPU should apply. However, the test procedure which is developed for an I/O device must, to some extent, be determined by the means which are available of applying external stimulus to the device. The means of applying external stimulus will therefore have an effect upon the effectiveness with which the device is tested.

4.3.4 Failure of the SA procedure in the presence of certain faults

In the case of two faults described in Chapter III - the 8085 TRAP fault and the bridging fault on the printed circuit board - the SA Procedure, failed completely, identifying the wrong component as being faulty, with no means of recovery. As discussed in Section 4.2.3 total failure of the procedure in the event of an incorrect identification could be avoided in many cases if an alternative device to be replaced were specified in the documentation, and this practice is recommended for future SA implementations.

The failure of the method to isolate these two faults is due to the fact that each fault violated an assumption made in early in the SA procedure. The TRAP fault violated the assumption that the CPU is fault-free after the free-run test, and the bridging fault violated the implicit assumption that no feedback from the output ports exists which could affect the behaviour of the CPU.

We have already seen that no reasonable provision (other than the one noted above) could have been made in the design of the SA procedure which would have allowed either of these faults to be correctly isolated. For the TRAP fault to be isolated correctly it would be necessary to test the CPU during reset, for which SA is not particularly

suitable, and for the bridging fault it would be necessary to perform the Stage I tests with the system in each of its possible operating configurations, which is a practical impossibility.

It is clear, then, that in any SA procedure there will exist the possibility that a fault will occur which violates some assumption made on the basis of an incomplete test, and cause the procedure to fail. This must apply to any "automatic" test method which requires little interpretation of test results by the person applying the method. All such methods must rely on the execution of individual tests in a logical sequence, with later tests in the procedure being based on assumptions made about the system after the execution of earlier tests. If any assumption made on the basis of one of these tests is wrong the procedure must be expected to fail at a later stage. This is the reason that it is so important to conduct thorough, effective and conclusive individual tests during the SA procedure. It must be recognized, however, that simply because of device complexity it will never be possible to completely test every component in a system, so the possiblity of failure of the procedure will always exist.

The fact that these two particular faults caused the failure of the SA procedure is a consequence of the properties of the SA method. That there will always be faults which could cause any automatic test method to fail is an inevitable consequence of device and system complexity. In implementing SA or any other test method in a system, the designer can only try to minimise the likelihood that a common fault will invalidate the procedure.

4.4 Conclusions

It is apparent from the discussions in this chapter that, as far as can be assessed on the basis of the SDK-85 implementation, SA does offer many advantages as a method of field service for microprocessor systems, most of which are highlighted in the SA literature. The price which must be paid to obtain these advantages could not be accurately assessed from the SDK-85 implementation, for reasons discussed earlier in this chapter. It is worth noting, however, that the attempt to achieve fault resolution to a single component with SA involved a greater design effort and more attention to detail than indicated by the SA literature.

A number of deficiencies of SA were also observed. These are characteristics of the technique, or its recommended method of implementation, which limit its effectiveness, preventing it from being the (fictitious) ideal field service method. These deficiencies can be classified into three groups:

(i) Those which are inherent to the method. Problems in this category include failure of the method in the presence of certain faults and the inability to isolate bus faults. When these problems occur they can generally be overcome by resorting to traditional methods and they occur sufficiently infrequently that they do not detract greatly from the attractiveness of SA.

(ii) Those which can be avoided by sufficiently careful design of the system. This category includes the problems of inability to observe signatures at some nodes, and minimization of the number of different control line setups for the signature analyser.

(iii) Others - problems which don't fit into either of the above categories and which, therefore, represent areas in which further development of the SA method is desirable. These include the development of effective tests for LSI devices and the provision of I/O stimuli for testing complex I/O devices.

The two problems cited as examples in this last category constitute a serious deficiency of SA as discussed earlier in the chapter. As the number of LSI devices used in microprocessor systems increases and as integration levels increase, SA will only be effective if individual devices (including I/O devices) can be thoroughly and effectively tested. Bennetts[1] similarly concludes that the successful testing of systems depends on the ability to effectively test LSI devices, but for somewhat different reasons.

Since the problem of testing LSI devices is neither inherent to SA nor easily overcome by measures adopted during design of a system, it is a problem which could, but need not, limit the effectiveness of SA in the future. If it is not to do so, some development of techniques for testing LSI devices during SA is necessary.

The issue of testing I/O devices encompasses that of developing thorough tests for individual LSI devices, as noted earlier. In fact, the particular problems which I/O devices present only become significant after systematic test strategies for the devices have been developed. For this reason it is considered that the problem of developing systematic, thorough device tests is of more immediate concern than that of developing methods of providing real time I/O for testing I/O devices. Indeed, it is considered that this is the greatest single problem limiting the effectiveness of SA.

It was decided, therefore, to further investigate the problem of testing LSI devices in the context of SA. The aim of the investigation was to determine whether a general systematic approach is feasible and, if so, whether suitable methods exist and could be adapted for use in the context of SA. If a systematic approach were found to be infeasible, it was hoped to identify the particular characteristics of devices or systems which make it so. The results of this investigation are presented in Chapters V and VI.

CHAPTER V.

TESTS FOR COMPLEX LSI DEVICES

5.1 Method of Implementation

The ability to effectively test individual LSI devices, such as microprocessors and their peripherals, has been seen in earlier chapters to be crucial to the successful testing of LSI based digital systems. This is true for both high volume automatic testing and low volume field testing, but in Chapter IV it was seen to be so for signature analysis in particular. It is the problem of effectively testing LSI devices during the SA procedure which will be considered in this Chapter.

No one single technique has been applied universally to test LSI devices. The pseudo-random and comparison testing methods discussed in Chapter II, combined with in-circuit testing where necessary, are used with some success by ATE, but it is clear from the literature discussed in Chapter II that they are not complete solutions. ATE methods for testing LSI devices tend (naturally) to take advantage of the speed and computing power of the test equipment, which is becoming larger, faster and more powerful as LSI devices become more complex. These methods are therefore not directly applicable to the testing of LSI devices in a field service environment. Ideally, a method of testing LSI devices during field service would require little external test equipment and could be applied by an unskilled serviceman in very little time. It should, in other words, share the characteristics of signature analysis. Therefore, in attempting to find a more effective method of testing LSI devices during the SA procedure, it is important to not depart significantly from the gen-

eral SA approach as it now stands.

Because the device tests must fit within the context of SA it is clear that each device in the system must be tested in the circuit, preferably without the use of any device or system specific test equipment. The logical approach, then, is to stimulate each device from within the system - that is, implement the device tests as a self-test routine executed by the CPU on itself and its peripherals. The alternative would be to stimulate the devices with an external tester of some sort, which would be expensive and would be unlikely to offer any more power and flexibility than test routines executing within the system itself. Several authors discuss the advantages of self-testing[31][88][93] which takes advantage of the flexibility of the microprocessor and the modularity of microprocessor systems to individually stimulate and test each major device in the system.

In a self-test the set of test vectors applied to the device under test is the program stored in ROM (in the case of the CPU) or the data written to the device by the CPU as it executes the self-test routine (in the case of other devices). The response of each device to this stimulus can be observed in one of two ways. Either the test program can include routines to monitor the response of the device and give a pass/fail indication at the end of the test or the outputs of the device can be observed directly (in which case the signature analyser would be the logical instrument to use). Subject to the ability to observe signatures at all device outputs of interest, the latter method allows more direct verification of device behaviour and may therefore be preferred.

It should be noted that self tests of the type described above,

with both forms of observation of device response, are performed extensively in Stages II and III of the SDK-85 SA procedure. However, all of these tests were developed on an ad hoc basis and the aim now is to find a more systematic approach to the development of such tests.

While device testing through the execution of a self-test program does minimise external hardware requirements, which is important in the field service environment, it also imposes several restrictions. The most obvious of these is that because the self-test program is stored in ROM, and there are practical limitations on the amount of ROM storage which can be provided in a system, the size of the self-test program is restricted. Srini[91] describes an extensive self-test scheme in which large diagnostic routines are held in on-line secondary storage and are loaded into RAM and executed as required. However, in general, microprocessor systems will not be equipped with secondary storage facilities, so test routines must be resident and therefore cannot be made arbitrarily large. As the cost of primary storage for microprocessors decreases the size of resident test routines will become less significant although in some applications it will continue to be important to minimise the amount of ROM storage required.

The second restriction is that the stimulus applied to devices can only be in the form of valid bus operations. The only way in which a CPU can be stimulated is by reading data from ROM through the data bus, while stimulus for a peripheral device is restricted to CPU "write" operations (together with any external input which may be applied to an I/O device). For example, it is not possible in a self-test routine to stimulate a peripheral device by setting its RD/ and WR/ inputs low, as this is not a valid bus operation. Thus the set of input vectors

which may be applied to a device is restricted. However, as these devices will almost invariably have been designed to operate on a bus and only respond sensibly to valid bus operations, stimuli which are not valid bus operations would be of questionable value. Therefore this restriction is not considered to be a serious one.

The final restriction, as noted in Chapter II, is that before a self-test routine can be executed, the system kernel must be tested and verified to be working - at least to some extent. A self-test routine for the CPU can be written in such a way that only the ROM, address decoding and busses are required to be fault-free. Peripheral tests, however, will in general use some RAM and therefore require that both the CPU and the system RAM be fault-free as well. In the context of the SA procedure these requirements are satisfied if the CPU test is conducted after Stage I, in which the system kernel is tested, and the peripheral tests are conducted after Stage II, in which the RAM is tested.

Finally, it may be noted that although the emphasis of this investigation is on developing device tests in the context of SA, because they are to be implemented as self-tests, any results obtained will have wider application to all forms of self-testing in microprocessor systems.

5.2 The Generation of Practical Device Tests

The discussion so far in this chapter has been concerned with the most suitable means of implementing LSI device tests (that is, of applying the test stimulus and observing device response) during sig-

nature analysis of a microprocessor system. The process of generating the tests (that is, of devising the most suitable test stimulus for each device) will now be considered.

Ideally a device would be tested by the application of all possible input vectors in all possible sequences. However, as has been noted many times already, this approach is simply not practical and some form of compromise is necessary, in which a restricted set of test vectors is applied to each device. The process of generating a device test is one of finding the most suitable and effective compromise for a given test environment (the field service/signature analysis environment in this case). The ad hoc device tests developed as part of the SDK-85 SA procedure are attempts at such compromises, but are clearly not the best ones possible. Those tests placed more emphasis on exercising the devices in some convenient manner than on testing the devices thoroughly, and in consequence were not as effective as desirable. A more systematic approach to the development of thorough, but reasonably sized, tests is required.

5.2.1 Functional tests

Up to the present time the approach to the development of systematic tests for LSI devices (particularly microprocessors) which has been adopted almost universally has been to develop the test based on the device's internal architecture[8]. The principle underlying this approach is similar to that involved in the difference between functional and in-circuit testing of systems at the board level.

To briefly repeat the discussion of these two methods presented in Chapter I, functional testing involves the exercising and observation of a board from its external inputs and outputs, with the aim of

detecting, and possibly isolating, faults in the board. Thus tests are developed for the board as a whole, and can be very complex for boards containing many devices. In-circuit testing, on the other hand, tests devices on the board individually, on the assumption that, because the devices are independent, if each device is tested and found to be fault-free then the board as a whole will function correctly. Because devices are tested individually, in-circuit tests consist of a sequence of several relatively simple device tests, rather than a single complex test for all logic on the board. In-circuit tests are therefore easier to develop than functional board tests.

At the level of LSI device tests the development of a test for the device as a whole is analogous to the development of a functional board test. The length and complexity of the test can be reduced if the LSI device is considered as a set of independent modules (analogous to individual devices on a board) which are tested individually. Then the device test will consist of a sequence of relatively simple tests for each of its modules, on the assumption that if each module is tested and found to be fault-free then the device as a whole will be fault-free. This approach takes advantage of the fact that most LSI devices are modular in design and are physically composed of independant modules or "functional units" (FU's) which can often be identified in a photomicrograph of the device[42]. Individual modules perform well defined and often simple functions independently of other modules. Tests may therefore be constructed to exercise the function of one module without greatly affecting others.

The fact that tests based on this approach seek to verify that each module, or FU, functions correctly has led to the unfortunate use of the term "functional tests" to describe the approach, even though

the principle followed is more in line with in-circuit board testing than functional board testing. The term "functional testing" will henceforth be used only to refer to the modular tests performed on LSI devices.

Although functional testing shares a common principle with in-circuit board testing, there are two important differences between the techniques. The first, and most obvious is that whereas individual devices on a board can be accessed directly in in-circuit testing, with a bed-of-nails or I.C. clip, this is not possible for the functional units within LSI devices. The internal FU's must be exercised by the application of test vectors to the external device pins. This means that it is not possible to exercise one FU in isolation. Any operation performed by the device in response to an input stimulus will, in general, involve several FU's.

The second difference is that the aim in performing LSI device tests is simply to detect faults of any kind in the device. The identity of any FU which contains a fault is of no concern because if a fault is detected the whole device is replaced. This is to be contrasted with in-circuit board testing in which the device tested is replaced if found to be faulty, rather than the entire board. The diagnostic requirements in functional device tests are therefore less demanding, which is some compensation for the fact that FU's cannot be accessed directly.

5.2.2 Literature on functional testing

Several authors have proposed methods for generating tests for LSI devices based on consideration of device architecture. Up to the present time these have all been developed specifically for micropro-

cessors rather than LSI peripheral devices.

Chiang and McCaskill[42] describe a method for generating tests for microprocessors which they call "module sensorialization". To apply this method the microprocessor is conceptually divided up into modules, including the arithmetic logic limit (ALU), register array, program counter, instruction decoder and timing logic. A series of short instruction sequences is applied to the device, each designed to exercise and verify the correct operation of one of these modules. For example, the first stage of the test is to continually force a "no-operation" instruction onto the data bus of the microprocessor, (as in the free-run stage of signature analysis), to test the program counter as it is incremented through its full range of possible values. The test for each module is designed to be a "worst case" test in some sense. The register array, for example, is tested with a galloping ones and zeros pattern, while the instruction decoder is tested by the execution of all instructions in the device's instruction set.

The authors propose the technique of "algorithmic pattern generation" (APG) to apply the test vectors (that is, the instruction sequence) to the device under test. A microprogrammed control unit generates the required instruction sequence to be applied to the DUT, using a small control store, thereby avoiding the need for a large on-line store to hold long test instruction sequences. It is important to notice that the test instructions generated by APG are applied to the microprocessor, rather than fetched by the microprocessor in the normal sequence of execution. Thus there are no limitations imposed by the need to fetch instructions from contiguous memory locations, and tests which manipulate the program counter can be performed freely.

Crichton[64] also discusses module sensorialization, noting that because of device complexity and the unavailability of gate-level logic diagrams for devices, it is not practical to test devices as a whole using conventional methods. He gives a little more detail on the derivation of test instruction sequences, considering the "data logic" and "control logic" of the microprocessor separately, and states that each module should be tested "with as much data as can be practically tolerated". It is noted that although the control and data logic sections of the microprocessor are conceptually tested separately, it is not possible to exercise one without involving the other, and that it is therefore impossible to test modules in isolation. However, if the device is fault-free all tests will be passed irrespective of whether modules are exercised in isolation.

The APG implementation of module sensorialization is criticised by Smith[124] on the grounds that, in testing modules within the microprocessor, it does not test for interaction between the modules - between registers in the register array, for example. He proposes that the existence and uniqueness of each module within the microprocessor should be verified. An example instruction sequence which would verify the existence and uniqueness of the registers in an Intel 8080 microprocessor is given. Significantly, this test sequence, being more thorough than those given by Crichton[64] and Chiang and McCaskill[42], is also longer.

Although module sensorialization and APG were originally proposed for the ATE environment, module sensorialization could be applied in the form of a self-test routine, with the test instruction sequences stored in ROM. However, this would place some restrictions on the tests which could be performed because the instructions must all be fetched

from the ROM. Tests which modify the program counter, for example, would be restricted. The response of the microprocessor to the test sequences could be readily observed with a signature analyser.

None of the papers discussed above gives any specific directions on how test instruction sequences for modules in the DUT should be developed, or defines which faults are detected by the method. That is, the method of module sensorialization is not based on a specific fault model. Chiang and McCaskill[42], Crichton[64] and Barraclough et al.[47] all state in general terms that the aim of the procedure is to verify the function of each module in the DUT with as much data as possible, or under worst case conditions, but give no guidelines for the generation of instruction sequences beyond that. Individual test sequences must therefore be developed on an ad hoc basis.

Robach and Gobbi[125] consider the testing of microprocessor systems in several environments, including that of the device manufacturer, who can use knowledge of the mask layout of the device to generate tests for it. For the system manufacturer, an external tester is recommended, to exercise subsystems within the microprocessor system, performing simple RAM, ROM, CPU and peripheral tests of the type performed in Stages II and III of the SDK-85 SA procedure. A method of testing a microprocessor system in the field through the execution of its "application program" is also presented. It is based on a method of testing computer control units presented in an earlier paper by Robach and Saucier[126]. The entire microprocessor system is considered as a set of functional units, comprising the "operative part", while the application program is the "control part". To test the system, the set of inputs applied to the control part (that is, the input data, or parameters applied to the system) are manipulated so that, during

execution of the application program, all functional units in the operative part are exercised and subsequently observed. Algorithms are presented by which a suitable sequence of execution within the application program may be determined, such that all functional units are eventually exersised and all possible execution paths within the application program are tested.

Although this method could, in principle, be applied to LSI device testing within a microprocessor system, for several reasons it is not particularly suitable for application in the context considered in this chapter. The extent of the tests performed on FU's within the system appears to be somewhat arbitrary. There is the implication in the approach that a FU is fault-free if it is involved in some operation which yields a correct result. Certainly the approach would not produce the thorough tests for LSI devices which are the subject of this chapter. The means by which the results of operations should be observed are not made clear. Thatte and Abraham[36] state that difficulties may be experienced in applying the method to testing microprocessors because of the limited accessibility of CPU registers and the consequent impossibility of observing the results of operations directly.

Clearly, the method can only test those FU's used by the application program. This may be satisfactory in a dedicated system, but is not so for a general purpose system such as the SDK-85. The application program of the SDK-85 (the monitor) does not use some of the facilities in the system, so these facilities could not be tested by execution of the application program; a specific test routine is required. Finally, the method does not appear to be particularly suitable for isolated device tests because it requires that the entire

system be divided up into (presumably small) FU's. The procedure for applying the method to a system which contains several complex LSI devices and a large application program would therefore be very complex and time consuming.

The testing of microprocessors is considered by Ballard[93] in the context of fault detection in microprocessor systems. He discusses, among other methods, the execution of a self-test routine by the microprocessor to detect any faults within the device. In particular, he considers thd problem of "fault masking" - the mechanism whereby a fault in the device is not detected by the test procedure because of the existence of a second fault which "masks" the first. With a view to reducing the likelihood of this occurring in a microprocessor self-test he proposes a method for determining the most reliable, or simplest, instruction and the most reliable functional unit of a microprocessor. Srini[91] presents a similar method. The simplicity of an instruction is assessed by counting the number of attributes or FU's involved in the execution of the instruction, while the reliability of a FU is determined by the number of gate levels, feedback paths, instructions and clocks used by the FU. The test routine is constructed so that it first exercises the most reliable FU using the simplest instructions (thereby minimising the probability that fault masking will occur). This FU is then used to test the second most reliable FU and instructions and so on, until all FU's are tested. Bilton[35] describes a similar bootstrapping technique for testing microprocessors.

The purpose of performing this bootstrapping procedure is not explained well by Ballard and needs some clarification. The ultimate aim in performing a self-test on a microprocessor (or any other device)

is to determine whether the device functions correctly - that is, whether it contains any faults. In the field service environment the nature of any fault detected is of no concern because a faulty device will simply be replaced and discarded. Functional tests of the type described in this chapter aim to detect faults in the DUT by individually exercising the various independent FU's within the device as thoroughly as possible, on the assumption that if all FU's are found to function correctly then the device as a whole will function correctly.

The only means of exercising a FU within a microprocessor is to force the microprocessor to execute an instruction sequence devised for that purpose. However, this must also involve several other (untested) FU's within the CPU, such as the program counter, data input buffers and instruction decoder. As noted earlier, it is therefore impossible to test any FU in isolation; the test for any FU must be supported by other FU's. Thus, a fault in the accumulator, for example, may go undetected because the data input buffers exhibit a complementary fault which gives rise to the correct results from a test devised for the accumulator. The likelihood of this type of fault masking occurring between FU's is reduced by minimising the involvement of extra FU's in the test for a given FU (unless the extra FU's have already been tested). For this reason the test for a FU should be constructed from the simplest possible instructions - those which use fewest other FU's.

In practice it is generally not possible to construct a test for a FU with only the simplest instructions of the CPU. As noted by Bilton[35], instructions in test sequences are often chosen for reasons of convenience rather than simplicity alone, although the

simplest instructions should always be used when there is any choice. After a FU has been tested with simple instructions, it may be used to support tests for other FU's, which employ less simple instructions. That is, instructions which use the tested FU may be used in tests for other FU's. It should be noted, however, that while tests should be conceptually bootstrapped in this manner, the actual order of execution is irrelevant. It doesn't matter which of the FU tests is executed first if the results of all FU tests are observed, and the DUT is only passed if they are all correct.

It is apparent, then, that Ballard's proposition that "using the most reliable instruction and functional unit, the CPU may be exercised from the most to the least reliable of its elements" need not be followed literally. To determine which FU is the most reliable based on the criteria proposed by Ballard would require a gate level logic diagram of the CPU, which is generally not available. Furthermore, there appears to be no reason that the simplest or most reliable FU should be tested before any other. The FU to be tested "first" should be the one which can be tested by the simplest instructions - those which use fewest other FU's. The logical complexity of a FU is therefore a less important consideration than the way in which it fits into the device's architecture, and the degree to which it can be tested in isolation. The complexity of a FU will be reflected in the number and type of instructions which are required to exercise it, but its complexity will not determine its position in the test order so much as the availability of simple instructions to exercise it.

The final architecture based method to be considered is that developed by Thatte and Abraham[36]. Information which can be readily obtained from the instruction set description of the microprocessor is

used to construct a register-transfer level graph-theoretic model of the device, in which nodes represent CPU registers and data flow between the registers is represented by links between the nodes. Simple fault models are proposed in terms of the register decoding, instruction decoding and data transfer functions within the CPU. The device is tested by executing instruction sequences devised to move data between registers along the various data paths and out to the external world for observation. Several algorithms are presented by which instruction sequences can be devised to detect all faults in the fault model. However, the algorithms require ad hoc decisions about which data and instructions should be used at various stages.

It is clear that the method, as presented, is specific to microprocessors, with their data path oriented architectures and would not be readily applicable to general LSI devices. The fault model is also very specific to microprocessors and would be irrelevant for many LSI peripherals, for which it would be necessary to propose and justify new fault models.

5.2.3 Discussion

The methods for developing microprocessor device tests discussed above all rely on consideration of the architecture of the device and the division of the device into independent functional units as a means of devising a test program of reasonable length. The differences between the techniques arise from the different methods of dividing the device into FU's and the different intended applications of the test routine. They are all (with the exception of the method of Robert and Gobbi, which is applied at the system level) specifically concerned with testing microprocessors.

None of the methods discussed defines an algorithm in sufficient detail to allow a test sequence to be derived without some judgement or inventiveness being exercised by the author of the test program. This fact is reflected by Bilton's statement[35] that test engineering is still "something of an art". None of the methods, with the exception of Ballard, is particularly well suited to the signature analysis self-test environment, for the reasons discussed in the preceding section. However, all of the methods do have their merits and there is scope for variations upon the methods (themselves variations on the modular or functional testing principle) to suit the particular test application and environment. Certainly the functional testing approach seems to be a suitable, practical approach to the generation of thorough LSI device tests. Indeed, in the absence of any documented alternative approach or any serious disadvantages apparent from the foregoing discussion, it is the most suitable approach.

As discussed at the beginning of this chapter, the requirement in the signature analysis context is for a test method for general LSI devices which is systematic and thorough, but also practical. The desirability of the self-test implementation has already been established and it is now apparent that functional testing is the most suitable approach to development of the tests. It was decided, therefore, to attempt to extend the SDK-85 SA Procedure by the inclusion of tests, developed by this approach, for the two most complex devices in the system (the 8085 and the 8279). It was anticipated that in so doing any limitations of functional testing in this environment would become apparent. In particular, any problems peculiar to peripheral devices as opposed to microprocessors would be identified. A description of the development and evaluation of a functional test for the 8085 follows, while the 8279 test and overall conclusions about functional testing will

be presented in Chapter VI.

5.3 8085 Functional Test

5.3.1 The approach to development of the test

In developing a functional test for the 8085 the intention was to form the basis for a systematic test approach for complex LSI devices in general. It was therefore important that the approach adopted not be too specific to microprocessors. It was also clear from the outset that the extent to which a systematic procedure can be applied will be determined by the architecture of the device being considered. Therefore the test was not developed to detect a pre-defined set of faults. Rather, the goal adopted was to develop a test which would detect the most general set of faults possible, given the limitations imposed by the device architecture. Only thus would the true limitations of functional testing in this application become apparent. It was clear, however, that it would be necessary to assume that any faults existing in the DUT were "hard" faults - that is, neither intermittent nor pattern sensitive. The initial fault model adopted for the 8085 (and subsequently the 8279) was simply the set of all hard faults.

Given this very general fault model, and the need to adopt a generally applicable approach, the functional test was developed broadly along the lines described by Ballard[93] and Srini[91]. A functional model of the CPU was adopted and its functional units were identified. A test sequence was developed using the simplest possible instructions to test each FU as thoroughly as possible without creating

an unduly long test. That is, the function of each FU was tested with as much data as practically possible. It was then intended to adopt an identical approach to the development of the 8279 functional test.

It was decided that the signature analyser would be used to observe the data appearing on the CPU data and address busses during the test, thereby monitoring its response to the test. This method of observation allows the behaviour of the CPU to be monitored more directly, as discussed earlier in this chapter.

5.3.2 The 8085 functional model

The first step in the development of the 8085 functional test was to adopt a functional model of the device, in effect breaking it up into FU's. Clearly, the more detailed the model is - that is, the smaller and simpler the FU's are, subject to the condition that all FU's should function independently - the more effective the functional test procedure can be. The most detailed information about the 8085 which is readily available is contained in the MCS-85 User's Manual[120] which defines its architecture and interfacing characteristics. In a manner which is typical for microprocessors of its class, the operation of the 8085 is described in terms of the effects which its various instructions have on its attributes. Since the device can only be tested by executing valid instructions, it is clear that the functional units should correspond to the entities manipulated by the instruction set - the registers, accumulator, ALU, program counter etc. These functional units are shown in the "functional block diagram" of the 8085 given in its data sheet[116][120].

The operation of the Intel 8080A microprocessor, to which the 8085

is architecturally very similar[22], is described in more detail in the Intel 8080 Microcomputer Systems User's Manual[127] than the operation of the 8085 is in the MCS-85 User's Manual. The functional block diagram of the 8080A which it contains includes more detail than that of the 8085, and a state transition table is provided (on pp 2.16 - 2.20) which gives details of the clock cycle by clock cycle activity of the CPU in terms of the entities shown in its functional block diagram. Because a state transition table, or information equivalent to it, is required to develop a functional test at the level of the FU's shown in the functional block diagram, it was decided to derive the functional model of the 8085 from that of the 8080A. Assuming that activity within the 8085 is similar to that within the 8080A during instruction execution, the derived state transition table would then represent what physically occurs within the 8085. Thus the 8085 functional model would be physically meaningful. Without the state transition table only a logical model of activity within the 8085, which may or may not be physically accurate, could be developed.

The functional block diagram used to develop the 8085 functional test is shown in Figure 5.1. It was adapted from the 8080A functional block diagram to allow for obvious architectural and implementational differences between the processors as follows:

(i) The 8080A Data Bus Buffer/Latch was modified to include inputs from the eight least significant bits of the address latch, providing for the multiplexed data/address bus of the 8085. Accordingly, the Address Buffer is now only eight bits wide.

(ii) A serial I/O module was included to allow for the serial input/output facilities of the 8085. A bidirectional connection to the internal data bus was provided to allow data

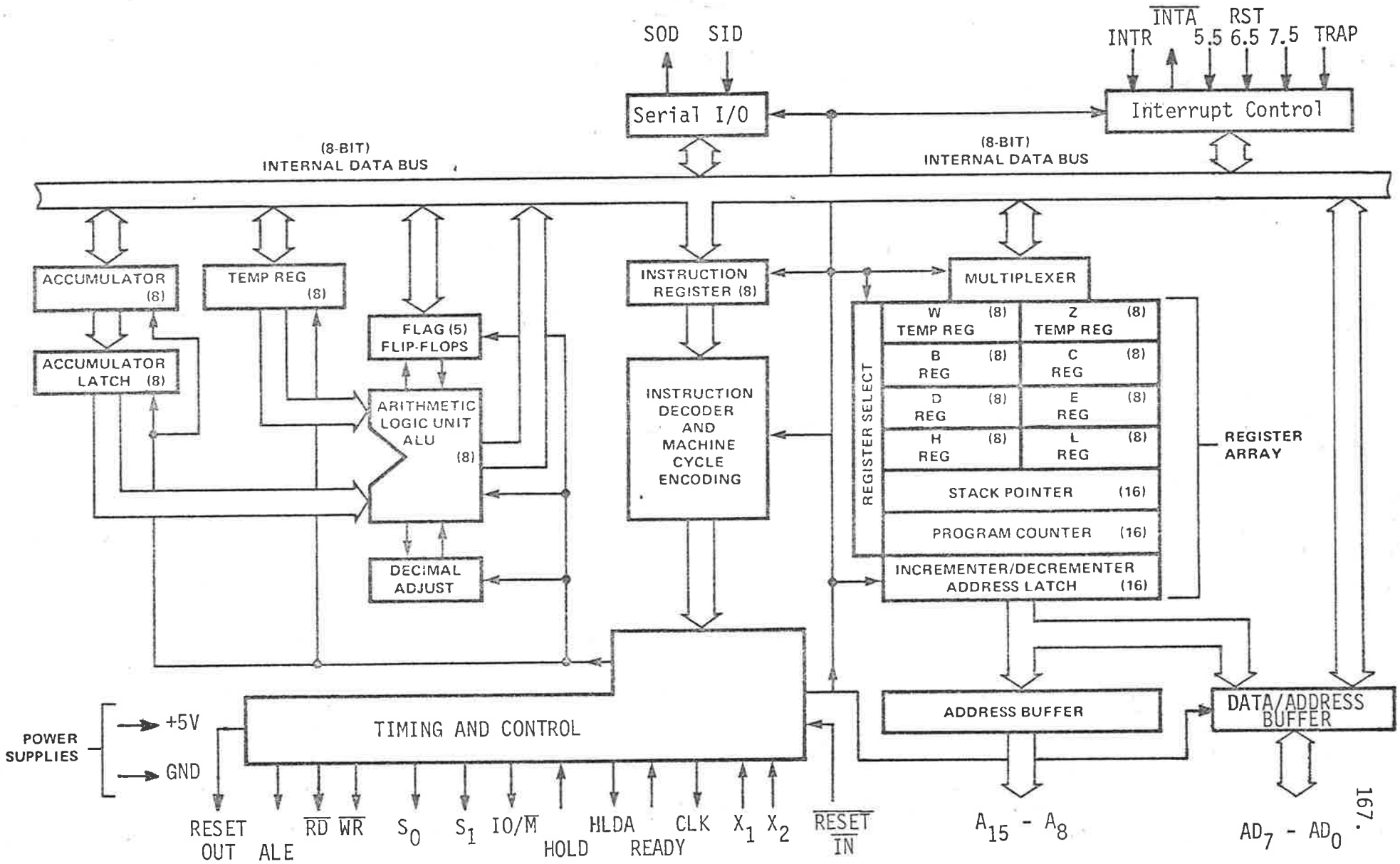


Figure 5.1. 8085 Functional Block Diagram.

(Reprinted by permission of Intel Corporation, Copyright 1981.)

transfer between the accumulator and the serial I/O pins during execution of the 8085's RIM and SIM instructions.

(iii) The interrupt logic and control lines were separated from the Timing and Control module. A bidirectional connection to the internal data bus was provided to allow interrupt status to be read and the interrupt masks to be set by the RIM and SIM instructions.

(iv) The external control and status lines emanating from the Timing and Control module were changed to those of the 8085. The internal crystal oscillator was included in the Timing and Control module.

(v) The -12 volt and -5 volt power supply inputs were eliminated.

Some of these revisions were influenced by the functional block diagram given in the 8085 data sheet and the modified functional block diagram (Figure 5.1) is consequently very similar to it. However, Figure 5.1 does show a little more detail, including the accumulator latch, register multiplexer, register select module and the decimal adjust module. Although some of these modules may not exist as such in the 8085 it was necessary to include them from the 8080A functional model because the 8080A state transition table, which was to be adapted for the 8085, contains references to them. If these modules had been omitted (as in the block diagram in the 8085 data sheet) significant changes to the state transition table would have been necessary, largely defeating the purpose of using it. The changes which were made to the 8080A functional block diagram do not conflict with any activity defined in the state transition table.

The state transition table for the 8080A[127] was modified for the 8085 to accommodate the known differences between the two devices

which could determined from their respective User's Manuals[120][127]. In so doing it was assumed that, because of the architectural similarity between the devices, only minimal differences exist in their state-by-state activity. The following changes were made to the state transition table:

(i) RIM and SIM instructions were included for the 8085. The assumed state-by-state activity during execution of these two instructions, which is based on the M1 cycle activity for other instructions, is shown in Table 5.1.

(ii) The numbers of clock cycles (states) occupied by some instructions were changed. The number of machine cycles used by the 8085 for each instruction is the same as for the 8080A but the number of states occupied is different in several instances. The MCS-85 User's Manual[120] states that, with the exception of M1 (opcode fetch) machine cycles, all machine cycles occupy three states. M1 cycles occupy either four or six states. Thus to account for the documented 8085 state times, the M1 cycles of the following instructions were assumed to be one state longer than in the 8080A:

INX *rp*; DCX *rp*; SPHL; CALL *addr*; C*cond addr*;
R*cond*; RST *n*; PCHL; PUSH *rp*; PUSH PSW.

On the other hand, the following instructions were assumed to have M1 cycles which are shorter by one state:

MOV *r1*, *r2*; INR *r*; DCR *r*; J*cond addr*.

Table 5.1. State Transition Table Entries for the 8085 RIM and SIM Instructions

Mnemonic	M1			
	T1	T2	T3	T4
RIM	PC OUT STATUS	PC = PC + 1	INST → TMP/IR	(INTERRUPT, SERIAL I/O) → A.
SIM	PC OUT STATUS	PC = PC + 1	INST → TMP/IR	(A) → INTERRUPT, SERIAL I/O

(iii) During the execution of conditional jump and call instructions the 8085 does not read the second byte of the destination address if the condition specified is not satisfied. In such cases the J*cond addr* and C*cond addr* instructions are one machine cycle (three states) shorter. These instructions were assumed to be otherwise identical to the 8080A instructions, except that the program counter was assumed to be incremented twice during the second machine cycle and, as noted above, the M1 state times are different. This behaviour implies a greater complexity in the Timing and Control logic of the 8085, but is otherwise inconsequential.

(iv) The only 8080A instruction which has a machine cycle (other than an M1 cycle) of more than three states is XTHL, which has an M5 cycle of five states. The M5 cycle of the 8085 XTHL instruction must be only three states long, although it is not clear how the necessary activity during M5 can occur in only three states. This anomoly may indicate that a feature was omitted from the functional block diagram which would allow greater parallelism during this cycle. However, in the absence of further information this was assumed not to be the case and the involvement of FU's in the M5 cycle of XTHL was assumed to be as for the 8080A.

It should be noted that the 8080A state transition table appears to be in error because it does not show that the stack pointer is decremented during the M4 cycle of XTHL, as must be the case.

The assumptions made with regard to the functional model of the 8085 were based largely on the degree of architectural similarity between it and the 8080A. It was assumed that the differences between the implementations of the two devices were as few as possible, given the documented differences between the devices. The validity of these assumptions is, of course, open to question but in the absence of further information no other reasonable assumptions could be made. Any inaccuracies in the assumed functional model could be expected to give rise to a less effective functional test because the model used would then be a logical one, rather than one which represents the physical implementation of the device. Nevertheless the model serves to illustrate the process of developing a functional test for a microprocessor as well as any, and in this respect can at least be expected to be reasonably representative of the 8080A, if not the 8085.

5.3.3 Development of the 8085 test

5.3.3.1 Instruction complexity

To determine which instructions of the 8085 are the least complex and therefore are most suitable to be used in test sequences for FU's, a table similar to that suggested by Ballard[93] and Srini[91] was constructed. It was derived from information obtained directly from the state transition table and showed, for each instruction of the 8085, which functional units or attributes of the device are used during execution of the instruction, and how many times they are used. For each instruction two measures of the complexity of the instruction were calculated, and these are listed in Table 5.2. The first of these measures - denoted 'F' - is simply the number of FU's affected or used during execution of the instruction, wherein each flag flip-flop

Table 5.2. Complexity Values for 8085 Instructions

Instruction	Number of functional units, F	Complexity C
MOV *r1, r2*	10	21
MOV *r*, M	10	24
MOV M, *r*	12	26
SPHL	11	23
MVI *r, data*	10	26
MVI M, *data*	10	32
LXI *rp, data*	11	39
LDA *addr*	12	46
STA *addr*	12	46
LHLD *addr*	13	62
SHLD *addr*	13	62
LDAX *rp*	11	22
STAX *rp*	11	22
XCHG	12	28
ADD *r*	18	29
ADD M	18	33
ADI *data*	16	35
ADC *r*	18	30
ADC M	18	34
ACI *data*	16	36
SUB *r*	18	29
SUB M	18	33
SUI *data*	16	35
SBB *r*	18	30
SBB M	18	34
SBI *data*	16	36
INR *r*	15	26
INR M	15	35
DCR *r*	15	26
DCR M	15	35
INX *rp*	10	24
DCX *rp*	10	24
DAD *rp*	18	47
DAA	16	25
ANA *r*	18	29
ANA M	18	33
ANI *data*	16	35
XRA *r*	18	29
XRA M	18	33
XRI *data*	16	35
ORA *r*	18	29
ORA M	18	33
ORI *data*	16	35
CMP *r*	18	28
CMP M	18	32
CPI *data*	16	34
RLC	12	19
RRC	12	19
RAL	12	19
RAR	12	19

(Table 5.2 continued on Page 174.)

Table 5.2 (continued)

Instruction	Number of functional units, F	Complexity C
CMA	11	17
CMC	10	16
STC	10	15
JMP *addr*	11	38
J*cond addr*	12	39
CALL *addr*	12	65
C*cond addr*	13	66
RET	12	38
R*cond*	13	39
RST *n*	12	46
PCHL	10	22
PUSH *rp*	12	43
PUSH PSW	15	43
POP *rp*	12	38
POP PSW	15	38
XTHL	15	64
IN *port*	12	36
OUT *port*	12	33
EI	9	15
DI	9	15
HLT	8	20
NOP	8	14
RIM	15	21
SIM	11	17

is counted as a separate FU, as suggested by Ballard. The second measure - the complexity, or 'C' - was intended to provide a finer indication of instruction complexity. It was calculated as the sum of the number of individual operations on each FU (where, for example, each writing of the program counter to the address latch would be counted separately) and the maximum number of states occupied by a machine cycle of the instruction.

The tabulated C and F values were derived from the assumed functional model of the 8085. The method of calculating these measures of complexity was based on that proposed by Ballard[93] and Srini[91] and was considered to be a reasonable approach to the task. However, it is clear that many variations of this method of calculating instruction complexity are possible and equally reasonable. The particular values calculated are not accurate measures of any particular property of the 8085, and were not intended to be. They were intended, simply, to provide a broad indication of which instructions are least complex. Because many other equally valid calculations of instruction complexity could have been performed the details of the calculation of C and F are not presented here. It is expected that any calculations performed along the lines described above would provide an adequate indication of instruction complexity.

It may be noticed from examination of Table 5.2 that there is an obvious correlation between the sequences of F values and C values. Given that these values are not to be taken as accurate in any sense, it is apparent that either F or C could be used to compare the suitability of two instructions for use in a FU test sequence. However, C does provide a finer indication of instruction complexity and confirms, for example, the intuitive results that a DCR M instruction

(F = 15, C = 37) is more complex than a DCR *r* (F = 15, C = 26) and that a DXC *rp* (F = 10, C = 24) is more complex than a STC (F = 10, C = 15). Therefore, to choose between two alternative instruction sequences for testing a given FU, the sums of the C values for instructions in the two sequences were compared and, in general, the sequence with the lower total C was used. However, it could be argued that the more easily calculated F values for instructions could have been used without significant penalty.

Finally, it should be noted that while the C values were used to choose between FU test instruction sequences, it was often necessary to use instructions with quite high C values for reasons of convenience - as mentioned by Bilton[35]. In some cases, particularly when choosing between instructions of similar complexity, it was also found to be necessary to apply other criteria - specifically, which extra FU's are affected by an instruction.

5.3.3.2 The accumulator test

Disregarding, for the present, the free-run test performed on the 8085 during Stage I of the SA procedure, the first FU of the 8085 for which a test was generated was the accumulator (ACC). It was clear that it could be tested more easily (that is, involving fewer extra FU's) than any other FU, because of its direct connection to the internal data bus and the consequent ease with which data could be moved between it and the external data bus. ACC functions simply as a register, so given that the fault model includes only hard faults, the most general test for it as an independent FU would be one which verifies its ability to store all 256 possible data values and reproduce those values on the internal data bus. The functional block diagram (Figure 5.1) shows a data path from ACC to the accumu-

lator latch (ACT) but, for reasons which will be explained in the discussion of the ALU test, it is not necessary to test this path as part of the ACC test.

Given that the ACC test should consist of writing and reading 256 distinct bytes of data, an instruction sequence to store the data in ACC and read it out again must be chosen. The simplest such sequence (the one involving fewest extra FU's) would consist of 256 MVI A instructions to store the data and OUT instructions to read it back again. (The STA and MOV M, A instructions could also be used to place the contents of ACC onto the external data bus, but neither was seriously considered because STA has a very high C value and MOV M, A moves the data through the temporary register (TMP).) However, such a test would require four bytes of ROM storage for every byte of test data, yielding a 1K byte test sequence just for ACC. While this in itself is tolerable, tests of this length could not be tolerated for each FU in the 8085 because the amount of available ROM storage is limited in practice. A test for a sixteen bit register equivalent to that proposed for ACC would occupy 256K bytes of storage which is clearly out of the question. It is interesting to note that Chiang and McCaskill[42] report that only six micro-instructions are required to generate this 256K byte test stimulus by APG.

Because of the storage limitations imposed by the self-test implementation, it was necessary to use a method of generating the test data for ACC which is more economical than storing the test data explicitly in ROM. This meant that the test data must be generated internally to the 8085 which, in turn, meant that the ALU or the sixteen bit incrementer/decrementer must be involved in the test, and a conditional jump instruction must be used. There are many possible

instruction sequences which would generate the 256 bytes of data for the ACC test but the one chosen, for reasons of simplicity (lowest total C value) was

```
        MVI   A, 00H
LOOP:   INR   A
        OUT   00H
        JNZ   LOOP
```

where the output address 00H was chosen arbitrarily.

Although this loop constitutes the simplest test of its type, it does make use of several FU's apart from ACC, and the input buffers, instruction register and instruction decoder which must be involved in the fetching and execution of instructions. The extra FU's involved are the TMP register (which is used by INR A), the ALU (by INR A), the data output buffer (by OUT 00H), the Z flag (by INR A and JNZ), and the W and Z registers (by JNZ). In general each of these FU's could contain a fault which would mask a fault in ACC. That is, even if the data observed on the data bus during execution of the OUT instruction were correct, it could be that this resulted from two complementary faults - one in ACC and one in another FU. Indeed, it is easy to postulate possible (if physically unlikely) fault pairs in ACC and almost any one of the other FU's involved in the test which would still yield the correct output data. It is complications of this type which make the generation of complete, conclusive tests for FU's difficult, even in the case of an FU as simple as the accumulator.

If the set of allowed faults in the device is restricted (that is, if the fault model is made less general) then the ACC test given above

is complete and conclusive. If, for example, it is assumed that only one FU in the DUT can be faulty at a time - which, given the physical modularity of LSI devices noted earlier, is not an unreasonable assumption - then incorrect ACC operation could not be masked by a second fault. Correct output data would then unambiguously indicate that the accumulator is fault-free. In practice, however, it is difficult to justify the assumption that only one FU can be faulty and the probability of complementary faults in two FU's, however small, is not zero.

A second restricted fault model which may be considered is one which includes only stuck-at faults. If any bit of ACC were stuck at one or zero, ACC could only store 128 distinct bytes of data. In that case no simple stuck-at fault in any of the other FU's involved in the ACC test could result in the correct 256 byte data sequence appearing at the output buffers. Once again the ACC test given above would be complete and conclusive.

Although the test was constructed specifically for the accumulator, it may be observed that during the test 256 distinct bytes of data are passed through the data output buffers. Therefore, applying the same reasoning as for the accumulator, under either of the two restricted fault models described above, the data output buffers are fully tested. The same is also true of the TMP register which, during the test, stores and passes on to the ALU 256 distinct bytes of data. Note, however, that the data path from TMP back to the internal data bus is not tested. It should also be noted that the other FU's involved in the ACC test cannot be considered to be fully tested under either fault model because they are not fully exercised during the test. The data input buffers, for example, only pass seven distinct

bytes of data.

The two restricted fault models discussed above are two for which it happens that the ACC test is a complete test for the accumulator (and two other FU's). These models cover many possible faults in the 8085, but certainly not all of them, and there is no guarantee that they cover the majority of faults which occur in practice. The validity of the stuck-at fault model for LSI devices, in particular, has been questioned by Galiay et al[60]. In fact, in the absence of extensive information on the nature of faults which do occur in practice, the only way to assess the effectiveness of this particular test is by trial on a large number of 8085's which are known to contain accumulator faults. The effectiveness of a test such as this can therefore only be determined in the course of time.

It would have been possible to conduct a less extensive test on ACC which involved fewer extraneous FU's. For example, it would have been practical to perform a walking bit test on ACC, consisting of eight MVI A and OUT Instructions, which would not have involved the TMP register, ALU, Z flag or W and Z registers. Because fewer FU's would be involved in the test, masking faults would be much less likely to occur in the first place, and it would have been easier to account for all possible masking fault pairs in analysis of the test. However, in simplifying the test so that it is easier to allow for general faults in multiple FU's and so that it is therefore not necessary to adopt a restricted fault model, the set of ACC faults tested for would be reduced considerably (which implicitly restricts the fault model)! That is, if a walking bit test were to be performed on ACC, it would only be tested for stuck-at and some bridging faults which would then be the extent of the fault model. Therefore, it was considered to be

preferable to test each FU as fully as possible, with the minimisation of involvement of other FU's as an important, but secondary, consideration.

Ideally the fault model under which the ACC test is complete would not be restricted at all. The test should detect the most general set of 8085 faults which include a fault in ACC. However, the given ACC test is, in practical terms, the best one which could be constructed. Therefore the possibility that a fault in ACC is masked can only be discounted if the other FU's involved in the test are shown to be fault-free by subsequent tests. If all of these FU's can be shown to be free of an unrestricted set of faults then ACC can be guaranteed to also be free of an unrestricted set of faults. In that sense the effectiveness of the ACC test depends on the effectiveness of the tests for the data input buffers, data output buffers, ALU, Z flag, and TMP, W and Z registers. However, in attempting to construct tests for these FU's the same difficulty arises as in the ACC test - namely, that no FU can be tested in isolation and therefore fault masking between FU's can occur if the most general class of faults is to be considered.

5.3.3.3 Variation of FU grouping

One approach which can be adopted in an attempt to overcome this difficulty is to logically isolate FU's by performing several almost identical tests in which just one of the FU's involved is varied. For example, to test the TMP register and ALU increment operation (which may be considered as a single FU for present purposes because they are always used together in the ACC test) a similar test to the ACC test, but using another register, could be performed. Thus, if no errors are detected during execution of the following test:

```
        MVI  B, 00H
LOOP:   INR  B
        STAX B
        JNZ  LOOP
```

after the ACC test has been successfully executed, then it may be assumed that the TMP/ALU increment function works correctly for data from the B register, as well as for data from ACC. Therefore, if a fault in ACC had been masked during the ACC test by a complementary fault in TMP/ALU, the same fault in TMP/ALU must also be masked by a fault in B. Thus if a fault did exist in ACC, it could only be masked by a fault in TMP/ALU if an equivalent fault (to that in ACC) existed in B. Such an occurrence must be considered to be extremely unlikely - much more so than the occurrence of a complementary pair of faults in ACC and TMP/ALU.

Inevitably, however, the B register test given above involves FU's other than B and TMP/ALU. In particular, it uses the register multiplexer (MUX), the register select logic, the address latch and the address buffers. The last two of these are used by the STAX B instruction to pass the contents of B out to the external address bus for observation. The STAX B instruction was used as a means of observing the contents of B to avoid passing the test data through the accumulator.

None of the four FU's mentioned above is directly involved in the ACC test and each could mask a fault in B. The likelihood of this occurrence is reduced considerably by repeating the test for each of the other registers in the register array. Thus if the multiplexer, for example, were to mask a fault in B, its fault must be

masked by equivalent faults in the five other registers in the array. However, there remain several other potential fault masking pairs which could not be eliminated in this way, including the multiplexer and the address latch, and TMP/ALU and the input buffer. These possibilities must be eliminated by subsequent tests before the register tests could be considered to be conclusive.

It may be appreciated from the foregoing discussion that the task of systematically eliminating all fault masking possibilities in this manner is extremely complex, even when only a few simple FU's are involved and fault masking between only two FU's is considered. Attempts to generate conclusive tests for all FU's in the 8085 by doing this were unsuccessful for several reasons:

(i) The large number of FU's necessarily involved in each test meant that a very large number of possible fault masking pairs existed for each FU test.

(ii) The 8085 contains very few FU's which are equivalent to the extent that one could be simply replaced in a given operation by another. Thus, with the exception of the registers in the register array, no two FU's could be tested by exactly equivalent instruction sequences using the same set of extraneous FU's. Elimination of fault masking pairs by varying FU groupings in this way therefore proved to be less than straightforward. It is expected that the task would be much easier in a device with a more regular architecture, and more "equivalent" FU's.

(iii) The fault model was too general. Because no restrictions were placed on the faults which were allowed in the DUT it was not possible to eliminate fault pairs on the grounds that

they were unreasonable or extremely unlikely, even though this may well have been so.

It is apparent, then, that a systematic approach to the generation of complete and conclusive FU tests in a device such as the 8085, for a general set of faults, is too complex to be tractable. This difficulty arises principally from the fact that it is impossible to gain access to FU's and test them without involving many other FU's.

However, a self-test routine for the 8085 was developed by application of the principles of functional testing and, as far as possible, of variation of FU grouping to eliminate possible fault masking pairs. There is no doubt that the resultant test procedure is far more thorough than any test performed on the CPU during the SDK-85 SA Procedure (particularly the free-run test). The failure of this exercise lay in the inability to develop a test which could be guaranteed to detect a very general set of faults.

5.3.3.4 Remaining tests

The details of development of test sequences for the remaining FU's in the 8085 are in many cases quite involved and will not be given here. The ACC and register tests described above illustrate the principles which were applied in developing each of the FU tests. As noted above the systematic approach to development of these tests was not entirely successful and most of the tests are inconclusive. Many of the FU tests include features which are a legacy of the attempts to develop the tests systematically. These features were left in the tests because it was felt that although they did not make the tests conclusive, they did at least make them more effective. The final version of the 8085 functional test procedure is listed in Appendix G.

Some general notes on the various FU tests are listed below.

(i) The emphasis in implementation of many of the tests proved to be on verifying data paths. The ACC test for example, consists in principle of verification that 256 distinct bytes of data can be stored in ACC. In implementation the test verifies that the data can be written from the internal data bus to ACC, then read from ACC back onto the data bus. Similarly the B register test verifies that data can be read from B, through the multiplexer, onto the data bus, written back through the multiplexer to B, then read from B to the address latch. In the course of the test procedure it was attempted to verify each such data path within the CPU, using as much data as practicable. For most data paths this could be achieved in the process of performing tests on the various FU's but a few test sequences were included specifically to test data paths which were not otherwise tested. Examples of such tests (referring to Appendix G) are "WZ register test", "BC and DE access from incrementer/decrementer", "Reading SP through MUX" and "Writing SP through MUX".

(ii) The self-test program is intended to be run as part of the SDK-85 SA procedure and must be preceded by the 8085 free-run test. This test exercises both the program counter and the sixteen bit incrementer, so no other tests were included specifically for these FU's.

(iii) The register array test differs slightly from that described in the preceding section. The six registers are initialised to different values so that the test also verifies the uniqueness of the registers, as suggested by Smith[124]. An instruction which moves the register to itself immediately after the

register is incremented was included in each register test. The instruction was included as part of an inconclusive series of tests which were designed to eliminate the possibility of any faults in the multiplexer and its transferral of data from registers to the internal data bus.

(iv) The temporary register (TMP) test was designed to verify that data written to TMP from the data bus can be correctly read back onto the data bus. The data path from the data bus through TMP to the ALU is tested during the ALU test.

In the ALU test all possible ALU operations are performed on all possible input data combinations, including both values of the carry flag where appropriate. The result of each ALU operation (that is, the contents of ACC and the flag register) can be observed on the data bus during execution of a POP PSW instruction after each operation. The data paths from TMP to the ALU and from ACC through ACT to the ALU each pass 256 distinct bytes of data during this test and are therefore completely tested by it. These paths are only ever used in the course of performing an ALU operation, so there is no need to test them separately from the ALU. In fact, it would be impossible to do so.

The DAA instruction is tested with all possible initial data values in the accumulator and the carry (CY) and auxiliary carry (AC) flags, with the result once again being observed during execution of a PUSH PSW Instruction.

(v) There are two possible means of writing data to the flag register, and three possible means of reading it back. Data can be written to the flags as the result of an ALU operation,

or loaded from the data bus during a POP PSW instruction. Data in the flag register can be read by the ALU (CY and AC flags only), read onto the data bus during a PUSH PSW or tested during execution of conditional jump, call and return instructions. During the ALU test each of these three methods of reading the flags is tested and conditional jump instructions were inserted into the test for that purpose. The "Flag/jump/call/return test" tests each of the three methods of reading the flag register after it has been loaded from the data bus. Thus all six possible read/write combinations for the flag register are tested.

(vi) In the test for the conditional jump, call and return instructions each flag flip-flop is set in turn and a sequence of all possible conditional instructions is executed. The execution path through the conditional instructions can be monitored by observing the address sequence appearing on the address bus. The flags are initialised by the execution of a POP PSW instruction, which retrieves the appropriate data from a table stored in ROM. A table is also stored in ROM which contains a sequence of return addresses for the conditional return instructions. If a conditional return is successful an address is popped from the table so that execution continues at the next conditional return instruction. Otherwise the stack pointer is incremented twice, which in effect also pops the entry from the table.

(vii) The temporary registers W and Z could not be tested by the usual method of writing and reading back 256 distinct bytes of data because there is no means of storing internally generated test data in W or Z. The only way in which data can be written to these registers from the internal data bus is by

execution of an instruction containing immediate data or an absolute address. As discussed earlier, it is simply not practical to store 256 such instructions in the test ROM. Therefore a simple "walking bit" test is performed on these registers.

Sixteen SHLD and LHLD instructions are executed, which load the specified address into W and Z and then transfer the contents of W and Z to the address bus for observation. The eight addresses used for the LHLD instructions were chosen so that they refer to ROM locations, so predictable data will appear on the data bus during execution of those instructions. Eight OUT instructions are also executed to verify that W and Z are both correctly loaded with the specified eight bit I/O address during execution of I/O instructions.

A walking bit test is also performed on the data path from the internal data bus, through MUX to the stack pointer (SP) because this path is only exercised by the LXI SP instruction, which specifies immediate data.

(viii) Only a very brief test is performed on the interrupt logic of the 8085 during the self-test program because a more extensive test would involve external hardware which is not tested until a later stage of the SA procedure. Similarly, the serial I/O logic is not tested because it would also require external hardware. The tests for these facilities are deferred until Stage III of the SA procedure.

The only tests on the interrupt logic which could be performed within the CPU were simple write/read tests on the interrupt

mask, the mask enable bit and the interrupt enable flag. This limitation on the test of the interrupt logic illustrates that the extent of the test performed on any FU is limited by the accessibility of that FU to the device's instruction set. Thus, even though the interrupt logic is quite a complex FU it can only be tested in a rather cursory manner at this stage because there is no way in which it can be exercised more extensively by the 8085 instruction set.

(ix) The final test in the 8085 self-test program is the instruction decoder test, in which all instructions not used in other tests are executed. By this stage of the test routine all FU's in the 8085 (except the instruction decoder) have been tested. Therefore the only purpose in executing all untried instructions is to verify that the instruction decoder and the timing and control logic function correctly for each instruction.

In the first part of this test unused arithmetic and logical instructions are tested. For each class of arithmetic or logical instructions the registers are initialised from a table stored in ROM, then a series of instructions involving each of the registers is executed. The data initially loaded into the registers was chosen so that each operation performed would influence the final result produced by the instruction sequence. This result is placed onto the external data bus for observation during execution of a POP PSW instruction. To test the memory addressing mode (the 'M register') the H and L registers are initialised with data such that the address contained in H and L refers to a ROM location. The data read from that address is therefore predictable.

The second part of the instruction decoder test consists of the execution of eight "immediate" arithmetic and logical instructions. The eight bytes of immediate data used by these instructions are unused opcodes of the 8085 which would otherwise not be read in through the data input buffers. Thus, remembering that almost all 8085 instructions are fetched and executed during the self-test program, all of the 256 possible data bytes are read in through the data input buffers during execution of the program. The buffers are therefore completely tested.

The many MOV instructions of the 8085 are tested next. The MOV instructions are executed in groups of seven which pass data around each of the registers. DCR instructions are inserted between the MOV instructions so that at the end of the seven MOV instructions, the registers contain different data. Thus the seven DCR instructions are tested as well. The contents of the registers are pushed at the end of each group so that they may be observed on the data bus. The H and L registers are initialised before every MOV r, M instruction so that the data read from memory comes from ROM.

Finally, several miscellaneous instructions are executed in a sequence such that the data appearing on the data and address busses depends upon the effect of each instruction. The sequence finishes in a chain of restart instructions, from RST 7 to RST 0 (stored in appropriate ROM locations), which causes execution to return to the start of the program.

Only two instructions (HLT and IN) are not executed during

the self-test program. The HLT instruction could not be tested because it would prevent the program from being executed repetitively, which in turn would prevent the observation of data with the signature analyser. The IN instruction could not be fully tested because data appearing on the data bus during its execution would depend on external, untested hardware. However, all FU's involved in the execution of an IN instruction are tested, and an IN instruction is executed during the routine, as explained below. Therefore, the level of confidence in the instruction should be quite high.

5.3.4 Implementation of the 8085 self-test program

As discussed earlier in this chapter the 8085 self-test program was intended to be run during, and as an extension of, the SDK-85 SA procedure. It must be executed after Stage I, in which the system kernel is verified, and before Stage II, in which the CPU is used to test other system components. It was therefore implemented as Stage IA of the procedure. An overall flowchart of the SA procedure, including Stage IA, is contained in Appendix D.

The two test programs for Stages II and III occupy most of the 2K byte expansion EPROM (A15), so it was necessary to store the 900 byte 8085 self-test program in the external 2708 EPROM. This EPROM had been added to the wire-wrap area of the SDK-85 at address 8000H, as noted in Chapter III, and had been ignored up until this time. To force execution of the self-test program after reset a fourth jumper plug was wired for the address selection socket (also described in Chapter III). This plug interchanges the CS0/ and CS1/ chip select lines and ties the A_9 and A_{10} address inputs to the expansion EPROM (A15) high. Thus, on reset execution commences at location 600H in

in A15, which contains a jump to the self-test program proper at 8000H. A15 also contains the chain of RST instructions (stored at 608H, 610H, ... , 638H) which is executed at the end of the self-test program. It may be recalled that the subroutines for the Stage III test program are stored from location 640H up, while the Stage II test program is located from 400H to 522H. The final arrangement of code in A15 is shown in the memory map, Figure 5.2.

It should be noted that the fact that the self-test program was stored in external EPROM is regarded as being incidental and of no real consequence to the investigations discussed in this chapter. The code was stored externally simply because there was not enough internal ROM space to accommodate it, and in other implementations the code would certainly be stored in the same ROM area as the Stage II and III test programs. Therefore, while the Stage I procedure should have been modified to test the data bus input buffers and the external ROM to fully integrate Stage IA into the SA procedure, this was considered to be neither necessary nor useful for present purposes.

As implied in the design of the self-test program the response of the CPU to the test is verified by the observation of signatures on the data bus (using both RD/ and WR/ as clocks) and on the address bus (using ALE as clock). In this way all data input to and output from the CPU, as well as all addresses referenced during the test are verified. In fact it is not necessary to verify input data (that is, sample data on the data bus with RD/) because all relevant input data comes from ROM which (in principle) will have been tested during Stage I. However, verification of input data does provide an extra level of checking in the system and results in a more explicit test on the CPU alone. It also serves to verify that the CPU is not being affected

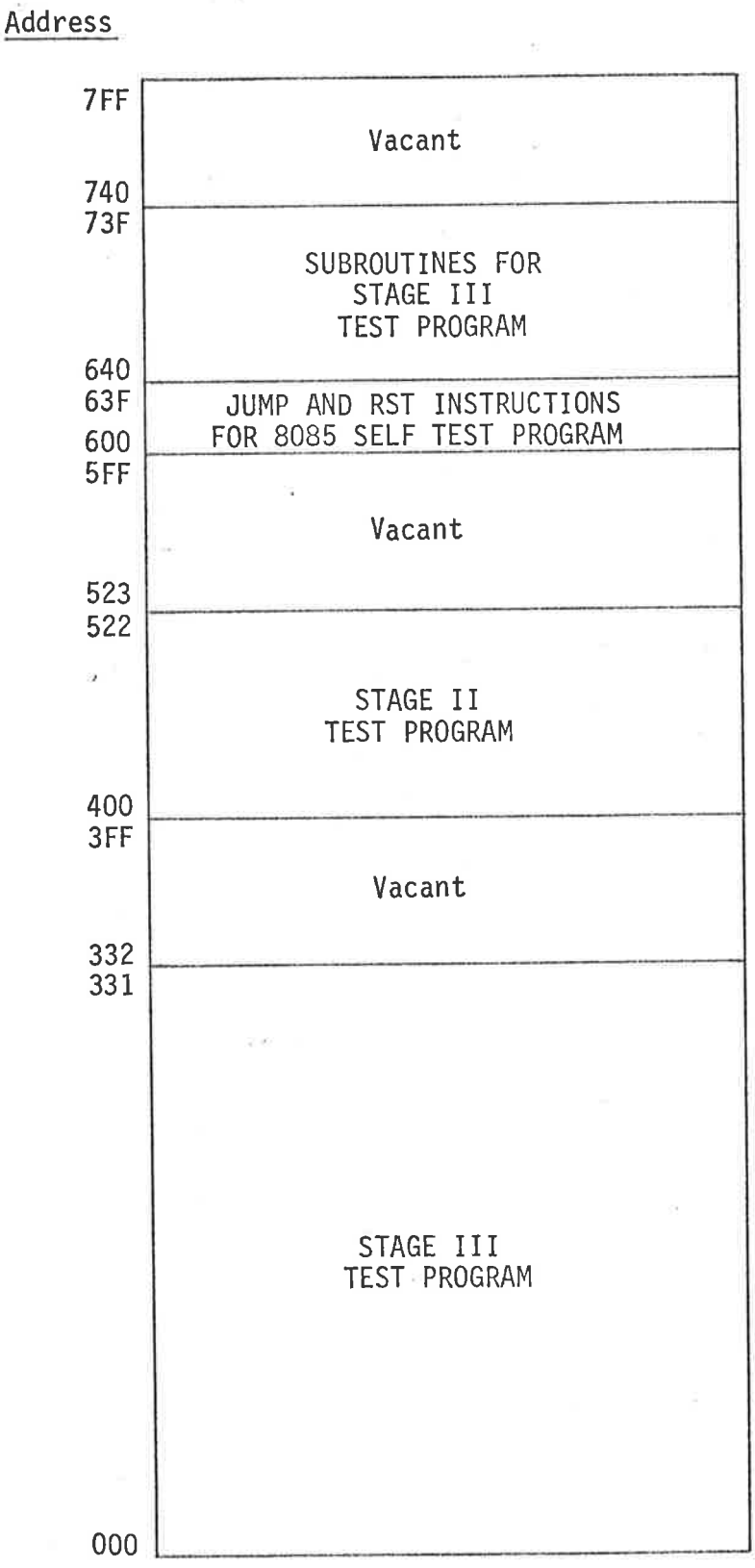


Figure 5.2. Final Memory Map of the Test EPROM, A15.

by unwanted feedback from devices which may be stimulated by CPU activity during the test.

To provide START and STOP pulses for the signature analyser with each of the three clocks used, IN and OUT instructions to addresses 30H and 38H (which activate the CS6/ and CS7/ outputs of the address decoder) are executed at the start of every pass through the program. The chip select pulses are gated with the IO/$\overline{M}$ status line by external logic to generate START (IOCS6) and STOP (IOCS7) pulses, as shown in Figure 5.3. The signature analyser is configured to start sampling on the negative (trailing) edge of START and to stop on the positive (leading) edge of STOP, so that data present on the data bus during execution of the IN instructions is not sampled. Thus, while the IN instruction is executed during the test program, its effects are not tested explicitly.

5.3.5 Evaluation of the 8085 self-test program

In the first instance the self-test program was verified by tracing execution through critical sections of the program with a logic state analyser, to verify that all tests were performed as intended. During this process it was observed that the ALU test and the sixteen bit register tests occupy by far the greatest part of the total test time.

Each pass through the test loop takes approximately ten seconds, which meant that it took thirty to forty seconds to observe each signature and ensure that it was stable. Therefore the total test time, for the observation of 27 signatures,was approximately fifteen minutes, which increased the total length of the SDK-85 SA procedure to 1 3/4 hours. Once again the process seemed to be very tedious, because it

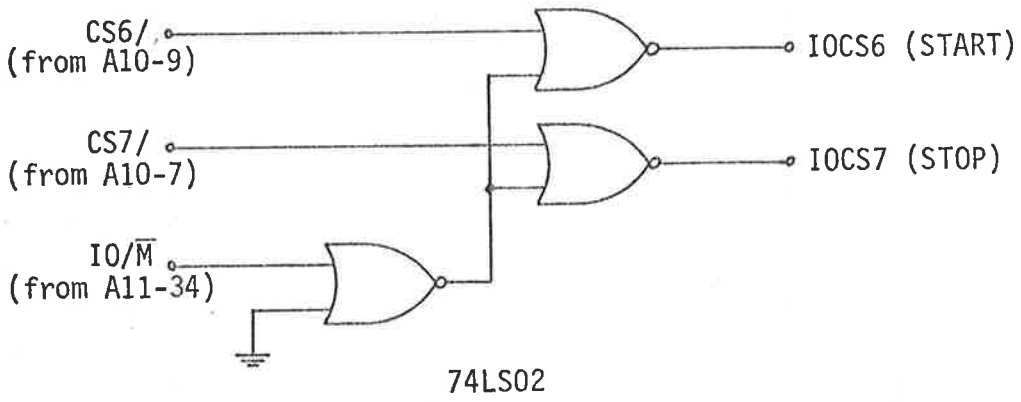


Figure 5.3. START and STOP pulse generation for the 8085 Functional Test.

was necessary to wait thirty seconds for each signature. It might be argued that a test for any single component in a system should not take as long as fifteen minutes because the extra information obtained from the test is unlikely to be sufficient to justify the time spent. Ideally, then, the 8085 test should be significantly shorter.

It has been noted by Wilkinson[34] and Bilton[35] that the effectiveness of any self-test program in detecting faults in a device is very difficult to assess. Application of the test to known-good devices merely verifies that the test yields consistent results on good devices, without giving any indication of its ability to detect faults. As discussed in the context of the ACC test the only way to fully assess the effectiveness of such a test is to run it on a very large number of faulty CPU's and observe whether or not it does detect the faults. Unfortunately, only a limited number of 8085's were available, only one of which was known to be faulty, so such a full assessment was not possible.

The test was run on six different 8085 devices. The first three of these were current Intel 8085A chips, which all produced the same set of data and address bus signatures, which are assumed to be the correct ones. These signatures are listed in Appendix H. The fourth device which was tested was the one which exhibited the TRAP fault described in Chapter III. It was also an 8085A and produced the same set of signatures as the other three 8085A's, which implied that it was fault-free. Thus the self-test program clearly failed to reject a device which was known to be faulty. However, this failure was expected. As discussed in Chapter IV, the nature of the TRAP fault is such that it could not be detected by signature analysis, irrespective of the tests performed by the self-test program, or the accuracy

of the device model on which it is based. There was no test for the TRAP logic included in the test program because no test could be. This failure was therefore not significant as far as assessment of the effectiveness of the test is concerned. Indeed, because the device produced correct signatures it can only be assumed that apart from the TRAP fault it was fault-free.

The last two devices to which the test was applied were Intel 8085 chips. These two chips produced the set of data bus signatures listed in Appendix H, but they both produced a set of address bus signatures which differed from that in Appendix H. Ordinarily this would imply that the two devices were faulty. However, because they produced the same address bus signatures and the data bus signatures were correct, it was assumed that the "incorrect" address bus signatures were observed because they were 8085, rather than 8085A devices, not because they were faulty. The fact that both sets of data bus signatures (with RD/ and WR/ clocks), including the V_{CC} signatures, were correct implied that the 8085's performed the same number of data bus operations as the 8085A's, and that the data transferred during each operation was also the same. This must be the case if the 8085 and the 8085A are operationally equivalent devices. Similarly, the different V_{CC} signatures obtained with ALE used as the clock implied that the 8085's issued a different number of ALE pulses during execution of the test program than the 8085A's did.

The conclusion which was drawn from these observations was that during the execution of some instruction, or instructions, both versions of the device execute machine cycles without issuing RD/ or WR/, during which one of the versions does issue an ALE pulse, and the other does not. This must be the case because only one ALE pulse

can be issued in each machine cycle, and RD/ and WR/ cannot be issued during a machine cycle without ALE.

Inspection of the (assumed) 8085 state transition table revealed that machine cycles M2 and M3 of the DAD *rp* instruction are the only ones in the entire instruction set which do not involve a data bus transaction. It was concluded, then, that the difference (or one of the differences) between the 8085 and the 8085A is that one generates ALE during the last two machine cycles of the DAD *rp* instruction, whereas the other does not. A logic state analyser was used to monitor the execution of the DAD *rp* instruction by both versions of the processor. This showed that the 8085 strobes the address of the following instruction onto the address bus with ALE during both M2 and M3 of the DAD *rp* instruction, whereas the 8085A does not issue ALE during either cycle.

This difference between the two versions is not documented explicitly in the available Intel literature. However, the fact that the 8085A does not issue ALE during M2 or M3 of DAD *rp* is documented in the form of a footnote to Table 3 of its data sheet[116]. The only reference to the 8085 (as opposed to the 8085A) which could be found was in the MCS-85 User's Manual[120] in which the behaviour of the 8085 during DAD *rp* is implied by the statement on page 2.3 that "ALE is present during [clock cycle] T_1 of every machine cycle".

This matter of the two versions of the 8085 illustrates that even minor differences between versions of a device, of little operational consequence, can be very important when developing a test for the device at the level of detail of the test developed for the 8085. The existing documentation of the difference between 8085 versions in exe-

cution of DAD *rp* is appropriate for its low level of importance to the general operation of the 8085. However, if devices are to be tested in any detail, it is necessary that there be readily available, detailed and rigorous documentation of each version of the device, at least at a level comparable to the state transition table provided for the 8080A[127]. If state transition tables for the 8085 and the 8085A had been available the DAD *rp* discrepancy could have been allowed for from the outset.

Apart from highlighting the need for detailed documentation, the 8085/8085A incident demonstrated the ability of the self-test program to detect a discrepancy in device bahaviour which might be interpreted as a fault. However, the absence of any 8085(A)'s with obvious functional faults has meant that the overall assessment of the effectiveness of the test program was very inconclusive. Certainly there are some sections of the 8085(A) (such as the timing and control logic) which are not explicitly tested by the procedure, but the significance of these omissions is not evident.

Nevertheless it is clear that the test developed for the 8085(A) is more thorough and effective than the free-run test and the "incidental" tests performed in Stage II of the SDK-85 SA Procedure (none of which detected the difference between the 8085 and the 8085A). There is no doubt that, with the knowledge of the type of tests performed during the functional test routine, a much higher level of confidence would be held in a device which had passed it than in one which had only passed the free-run test.

CHAPTER VI.

FUNCTIONAL TESTING OF A PERIPHERAL DEVICE

6.1 Aim of Development of the 8279 Test

As for the 8085, the primary aim of developing a functional test for the 8279, as an example of a peripheral device, was to produce a test routine which would be more thorough and effective than the one originally included in the SDK-85 SA Procedure. Although the earlier discussion of methods of developing systematic tests for LSI devices was directed particularly towards microprocessors, the principles involved and the virtues of functional testing apply equally to peripheral devices.

As a general class, however, peripherals are architecturally quite different from CPU's and have a very different mode of operation. They are generally designed to serve as slaves to a CPU and receive instructions under program control from the CPU. They are also commonly designed to interact with the world external to the microprocessor system. These differences meant that in attempting to apply the principles of functional testing to peripherals, different problems and limitations to those of microprocessors would be encountered. Thus a secondary aim of development of the 8279 functional test was to determine some of the particular problems which peripherals present.

6.2 The 8279 Functional Test

6.2.1 The approach to development of the test

It was intended from the outset to use the same approach to development of the 8279 functional test as was adopted for the 8085 - namely, to attempt to test each identifiable FU of the device as fully as possible without restricting attention to a particular fault model. The reasons for adopting this approach were discussed at the start of the description of the 8085 functional test. It has been seen that the approach was not entirely successful in the case of the 8085, but certainly did lead to a more effective test for the device. There was no reason to expect that the approach would be any more successful for the 8279, but it was expected that it would produce a more thorough test for the device than the ad hoc test originally developed as part of the SA procedure.

6.2.2 The 8279 functional model

It is a noticeable trend that the documentation supplied by device manufacturers for their peripheral devices is much less extensive than for their microprocessors, despite the fact that many peripheral controllers are as complex as some microprocessors. Thus, while Intel give a fifty page description of the 8080A and its operation in the 8080 Microcomputer System User's Manual[127], formal documentation of the 8279 consists of a twelve page data sheet[116]. As we have seen, the 8080A documentation includes a functional block diagram of the device and a clock-cycle by clock-cycle description of activity within the device in terms of the block diagram. In contrast, the 8279 data sheet briefly (and incompletely) describes the overall effects of each device command, as loosely related to its functional block diagram. Certainly no information as detailed as the 8080A

state transition table is given. In the absence of this type of detailed information about device operation it was only possible to base the functional model of the 8279 on its functional block diagram, which is shown in Figure 6.1. It may be observed that the functional block diagram does not contain a lot of detail. However in comparison with those supplied for many other peripherals (such as the 8271 floppy disk controller[116]) it is quite informative.

As noted above, one characteristic which distinguishes peripheral devices from CPU's is that they are intended to function as slaves - to act on commands received from a controlling device (the CPU) rather than fetch and execute instructions of their own accord. They must therefore be stimulated under program control by the CPU. The stimulus data will consist of a series of valid commands and test data, selected to exercise the various functions of the device. The 8279 has only eight valid commands to which it responds. These are:

Set keyboard/display mode;
Program clock prescaler;
Read FIFO/sensor RAM;
Read display RAM;
Write display RAM;
Set display write inhibit/blanking;
Clear;
End interrupt/set error mode.

In addition, it is possible to read the status register, to read data from the FIFO/sensor RAM, and to write data to or read from the display RAM. These commands are the basic tools which may be used to stimulate the device and exercise the functional units within it.

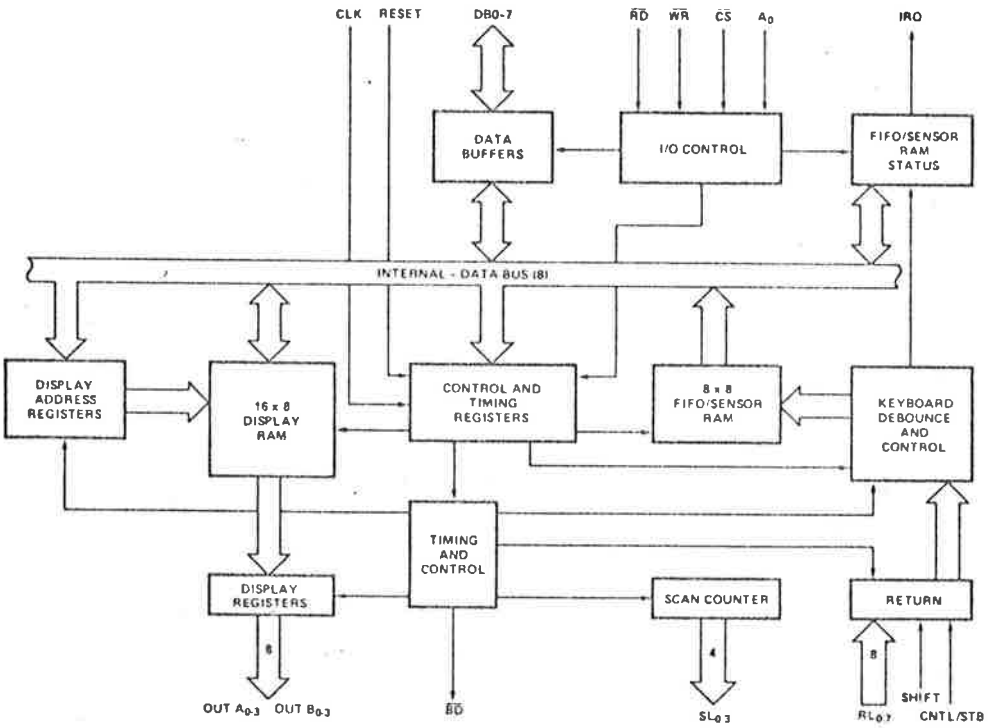


Figure 6.1. 8279 Functional Block Diagram.

It is immediately clear that this set of commands is much less extensive than the 8085 instruction set and must therefore provide much less flexibility in stimulating the device under test.

Examination of the 8279 functional block diagram and the description of each of the commands given in the data sheet also reveals the following significant differences between the 8279 functional model and the 8085 functional model discussed earlier in this chapter:

(i) The 8279 functional block diagram contains less detail than that of the 8085. For example, in the diagram of the 8085 all CPU registers are shown, as are all internal data paths. In the 8279 diagram, however, individual registers and the data paths between them are not shown, but are included within larger, more complex modules. Because of their greater complexity these modules must be harder to test than the 8085 FU's, many of which were single registers. The exact functions of the modules in the 8279 are also not as well defined as the functions of the 8085 FU's.

Since the effect of each 8279 command is broadly specified in terms of these modules, and no finer detail was available, it was apparent that the FU's of the 8279 could be no smaller than the modules shown in the functional block diagram. The individual FU tests for the 8279 would therefore be more complex and less effective than those of the 8085.

(ii) Access paths to the 8279 modules are less direct than those to 8085 FU's. Whereas most FU's in the 8085 could be accessed almost directly from the internal data bus, data paths in the 8279 are fewer and less direct. Consequently, while data can

be quite easily written to and read from some 8279 modules (such as the display RAM), others are not directly connected to the internal data bus and can only be indirectly exercised through several other modules. This, once again, meant that some modules would be more difficult to stimulate and hence, test than the 8085 FU's.

(iii) It was seen in the case of the 8085 that the inability to test FU's in isolation was, to some extent, compensated for by the ability to test FU's in various groups, thereby reducing the probability that a fault in one FU would mask a fault in another. This variation in grouping was only possible because of the flexibility of the 8085 architecture and the ability of the device to perform many operations on several distinct, but similar FU's. This flexibility of the device is reflected in the size of its instruction set. However, as noted above, the 8279 "instruction set" is very much smaller than that of the 8085. This means that when the device performs an operation in response to a given command its modules tend to be used in fixed groups. Thus the return latches, keyboard debounce and control logic, and the FIFO/sensor RAM all tend to be involved in any keyboard operation and there is no test which could be performed on any one without involving the others. The display RAM is something of an exception to this "fixed grouping" property because data can be written into and read from it without extensively involving other FU's, so it can be tested as a separate and largely independent FU.

The consequence of these three properties of the 8279 - the complexity of its modules, the poor accessibility of some modules, and the inflexibility of its architecture - is that the functional units

of the device are effectively much larger than the modules shown in its functional block diagram. It is not possible with the available set of commands to individually exercise these modules or use them outside of their "fixed groups". In other words, these modules cannot be treated as FU's because they are not sufficiently independent. In fact, with the exception of the display RAM, the 8279 consists of only two FU's - one which performs the display control function, and one which performs the keyboard input function. It is only at this relatively high level that the FU's are sufficiently independent and can be individually exercised and tested.

6.2.3 Development of the 8279 test

For the sake of consistency with the approach adopted for the 8085, and in spite of the observations discussed above, a table was constructed which showed each 8279 command and each module or attribute of the device, with the intention of finding its simplest commands or instructions. Not surprisingly, this proved to be of very little value. It merely showed (as far as could be determined from the brief description of each instruction given in the data sheet) that each command acts on a specific, fixed set of modules, and that individual modules are affected by very few commands. Therefore, in seeking to exercise a given module there would be very little (if any) choice of which command or commands to use. Thus, while the consideration of which command is simplest, and hence least likely to give rise to fault masking, is relevant in principle, in practical terms it is completely irrelevant.

This result simply confirmed that to test the 8279 it would be necessary to consider it as consisting of just two independent FU's, both of which have several operating modes. The functional test must

therefore consist of a sequence of commands which exercise both FU's in each of their modes as fully as possible. That is, the test sequence must be composed of a series of normal commands to the 8279 which instruct it to perform its usual tasks of keyboard scanning and display multiplexing. The only flexibility allowed in the construction of the test is in the selection of the test data used for the keyboard and display RAM's. Thus the development of the 8279 test degenerated into an exercise in operating the device in each of its possible modes.

Ideally, the 8279 test would be a self contained, automatic test performed very early in the SA procedure, as the CPU, ROM and RAM tests are. However, because the 8279 is an I/O device, any thorough test for it must include the application of external input data (at the keyboard return line inputs) and the observation of output data (at the scan line and segment outputs). If the test were to be fully automatic it would be necessary to stimulate the return line inputs under program control (probably by looping parallel output ports back to the 8279 RL inputs), attempting to simulate operation of the key matrix. In view of the fact that the RL inputs are sampled synchronously with the scan line outputs, the simulation of realistic key closures (including key bouncing) in software would be quite difficult. In an automatic test of the type performed on the 8085 it would also be necessary to observe data appearing on the scan line and segment outputs during the test with the signature analyser. While this is possible in principle, in the course of the test the keyboard and display modes must be changed frequently, with unspecified effects on the scan line and segment outputs, so problems with signature instability could be expected.

It was therefore considered to be better to test the 8279 in its "natural environment" - that is, with manual keyboard input and display verification. This meant that the test should be included in Stage III of the SDK-85 SA procedure, together with other tests for which operator interaction is required. In fact, it could directly replace the existing tests 0 and 1 of Stage III, which comprise the original test for the 8279. In this form the test would not be as self-contained and independent as the CPU test, but no test for an I/O device could be. The important point is that the test is conducted at the start of Stage III, so the 8279 is tested before it is used in tests for other SDK-85 facilities.

In development of the 8279 functional test the aim, as for the 8085, was to test each FU as thoroughly as possible. In practice, as discussed above, this amounted to aiming to test each attribute and operating mode of the two 8279 FU's as fully as possible. In a sense this was a regression to the approach which was adopted to the development of the original device tests in the SA procedure, of testing device operation as thoroughly as reasonably convenient. Consequently, the 8279 functional test turned out to be similar in many respects to the original 8279 test (tests 0 and 1 of Stage III). The reason for this similarity is that the two FU's considered in the functional test are not much simpler than the device as a whole, which the original test addressed. However, because the functional test was designed to test each attribute and mode of operation of each FU, it must be more systematic and thorough than the original.

During development of the test difficulties were once again experienced in attempting to use the 8279 in some of its modes. In addition to those mentioned in Chapter III, several other undocumented

characteristics of the device were discovered. They included: that when the operating mode of the display is changed the keyboard FIFO character count is set to zero; that in sensor matrix mode a key closure is represented by a '1' in the sensor RAM, not a '0' as would be expected; and that the sensor closure flag is latched when a key is closed, and reset only when the corresponding interrupt is cleared. None of these characteristics constitutes unreasonable behaviour by the device, but they all should be documented - particularly the effective inversion of sensor matrix data. This once again illustrates the generally inadequate level of documentation of peripheral controllers.

The final test routine, which is listed in Appendix I, is nearly 900 bytes long. For the purposes of evaluation it was stored in the external 2708 EPROM at address 8000H, temporarily replacing the 8085 self-test program. The Stage III program was temporarily patched to include a jump to location 8000H immediately after initialisation, and at the end of the 8279 test the Stage III routine is entered again at the start of test 2 (the serial I/O test). The instructions for running the test are presented in Appendix J. If the test were to be included as part of the final SA procedure these instructions would replace those for tests 0 and 1 in the Stage III procedure (contained in Appendix F).

6.2.4 Outline of the 8279 test

A brief description of each of the steps of the 8279 functional test follows:

(i) A "clear all" command is written to the 8279 and tests are conducted to verify that the "display unavailable" flag (Du)

in the status register is cleared within the specified 160μs, that the keyboard input FIFO is empty and that the FIFO underrun flag is set if the FIFO is then read. A subroutine is then entered which waits for a character to be entered into the FIFO, which is taken as an indication from the operator that the display (which should consist of all segments being uniformly lit) is correct.

The display is then cleared again, but with 20H used as the clear code, so 'b' display segments should turn off. Du is checked again and the FIFO status is read, to verify that the previously entered character was not lost during the display clear operation. After a second keyboard entry (indicating approval of the display) the same sequence of tests is repeated, but with FFH used as the display clear code. After the third keyboard entry the next step of the test is commenced.

If an error is detected in any of these tests an error display routine is entered, which writes a unique error number to the display for each possible error. It is possible, of course, that an error message would not be displayed correctly because of a fault in the 8279, but any variation from the expected display sequence is sufficient to indicate that the device is faulty. Unique error numbers were used primarily as an aid in debugging the program, and also to identify particular errors (associated with interrupt operation) for which further action is required to determine whether the 8279 or the 8085 is at fault.

(ii) The second test in the sequence is a read/write test on the display RAM which is intended to verify that all RAM locations are unique and contain no stuck-at bits. The test also exercises the display RAM address auto-increment and direct select mechanisms. It should be noted that the display RAM test is an unusually simple and complete one, largely because of the good accessibility of this particular module (noted earlier).

(iii) The operation of the display blanking and write inhibit controls is tested, with input from the keyboard to confirm that the resulting displays are correct.

(iv) The status of the FIFO is read to confirm that it has been unaffected by the preceding tests and still contains all characters entered from the keyboard so far. A "clear FIFO" command is written to the 8279 and the status flags are read to see that they have all been cleared. The display RAM is also read to verify that it was not affected by the FIFO clear operation.

(v) A walking bit test is performed on the display output buffers. Each of the eight display segments is turned on in all display digits for approximately half of a second. An entry from the keyboard is taken as confirmation that the display sequence was correct.

(vi) In the next test the four encoded scan display modes - eight and sixteen digits, left and right entry - are tested. In each mode the sixteen hexadecimal digits form '0' to '9' and 'A' to 'F' are shifted across the display. The display scanning circuit in the SDK-85 does not decode the most significant scan line, SL3, so in the sixteen digit display modes the eight most significant digits ('8' to 'F') are superim-

posed on the eight least significant digits ('0' to '7'). In the eight digit right entry mode the entry of characters from the middle of the display is tested. In each case a keyboard entry is taken as an indication that the display sequence was correct.

(vii) The display is finally tested in the decoded scan mode. Again because of the particular display scanning circuit used in the SDK-85, only digits three and five are scanned in the decoded scan mode, while none of the keyboard rows are scanned. The display should therefore show only two characters, which were left in the display RAM after the previous test. This display is maintained for a fixed period of five seconds because the keyboard cannot be used to indicate that the display is correct.

(viii) The keyboard controller test commences with a "clear all" command to the 8279. The FIFO status is read to ensure that the FIFO is empty, and the state of the RST 5.5 interrupt input to the 8085 is read. If it is asserted an error message is displayed and the operator must then use a logic probe to trace the line from the 8279 IRQ output to the 8085 RST 5.5 input, to determine which device is at fault.

(ix) The next test checks the operation of the 8279 interrupt logic together with the 8085 RST 5.5 interrupt, which could not be tested during the 8085 test because it involves the 8279. The routine waits until a character is entered at the keyboard (as indicated by the FIFO character count in the status byte) and then checks that only one character has been entered. The state of the 8085 RST 5.5 input is read and, if an interrupt is pending (as should be the case) RST 5.5 is unmasked and interrupts are enabled. A test is conducted to verify that

the interrupt does then occur. The FIFO is then read and the RST 5.5 input status is checked to see that it is no longer asserted. Finally, FIFO status is read to verify that the FIFO is empty again.

An error detected in the interrupt tests could be due to a fault in either the 8279 or the 8085, so once again a logic probe must be used to isolate the faulty device if one of these errors occurs.

(x) The FIFO overrun mechanism is tested next. A routine is entered which waits until eight characters have been entered into the FIFO, and then waits until the overrun flag is set by the entry of one more character. An error is indicated by the display remaining blank after nine key entries.

(xi) The digit '0' is displayed in the right-most digit of the display, prompting the operator to press the '0' key. When the correct key code is read from the FIFO the display is updated to '1' and the operator must enter '1'. This process is repeated until all 22 keys have been entered in sequence.

(xii) The keyboard controller is re-initialised to operate in sensor matrix mode and to read the sensor RAM in auto-increment mode. Up to 64 "end interrupt" commands are written to the device to remove any interrupts resulting from the change in operating mode. All sensor RAM locations are then read to verify that they all contain 00H, indicating that no keys are pressed. The sensor closure flag (SC) is also read to verify that it is clear.

Once again '0' is displayed, prompting entry of the '0' key. When a key is pressed, as indicated by SC being set, the pres-

ence of a pending RST 5.5 interrupt is verified, an "end interrupt" command is written to the 8279, and the RST 5.5 input to the 8085 is checked again to see that the interrupt request has been removed. Row zero of the sensor RAM is then read and the data is subsequently checked to see that the key pressed was the correct one. The routine waits until the key is released (SC is cleared) and then clears the associated interrupt. If the key entered was not '0' the process is repeated until the correct key closure is detected.

After the '0' key closure has been detected and processed, '1' is displayed and the procedure described above is repeated for the '1' key. In this case, however, the auto-increment read mode of the sensor RAM is disabled and the key closure interrupts are removed by reading the sensor RAM. Once again the routine is repeated until the correct key is pressed. After '1' is entered the auto-increment read mode is enabled again and each of the remaining twenty keys is prompted in turn. For each character displayed, whenever a key is entered the sensor RAM is scanned to verify that the correct key was pressed.

(xiii) Finally, the "N-key rollover" and "2-key lockout" modes of the keyboard controller are tested. The device is re-initialised to operate in the N-key rollover, scanned keyboard mode and a loop is entered in which the display shows a character corresponding to the last key character entered into the FIFO. To test N-key rollover operation the operator must verify that the display is updated as each new key is pressed, whether or not any other keys are already pressed.

When the 'NEXT' key is pressed the "2-key lockout" mode is selected and a similar display loop is entered. The operator should then verify that if a key is pressed a second key closure is not recognized until the first key is released. When the 'NEXT' key is detected again the 8279 test has been completed and the Stage III test program is re-entered at the start of the serial I/O test.

6.2.5 Limitations of the 8279 test

The test described above does test the 8279 in most of its operating modes, exercising most of the attributes of each functional unit. However some sections of the device, including the clock prescaler, the keyboard (in the decoded scan and strobed input modes), the special error flag and the sensor RAM, are not fully tested.

For the clock prescaler to be fully tested it would be necessary to program each value of the clock prescaler in turn, and for each one verify that the correct frequency ratio then existed between CLK and SLO. This would require the use of an oscilloscope or (preferably) a signature analyser, which would be inconvenient and would not fit in with the context of the test. It is also doubtful whether a signature analyser could be used successfully, in view of the difficulties which were experienced (and discussed in Chapter III) in attempting to obtain stable signatures on the scan line outputs with CLK as clock. Furthermore, the use of very small clock prescaler values would have created problems with key debouncing and made sequencing of the tests (which is achieved through keyboard input by the operator) very difficult.

The omission of tests on the keyboard in decoded scan and strobed

input modes was necessary because of limitations imposed by hardware provided on the SDK-85. The particular arrangement of the keyboard scanning circuitry means that, with only 22 out of a possible 64 keys present in the matrix, no keys are scanned in the decoded scan mode, so keyboard input could not be tested in that mode. The fact that only 22 keys are provided in the matrix also means that a complete test of the sensor RAM (one which tests all 64 bits) is not possible. There is simply no hardware which would allow the strobed input mode to be conveniently tested.

The "special error" mode flag of the 8279 could only be tested by the application of two simultaneous key closures in the N-key roll-over mode, which is virtually impossible to achieve manually. This could have been done if the input stimulus had been applied from a parallel output port rather than manually, but, as discussed earlier, there were good reasons for not adopting this approach.

It may be seen, then, that these limitations on the coverage of the 8279 test were imposed by external hardware restrictions of one type or another. It may also be noted that the display scanning hardware is responsible for difficulty in verifying displays in the sixteen digit display modes, because it causes characters to be superimposed.

6.2.6 Evaluation and discussion

Once again, in seeking to assess the effectiveness of the test developed, the problem arises that the only way to do this is to apply the test to a large number of faulty 8279's. Without such a range of known-faulty devices it is not possible to determine whether a particular step of the test, or the test as a whole, is effective.

Only one 8279 which was known to be faulty was available for evaluation of the test - the one discussed in Chapter III which incorrectly turned on the bottom segment of the left-most display digit. The fault was detected by the functional test procedure, which was to be expected because it contains the same "character shifting" test which detected the fault in the original procedure (that is, test 0 of Stage III). No other errors were detected during the functional test. While the test did detect the known fault in the device, this success is once again not very reassuring because none of the other steps of the test procedure detected the fault, which only appears with certain display patterns. One cannot help but feel that if a different character sequence had been used to test the four display modes then the fault would not have been detected. Thus the fault appears to have been detected more through good luck than good planning, which implies that the test is not sufficiently thorough or systematic. Other 8279's which were tested showed no errors during the functional test.

Therefore, as in the case of the 8085 functional test, the limited evaluation of the 8279 test which could be performed was very inconclusive. Like the 8085 test, however, it is clear that the new 8279 test is a more thorough and systematic test than the one originally performed in the SDK-85 SA procedure, and would instil a higher level of confidence in an 8279 which passes it. However, because the principles of functional testing were applied less successfully to the 8279 than to the 8085, the 8279 functional test would appear to be less of an improvement over the original 8279 test than the 8085 functional test is over the original test for it (the free-run test).

Some of the difficulties which might be expected in the development of tests for peripherals in general did not prove to be signifi-

cant in the case of the 8279. Test execution time was not a problem because application of input stimulus and verification of output was performed manually, which meant that the actual CPU execution time for the test was insignificant in comparison with the time taken by the operator to perform his share of the test. In fact, the test can be completed in less than five minutes, so the updated Stage III can be completed in less than ten minutes. This is still much shorter than the earlier stages of the SA procedure, so the more thorough 8279 test does not involve a significant penalty in test execution time. In general this would not be the case. If signature analysis were to be used to verify output data in tests for other peripheral controllers, test times approaching the fifteen minutes taken by the 8085 test could be expected. In fact, because the stimulus data is applied to peripherals under program control, at a lower data rate than a CPU can fetch its own stimulus from ROM, peripheral tests might well be expected to take longer than CPU tests.

The problem of providing external I/O stimulus and verification was also not very significant for the 8279 because of the particular nature of the interfacing task which it performs. As discussed above, some limitations were imposed on the test by the lack of suitable hardware which could be used to test some operating modes and features. It should be noted, however, that those modes and features of the 8279 not tested could not be meaningfully used in the SDK-85 during normal operation because of this lack of hardware. Therefore, as far as application of the device in the SDK-85 is concerned, these omissions from the test procedure are of no consequence.

In general, however, when developing a functional test it can be useful to cause a device to perform an operation which is not normally

meaningful, just to exercise a particular FU. Therefore, if a hardware limitation in a system restricts the set of operations which a device can perform (whether meaningful in a particular application or not), it limits the flexibility available in developing the functional test - and it has been seen that flexibility can be very important in the development of functional tests. Thus minor I/O hardware limitations of the type observed in the SDK-85 can have an indirectly, but significantly, detrimental effect on the development of functional tests. This is a possible further disadvantage of the "natural environment" method of testing I/O devices discussed in Chapter IV.

It may be seen, then, that in some respects the 8279 was easier to test than general peripheral controllers might be exepected to be. This is a result of its simplicity relative to many other peripherals and of the fact that it is intended to provide the interface between the microprocessor and an operator. The 8279 is not, however, the simplest of all peripheral controllers available. The Intel 8259 interrupt controller[116], for example, performs a relatively simple function and has greater accessibility to internal registers than the 8279, so it would be even easier to generate a functional test for it. The 8279 is therefore not an atypical peripheral controller, and does share most of the characteristics of such devices, including: being essentially dedicated to a single task; being a slave device for a CPU; having a fixed, inflexible architecture; having a fixed I/O environment; and being incompletely documented. To that extent, the experiences with the 8279 described in this chapter are relevant to peripheral controllers in general.

6.3 Conclusions on Functional Testing

It has been seen that in the case of neither the 8085 nor the 8279 was it possible to achieve the ideal of functional testing - to test the device by completely testing all independent functional units within the device individually. It proved to be possible only to follow the general principle of the method (to a greater or lesser extent) to develop tests which were undoubtably more thorough and complete than the original tests for the devices, but which could not be proved to be more effective. In fact, in the limited trials performed, the two functional tests were no more successful in detecting device faults than the original tests. During development of the tests it was found to be necessary to use ad hoc techniques extensively, but this appears to be the case for all of the functional testing methods discussed in Chapter V.

The basic limitation on the success with which functional testing can be applied to a device is the accessibility (that is, controllability and observability) of its functional units. This, in turn, depends upon the architecture of the device, which determines whether it is possible to exercise and observe sufficiently simple FU's without involving many other FU's. In the case of the 8085 it was possible to compensate to some extent for accessibility problems by taking advantage of the flexibility of its architecture to exercise FU's in different groups, thereby minimizing the likelihood of fault masking. This was not possible in the case of the 8279 because its architecture, characteristically of peripheral controllers, is very inflexible. Clearly, the principles of functional testing would be most easily applied to a device with a very regular and flexible architecture.

The calculation of instruction complexities, to determine which

instructions are most suitable for use in functional test routines, proved to be of some use for the 8085 test, but much less for the 8279 test because there was little choice of which commands should be used to exercise a given FU. Even in the case of the 8085 test, considerations of instruction simplicity were often overridden by the need to produce efficient test code, and in later sections of the test were substantially ignored. There is no doubt that the principle of involving the fewest extraneous FU's in the test for any FU is important. However, in the absence of any specific fault model, it is not clear just how important it is.

It is implied in the reasoning of Ballard[93] and Srini[91] that there is a reasonable likelihood that a fault in one FU will be masked by a fault in one other FU, but that the likelihood of fault masking between three or more FU's is negligible. The proposition is certainly logical, but the point at which the likelihood of fault masking becomes negligible may be questioned. In fact, there is no evidence to support this proposition or any other such proposition concerned with such loosely defined faults and functional units. In view of the questionable degree of importance of minimising the number of FU's involved in tests, and the limited use which can be made in practice of assessments of instruction complexity in any case, it may be that such quantitative assessments are not necessary and that an intuitive assessment of instruction complexity would be sufficient for generating simple test sequences.

It was found that the time taken to perform the 8085 functional test was unacceptable. Most of the fifteen minutes required for the test is spent on the ALU test, so it is expected that equivalent tests for other eight bit microprocessors would take approximately

the same time. Equivalent tests for very complex peripherals might take even longer, for the reasons discussed in the preceding section. It may be that the benefits obtained by performing such a long device test are not worth the fifteen minutes of technician's time which they cost. In fact, it may prove to be more economical to simply replace the device every time than to spend the time testing it, given that the test is not conclusive. Certainly such long device tests will become less attractive as labour costs rise and device costs fall.

For the CPU test to be useful, then, its length should be reduced, either by reducing the thoroughness of the tests for some FU's (particularly the ALU) or by finding an even more efficient means of verifying input and output data than conventional signature analysis. If any FU test sequences were shortened the overall effectiveness of the CPU test must be reduced, which leads to the traditional tradeoff between test speed and test effectiveness. In this type of functional test in particular it would be very difficult to determine what the best compromise would be. As a more efficient means of observing data on the CPU busses during the test, a data serialising facility, such as provided on the HP5001A microprocessor exerciser[111][112] or the generation of signatures for parallel data, such as performed by the Paratronics 532 logic analyser[83], may be considered.

Finally, the investigations described in this, and earlier chapters showed the great importance of comprehensive documentation from device manufacturers for all types of LSI devices. Most obviously this must include a complete description of the behaviour of the device as observed at its output pins, so that it may be designed into and used in a system without requiring (as the 8279 did) experi-

mentation to determine exactly what it does. Nicholson[128] briefly discusses this need for more complete documentation of device behaviour, together with some of the reasons that it is difficult to achieve. Nevertheless, the omission of important details of operation from data sheets such as was observed in the case of the 8279 should not occur. Secondly, device documentation must include a detailed description of internal device activity (equivalent to the state transition table of the 8080A) if any form of functional test is to be developed for the device. It is also important that, while there is any possibility of different versions of a device being encountered in the field, device manufacturers clearly document all differences between the versions, even though they may not be operationally significant. In the course of this study three different versions of the 8085 were encountered, and current Intel literature only acknowledges the existence of one. The existence and nature of earlier versions of the device should at least be mentioned in data sheets for the current version.

In summary, it has been seen that the principles of functional testing serve as useful guidelines for the development of thorough tests for LSI devices in the context of signature analysis. The extent of the success achieved in developing a systematic device test depends very much on the architecture of the device in question. In this respect, and others, it is harder to generate systematic tests for intelligent peripheral controllers than microprocessors.

The ability to generate an effective functional test for a device relies ultimately on good accessibility to simple internal functional units and comprehensive documentation of the device. Therefore there

is little that a system designer can do to increase the ease with which a given device can be thoroughly tested. This is the responsibility of the device manufacturers and it is clear that unless they improve the level of documentation and make provision for device testability, as more complex devices are introduced it will become impossible to effectively test devices in the field, even to the limited extent to which the 8085 and 8279 were tested.

CHAPTER VII.

CONCLUSIONS

7.1 The Completeness of the "SA Solution"

The aims of this study were to investigate the problems of field-servicing microprocessor based digital systems, to assess the effectiveness of current methods, and to determine what further developments are required in this area. The literature survey revealed that signature analysis appeared to be the most promising solution to the problems of field servicing. Consequently, a trial implementation of SA was performed on the SDK-85 with a view to determining how complete the "SA solution" to the problems of field servicing is. It is now possible to present an overall assessment of the effectiveness of SA based on this implementation.

In practice SA showed all of the advantages claimed for it, including: being an inexpensive, fast method of resolving faults down to the component level; being very thorough (given sufficient effort at the design stage); and requiring little skill on the part of service personnel. However, several limitations were also apparent. It was found that fault resolution to component level could only be achieved by a great deal of attention to details of testability during the design of the system. The technique proved to be unable, in practice, to cope with certain faults. Most seriously, the provision for systematically testing LSI devices proved to be inadequate, and this appeared to be the greatest single limitation on the ability of the technique to isolate faults in microprocessor systems.

On the basis of this conclusion it was decided to investigate

means of providing more thorough tests for LSI devices during a SA procedure. Architecture-based methods of testing, and in particular functional testing, appeared to be the most popular approaches to the generation of thorough, efficient tests for microprocessors and were therefore considered to be most suitable for development in the SA context. Attempts were made to apply the principles of functional testing to generate tests for the 8085 and the 8279, the conclusions from which can be summarised as follows:

The extent to which a thorough test can be developed for a device is limited primarily by its architecture, and in particular, by the accessibility of simple functional units within the device. There is no doubt that the two tests developed under the functional testing guidelines were more thorough than the ad hoc tests used initially in the SA procedure but, without a fully systematic approach, the general success of the method in developing an effective test cannot be guaranteed. It is clear that if LSI devices are to be efficiently and effectively tested, some form of on-chip provision for easier access to internal functional units is required as a means of improving device testability. This is certainly true for functional testing, and it is difficult to see how any other device test method could be both effective and efficient on LSI devices unless some provision is made for on-chip testability to componsate for device complexity. The study of functional testing also showed that device documentation is generally inadequate and in some cases insufficient to allow the device to be used without experimentation.

It must be noted that these conclusions have been formed on the basis of only one trial implementation. Efforts were made to ensure that the implementation would be representative of a wide range of

microprocessor systems and, to that extent, the results obtained are considered to be representative. However, there is certainly a need for confirmation of these conclusions about both signature analysis and LSI device testing, based on further trial implementations.

Subject to this qualification it can be concluded that given sufficiently detailed attention to testability during system design and means of adequately testing individual complex devices, SA is very nearly the complete field service solution for current eight bit microprocessor systems. Unfortunately, satisfactory means of testing complex devices in general are not available and this, being the most severe limitation on the effectiveness of SA, now appears to be the most serious problem of field servicing microprocessor systems.

7.2 Future Prospects of Signature Analysis

It is characteristic of the microelectronics industry that the "state of the art" changes very rapidly, with device integration levels doubling every eighteen months to two years[6][23][129]. Therefore conclusions which may be drawn about eight bit microprocessor systems, which represented state-of-the-art technology in the late 1970's, may not be relevant to state-of-the-art systems in the early 1980's.

The above discussion of the effectiveness of SA was concerned specifically with eight bit microprocessor systems which are, and will be for some time, very widely used despite the current availability of more sophisticated and powerful sixteen bit microprocessors[22][24][130]. Therefore the conclusions discussed above will be directly relevant for

several years to come. However, it is appropriate to briefly consider current trends in the development of microprocessor systems, and the effects which these will have on the effectiveness and applicability of signature analysis and functional testing.

7.2.1 Trends in microprocessor system development

In recent years the most obvious development has been the steady increase in device complexity, as measured by the number of transistors integrated onto a single chip[23]. This increase in integration levels has allowed the development of a new generation of microprocessors, with the current state of the art being represented by the Motorola M68000 sixteen bit microprocessor which contains approximately seventy thousand active devices[129]. Over the next year or two the first 32 bit microprocessor will be introduced[24][123].

It is possible to discern certain trends in the architecture of these new microprocessors. Generally their architectures are more regular and orthogonal than their eight bit predecessors[24][130]. They contain several essentially general-purpose registers which can be used as operands in most CPU operations. Most instructions can access operands using a wide range of addressing modes, with very few "special case" instructions and addressing modes which only act on specific registers. The new devices include several more complex instructions which provide support for high level (and often structured) programming languages, one example being the instructions of the Motorola M68000 which link and unlink procedure stack frames[130][131]. The Intel iAPX 432 32-bit processor was designed specifically to support the high level language Ada[123]. Most of the new devices are microprogrammed, although the Zilog Z8000 is a notable exception[130]. Finally, the new devices variously include a range of capabilities

designed to allow the development of very large, complex systems. These capabilities, which increase the complexity of the behaviour of the devices, include memory management, instruction pre-fetching, and system and user modes of operation[130][131][132][133].

As LSI devices have become cheaper and as peripheral devices have become more powerful and intelligent, there has been a trend in system design towards distributed processing (or distributed intelligence)[130]. Apart from the more common use of intelligent peripheral controllers, this trend has also seen the increasing use of multi-processor systems (to the extent that the new microprocessors include instructions and hardware features intended to make the implementation of multi-processor systems easier[130][131][132][133]).

With regard to future developments, it has been predicted that it will be possible to integrate one million transistors onto a single chip by the mid-1980's. Patterson and Séquin[129] discuss the microprocessor of the mid-1980's (which they refer to as "P1985"), pointing out that system designers do not have the design tools or the imagination to fully exploit the capabilities which a one million transistor microprocessor could possess. This presents a problem of VLSI product definition, as predicted by Moore[6]. Patterson and Séquin predict that P1985, instead of being a very much more powerful CPU with an enormous and extremely powerful instruction set, will contain vast quantities of on-chip memory. Thus the greatest part of the chip will consist of a regular memory array, thereby making the most efficient use of the available chip area, at the same time taking advantage of the processing speed improvements which are possible with on-chip memory. The memory will be arranged in a multiple level heirarchy, ranging from relatively slow, dense primary memory to smaller quantities

of much faster memory used in various caches. The memory heirarchy will include error detection and correction and, significantly, will allow for extensive microdiagnostic routines. Thus it may be expected that devices of the mid-1980's will be designed with a view to integrating large portions of a system onto a chip, rather than simply producing CPU's which are many times more powerful than current sixteen bit devices.

Finally, trends in the economics of field testing should be considered. Device costs, system costs and labour costs can all be expected to change during the next few years, in a way which may make component level repair of microprocessor systems less attractive. It is beyond the scope of this discussion to predict whether component level repair in the field will become uneconomical, but as skilled labour rapidly becomes more expensive, the economics of field service must be continually reassessed to see whether an effective component level field service method is even wanted.

7.2.2 The applicability of SA to future systems

The trends and architectural changes discussed above will clearly affect the ability of the combination of SA and functional testing to isolate faults in state-of-the-art systems to the component level (assuming that this continues to be desirable). Different device architectures and greater device complexity will present different problems to those encountered in the SDK-85, so it is necessary to consider the relevance of the conclusions drawn from the SDK-85 implementation to future systems.

It has been seen that the two greatest limitations on the effectiveness of SA in the SDK-85 were the inability to observe signatures

at certain nodes and the inadequacy of device tests performed during the procedure. There appears to be no reason to expect that these would not also be the greatest limitations in more recent and future systems. The first of these two problems can be overcome, as for eight bit systems, by sufficiently careful design of the system in the first instance to ensure that data at all nodes in the system can be observed with a signature analyser. It may be expected that this will incur a penalty either in system performance or in system cost because of additional hardware requirements. However, there is once again no reason to expect that these penalties will be significantly greater than for a system such as the SDK-85.

Multiprocessor systems have the potential to create serious problems for signature analysis because they typically consist of several asynchronous subsystems, with no well defined control heirarchy. However, it is very difficult to generalise because there is no single established system architecture and particular difficulties will vary from system to system. The best approach for loosely coupled systems would be, as suggested in the "Designer's Guide to Signature Analysis"[95], to take advantage of the relative independence of each processor and test each subsystem separately, ensuring that its inter-processor communication mechanism in particular is fully tested. For this reason the multiprocessor interface should be as simple and well defined as possible. More tightly coupled systems will exhibit a greater degree of synchronism so the application of SA on a system-wide basis should be more feasible. Again, it is difficult to generalise, but it should be possible to avoid most problems by sufficient attention to detail during design of the system.

It would appear, then, that as for eight bit systems like the

SDK-85, the applicability of SA to more advanced systems can be ensured by sufficiently careful design for testability. The primary aim is still to achieve completely synchronous and predictable behaviour within the system but, because these systems will generally be more complex, this is expected to involve more design effort and time than for simple eight bit systems.

The issue of device testing represents a far greater problem, simply because devices will be very much more complex. For example, it would be impractical to perform a functional test equivalent to that of the 8085 on a sixteen bit processor, because the ALU is sixteen bits wide and has 2^{32} possible input combinations (not counting initial flag settings), compared with 2^{16} for the 8085. The difficulties presented by much greater device complexity are offset to some extent by the more regular and flexible architectures of the devices which must make functional testing easier, but they are still overwhelming.

It would appear, in fact, that functional testing as applied to the 8085 and the 8279 (with limited success) may not be a feasible proposition for more recent and future devices. The greater complexity of these devices must mean that accessible FU's within the devices are more complex and therefore, harder to test. It is only necessary to examine the literature currently available for the Intel 8086 sixteen bit microprocessor[116][132] to see that the functional block diagram and information about internal activity which are provided are much less complete than the information which is available for the 8080A. Indeed, as devices become more complex it will become impractical to supply as much information about their internal behaviour as is supplied for the 8080A. It should also be noted that as CPU instructions become more complex and oriented toward the support of high level languages[24]

they become removed from the internal hardware - effectively buffered from it by the microcode for the device. Thus, in processors such as the Intel iAPX 432 it will not be possible to exercise internal FU's with the device's instruction set because the instructions will not be defined in terms of their effects on internal FU's, but rather, in terms of their effects on high level data "objects" and processor resources[123]. Details of implementation of the processor may be obscured completely.

It is also to be expected that the algorithmic test methods, such as those of Robach and Gobbi[125] and Thatte and Abraham[36] will become ineffective for much more complex devices, either because the algorithms will become computationally infeasible or because the ad hoc decisions required during execution of the algorithms will be too complex to be dealt with. This is a direct result of the exponential increase in device complexity. The method of Thatte and Abraham faces the additional problem that it is based on a RTL model of the CPU and relies upon an instruction set description at that level. It would appear that CPU architecture will cease to be defined at that level, so the information required by the method will no longer be available.

Therefore it is once again clear that if devices are to be tested effectively there must be some provision made on-chip for testability. This is important for devices of equivalent complexity to the 8085, but is absolutely essential for devices of significantly greater complexity. None of the current test generation techniques can be expected to generate an effective and efficient test for such devices if they must be tested only from their external I/O pins.

Finally, the difficulties of testing I/O devices in particular will only increase as the complexity of the devices increases and they become able to handle much more complex I/O environments. There is no accepted current method of testing the I/O behaviour of these devices in the field in real time. The technique of "natural environment" testing discussed in Chapter IV is one method which may be developed, but without further investigation it is difficult to assess its potential. This whole aspect of device testing is very open.

7.3 Chip-Level Design for Testability

It may be seen, then, that signature analysis should be applicable to microprocessor systems of the future just as it is to small eight bit systems, but that its effectiveness will, more than ever, be limited by the problems of testing individual devices. Unless these problems are overcome neither signature analysis nor any other method will be successful in the field service environment.

The inevitable conclusion, and one which has been echoed many times in the course of this study and in the literature, is that some form of on-chip provision for testability is essential. This will obviously involve a penalty in the form dedicating some portion of chip area and possibly some I/O pins for testing purposes. However, given that device manufacturers are able to integrate more logic onto a chip than they can often use[6], the extra logic requirement for testing purposes is not considered to be a severe penalty. Input/output pins are a more precious resource, so no more than one or two (possibly multiplexed) pins could be used for testing.

There are many possible approaches to improving device testability. At the simplest level extra registers could be included on a chip to provide greater accessibility to internal FU's, and allow the external inputs and outputs of the device to be manipulated and observed. Such facilities could have improved the testability of the 8279 considerably but the approach may be less useful for devices of very much greater complexity, because they would require very many such test registers to provide adequate accessibility to internal FU's.

The functional conversion method described by Könemann, Mucha and Zwiehoff[78], in which on-chip logic is converted to function as a feedback shift register which takes signatures of FU outputs, thereby improving their observability, would again be suitable for relatively simple LSI devices. However, the difficulty of stimulating FU's so that they are meaningfully tested would appear to limit its suitability for more complex devices. Level Sensitive Scan Design, discussed briefly in Chapter II, is a third method of enhancing device testability, and is one which has the potential to provide the most complete fault coverage. It has the disadvantage that it requires that the device be considered at the gate level during both design and testing, which may not be practical for a device containing one million transistors or more. For such a device, test times would also be very large because of the need to shift data serially through each storage element in the device[78].

A fourth method of providing on-chip testability is that used in the Motorola M6805 single-chip microcomputer[134], which executes an internally stored test routine whenever the appropriate input pin is raised to eight volts. The outcome of the test is indicated on two lines of one of its output ports. The tests performed are similar to

the type conducted in the SDK-85 SA procedure, but there is no reason why in general more elaborate tests could not be performed. Perhaps the most complete solution to the problems of field servicing would be for all devices (or at least all complex devices) in a system to independently and simultaneously perform a self-contained test on themselves, giving a simple pass/fail indication at the end of the test, similarly to the M6805. These tests could conveniently be performed whenever the system is first powered up. It would then be unnecessary to test devices by forcing the CPU to execute a system-wide self-test program of any sort.

The most thorough and effective means of implementing these on chip test routines would be in the form of microdiognostics, as proposed by Patterson and Séquin[129]. It is possible to more effectively test each FU within a device by microprogram, simply because programming in microcode allows much greater accessibility to FU's. With the direct control over each FU provided by microprogramming the issue of device complexity becomes less important. As in the ideal functional test, each simple FU can be tested fully and independently, so the device tests would consist of a sequence of simple microprogrammed FU tests. The degree of accessibility allowed by the device's (macro) instruction set would no longer be a problem. If suitable provision were made in the design of the chip, all internal logic right up to the device I/O pins could be tested by the microdiagnostics. This may mean that it would not be necessary to test the external I/O behaviour of the device, so all of the difficulties of doing that would no longer be important.

A completely self-contained microdiagnostic test of each device has the additional advantage that the test routine for each device

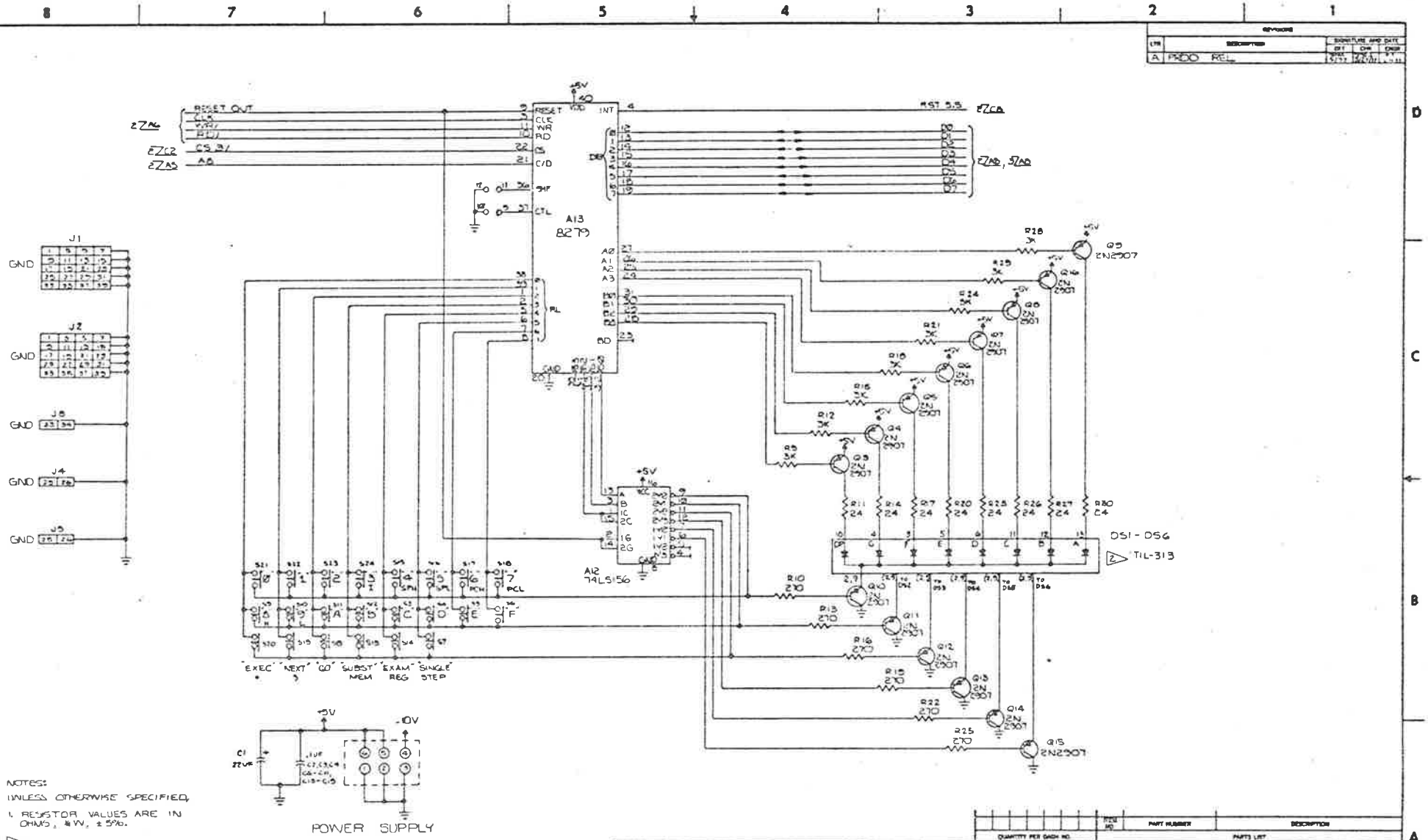
need only be developed once, by the chip designer. The effort required of the system designer to ensure system testability would therefore be very little - ideally, consisting only of ensuring that device interconnections can be verified.

Clearly, such extensive self-contained tests will require very large diagnostic routines, and until the technology to integrate one million transistors on a single chip becomes available, they may be impractical. Nevertheless, some form of on-chip provision for testability is required, even for current devices, and the sooner self-contained tests of this type can be implemented, the sooner will complete system testability be ensured. In the meantime, device manufacturers should consider some of the simpler methods of enhancing device testability.

The principle of design for testability is one which has been advocated for several years and is now well known. It must be concluded from this study that if current and future digital systems are to be effectively and economically tested in the field (and elsewhere) then the principle of design for testability must be considered at all levels of design. Furthermore, as technology allows an ever-increasing portion of complete systems to be integrated onto a single chip, the design for testability increasingly becomes the responsibility of the chip designer and manufacturer.

APPENDIX A

SDK-85 CIRCUIT DIAGRAMS



NOTES:
UNLESS OTHERWISE SPECIFIED,

1. RESISTOR VALUES ARE IN OHMS, ¼W, ±5%.

2. THIS PIN OUT ARRANGEMENT REPRESENTS 6 INDIVIDUAL 7 SEGMENT LED DISPLAYS (TIL 313). ALL ANODE CONNECTIONS OF THE CORRESPONDING SEGMENTS ARE WIRED TOGETHER. CATHODES ARE WIRED SEPARATELY TO EACH TRANSISTOR (Q10-Q15)

3. JUMPER 5 TO 5 AND 6 TO 6 WHEN BUS EXPANSION CIRCUITS ARE NOT INSTALLED.

SPARE GATE COUNT		
TYPE	REF DES	QTY

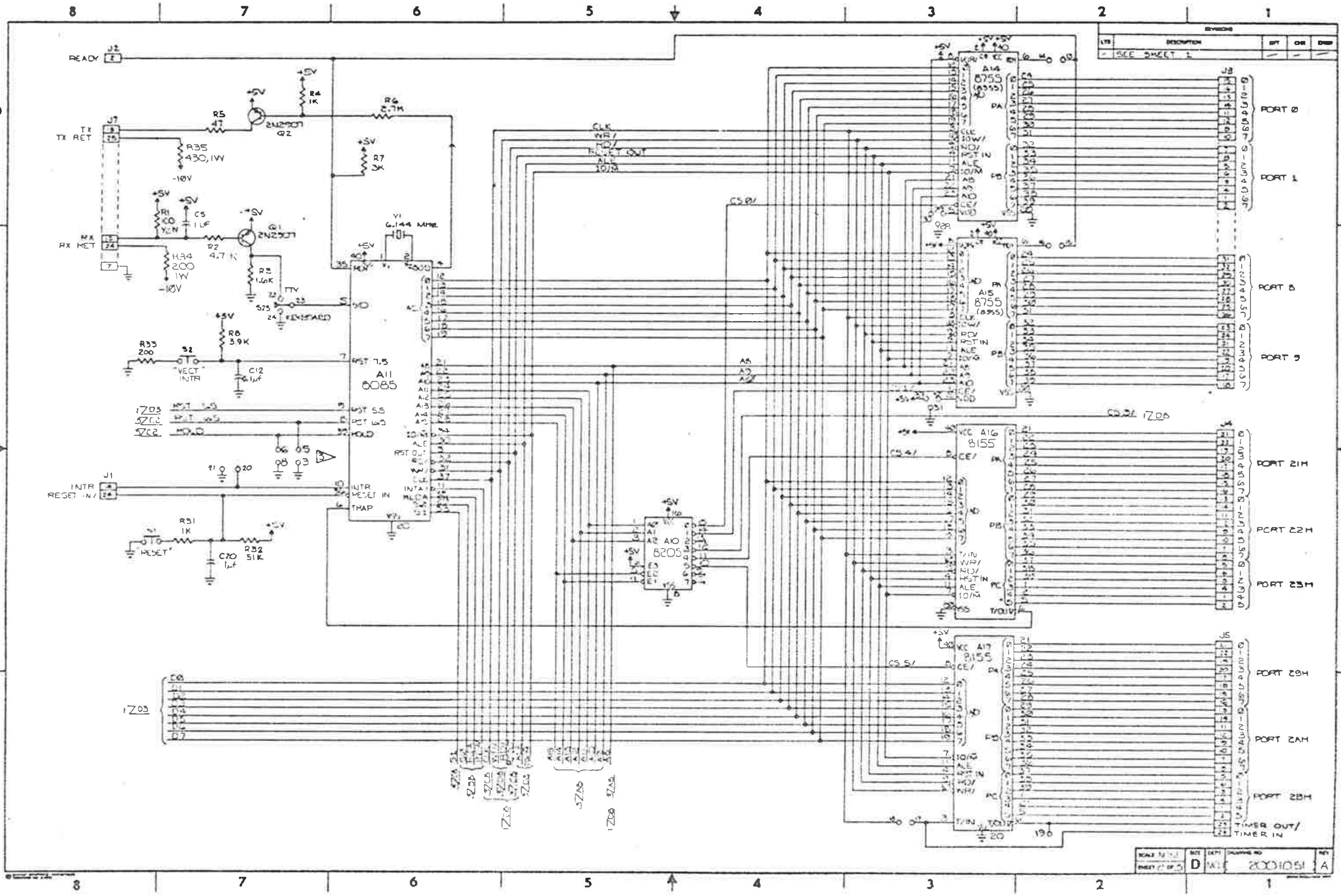
REF DESIGNATIONS	
USE USED	NOT USED
C20	
A17	
R33	
Q16	
DS6	
A3	
S15	

REV	DESCRIPTION	DATE	APPROVED
A	PROD REL	5/31	

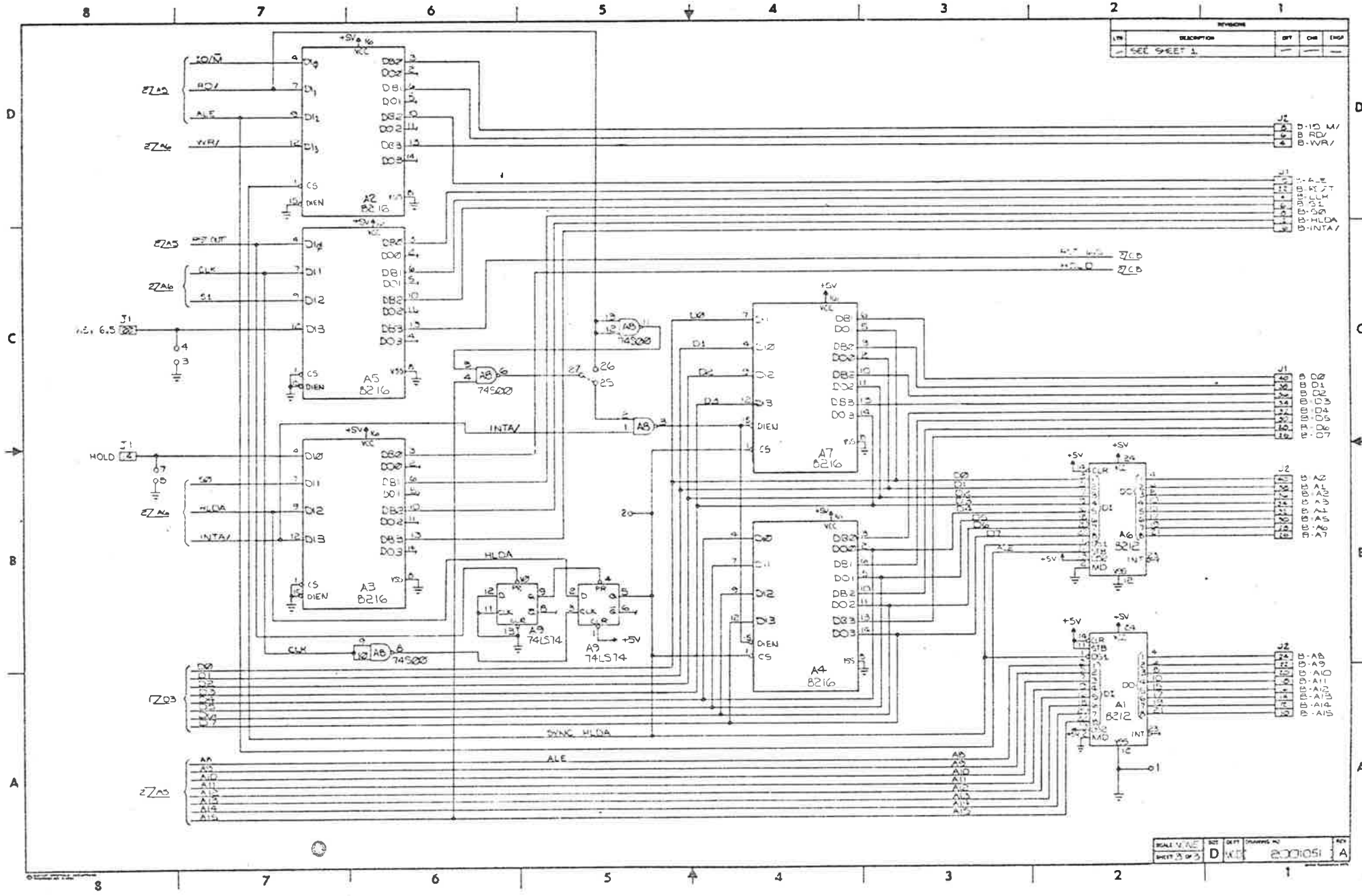
PART NUMBER	DESCRIPTION
4001117	SDK 85
NEXT ASSY	USED ON

SCALE: NONE
DRN BY: 5/7/76
CHK BY: 5/16/76
APP BY: 5/17/76
AUTH BY: 5/17/76
SHEET 1 OF 3

TITLE: SCHEMATIC SYSTEM DESIGN KIT
SIZE: D
DRAWING NO.: 2001051
REV: A


READY
TX
TX RET
RX
RX RET
R35 430,1W
R5 47
2N2907 Q2
R4 1K
R6 2.7K
R7 3K
Y1 6.144 MHz
R1 100 1/2W
C5 1 UF
Q1 2N2907
R2 4.7K
R34 200 1W
R3 1.6K
TTY
KEYBOARD
R33 200
S2 "VECT" INTR
R8 3.9K
C12 .1 µf
A11 8085
RST 7.5
RST 5.5
RST 6.5
HOLD
INTR
RESET IN
TRAP
S1 "RESET"
R31 1K
C20 1µf
R32 51K
CLK
WR/
RD/
RESET OUT
ALE
IO/M
A14 8755 (8355)
A15 8755 (8355)
A10 8205
A16 8155
A17 8155
CS 0/
CS 2/
CS 4/
CS 5/
PORT 0
PORT 1
PORT 8
PORT 9
PORT 21H
PORT 22H
PORT 23H
PORT 29H
PORT 2AH
PORT 2BH
TIMER OUT/
TIMER IN
SEE SHEET 1
D
2001051
A

240.


REVISIONS
LTR DESCRIPTION DRT CHK ENGR
A SEE SHEET 1
A2 8216
A5 8216
A3 8216
A7 8216
A4 8216
A6 8212
A1 8212
A8 74S00
A9 74LS14
HOLD
INTA/
HLDA
CLK
SYNC
ALE
IO/M
RD/
WR/
DEN
CS

APPENDIX B

LISTING OF THE SIGNATURE ANALYSER SIMULATION PROGRAM

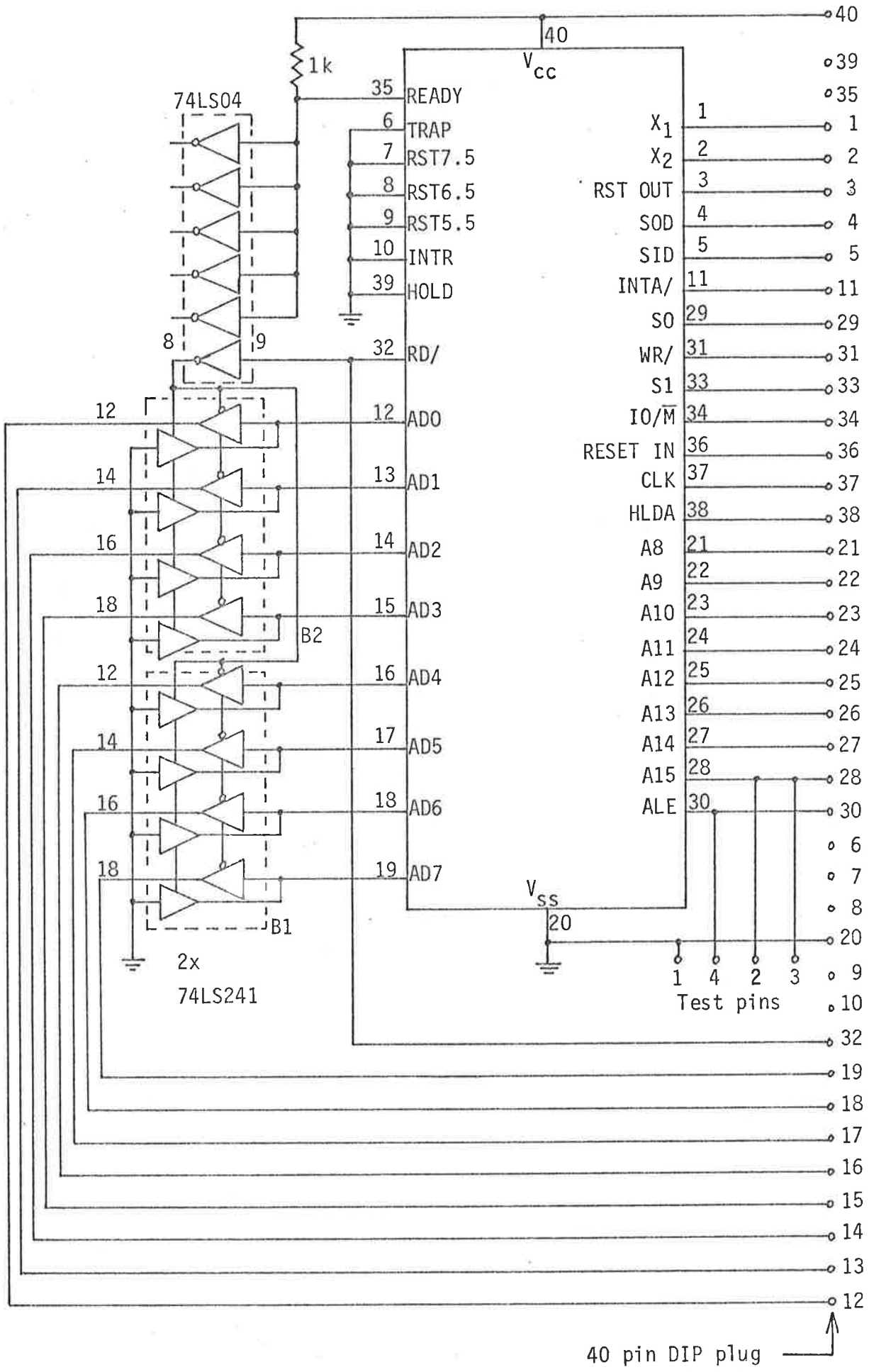
```
000006 PROGRAM SA(INPUT,OUTPUT);
000464
000464
000464 (*    THIS PROGRAM SIMULATES THE OPERATION OF A 16 BIT SIGNATURE
000464    ANALYSER.  THE NUMBER OF DATA SAMPLES (CLOCK PULSES), THE FEED-
000464    BACK BITS OF THE SHIFT REGISTER, AND THE SAMPLED DATA ARE READ
000464    FROM THE INPUT FILE.  THE RESULTANT SIGNATURE IS WRITTEN TO THE
000464    OUTPUT FILE.
000464      THE FIRST LINE OF THE INPUT FILE MUST CONTAIN AT LEAST TWO
000464    INTEGERS, THE FIRST SPECIFYING THE NUMBER OF CLOCK PULSES, AND
000464    THE SECOND SPECIFYING THE NUMBER OF FEEDBACK BITS FROM THE SHIFT
000464    REGISTER.  IF THE SECOND NUMBER IS NEGATIVE, THE 4 BITS USED
000464    BY THE HP5004A (7,8,12 AND 16) ARE ASSUMED.  IF THE NUMBER IS
000464    POSITIVE, (IT MUST BE < 17) THE NUMBERS OF ALL FEEDBACK BITS
000464    (BETWEEN 1 AND 16) MUST FOLLOW.
000464      THE REMAINING LINES OF THE INPUT FILE MUST CONTAIN ONLY 1'S
000464    OR 0'S, SPECIFYING THE INPUT DATA SAMPLES.  IF THE INPUT DATA
000464    RUNS OUT BEFORE THE SPECIFIED NUMBER OF CLOCK CYCLES, THE LAST
000464    BIT VALUE READ IS USED AS DATA FOR ALL REMAINING CLOCK CYCLES.   *)
000484
000464
000464 TYPE  BIT=0..1;
000464       HEXNUMBER=0..15;
000464       ZEROTO16=0..16;
000464
000464 VAR   CLOCK,NOFDBK,NCP:INTEGER;
000467       I,J:ZEROTO16;
000471       D:ARRAY[ZEROTO16] OF BIT;
000512       FEEDBACK:ARRAY[HEXNUMBER] OF ZEROTO16;
000532       CH:CHAR;
000533       N:HEXNUMBER;
000534       DATA:BIT;
000535       ERROR:BOOLEAN;
000536
000536 PROCEDURE INITIALISE;
000003 BEGIN
000003 ERROR:=FALSE;
000007 FOR I:=1 TO 16 DO D[I] := 0;
000020 READ (NCP,NOFDBK);
000026 IF NOFDBK<0 THEN BEGIN
000030   FEEDBACK[1]:=7;
000031   FEEDBACK[2]:=8;
000032   FEEDBACK[3]:=12;
000032   FEEDBACK[4]:=16;
000033   NOFDBK:=4
000033   END
000034  ELSE
000035   FOR I:=1 TO NOFDBK DO
000036     READ (FEEDBACK[I]);
000052 READLN;
000053 IF EOF THEN BEGIN
000054   WRITELN (' NO DATA ON INPUT FILE');
000062   ERROR:=TRUE
000062   END
000063 END;
                    (*INITIALISE*)
000071
000071 PROCEDURE GETCH;
000003 BEGIN
000003 IF EOLN AND NOT EOF THEN READLN;
000010 IF NOT EOF THEN READ(CH)
000016 END;
                    (*GETCH*)
000021
000021 PROCEDURE GETINPUT;
000003 BEGIN
000003 DATA:=0;
000007 GETCH;
000010 IF (CH<>'0') AND (CH<>'1') THEN BEGIN
000015   ERROR:=TRUE;
000016   WRITELN (' INVALID INPUT DATA AT CLOCK PULSE ',CLOCK:4)
000027   END
000030  ELSE
000031   DATA:=ORD(CH='1');
000036
000036 FOR I:=1 TO NOFDBK DO
000040   DATA:=(DATA+D[FEEDBACK[I]]) MOD 2;
000056 D[0]:=DATA
000056 END;
                    (*GETINPUT*)
000071
000071 PROCEDURE SHIFTRIGHT;
000003 BEGIN
000003 FOR I:= 16 DOWNTO 1 DO
000007   D[I]:=D[I-1]
000016 END;
                    (*SHIFTRIGHT*)
```

```
000026
000026 PROCEDURE WRITESIGNATURE;
000003 BEGIN
000003 WRITE (' SIGNATURE AFTER ',NCP:4,' CLOCK PULSES IS ');
000024 FOR I:=4 DOWNTO 1 DO BEGIN
000031   J:=4*I;
000033   N:=D[J-3]+2*D[J-2]+4*D[J-1]+8*D[J];
000053   CASE N OF
000057     0:CH:='0';
000061     1:CH:='1';
000063     2:CH:='2';
000065     3:CH:='3';
000067     4:CH:='4';
000071     5:CH:='5';
000073     6:CH:='6';
000075     7:CH:='7';
000077     8:CH:='8';
000101     9:CH:='8';
000103    10:CH:='A';
000104    11:CH:='C';
000106    12:CH:='F';
000110    13:CH:='H';
000112    14:CH:='P';
000113    15:CH:='U'
000113     END;
000135   WRITE(CH)
000141   END;
000143 WRITELN
000143 END;  ,
                    (*WRITESIGNATURE*)
000153
000153 BEGIN        (*MAIN PROGRAM*)
000153 INITIALISE;
000025 CLOCK:=1;
000026 WHILE (CLOCK<=NCP) AND NOT ERROR DO BEGIN
000032   GETINPUT;
000033   SHIFTRIGHT;
000034   CLOCK:=CLOCK+1
000034   END;
000036 IF NOT ERROR THEN WRITESIGNATURE
000037 END.


SIGNATURE AFTER   40 CLOCK PULSES IS 5158
```

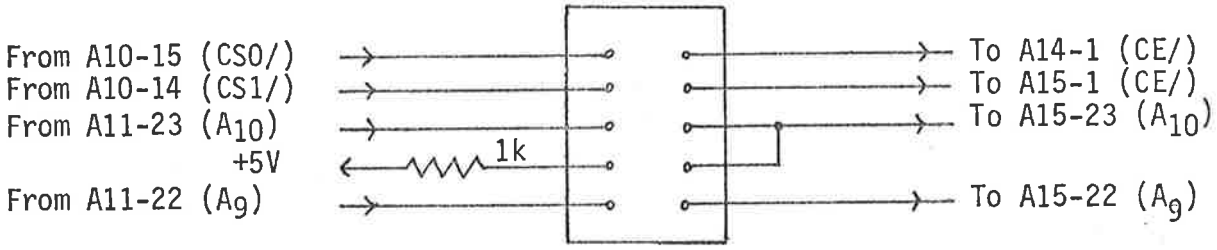
APPENDIX C

SDK-85 EXTERNAL TEST HARDWARE



(a) 8085 Free-run Adapter.

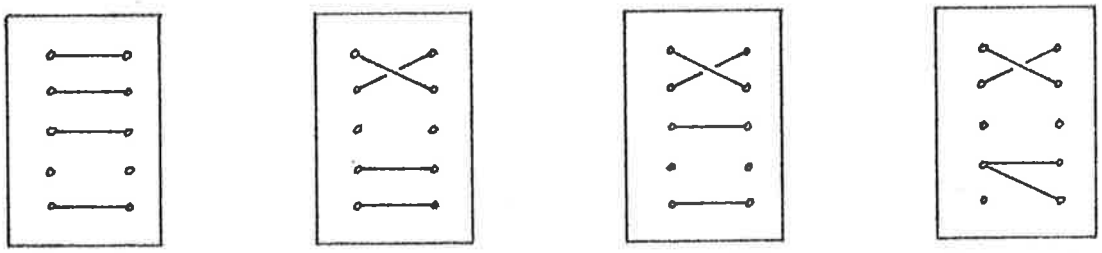
(i) Address selection socket (on the SDK-85 wire-wrap area.)



The following connections on the SDK-85 printed circuit board must be broken:

A10-15	to	A14-1
A10-14	to	A15-1
A11-23	to	A15-23
A11-22	to	A15-22

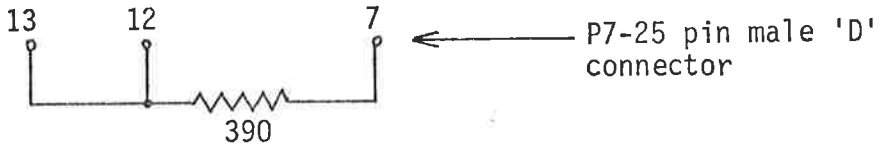
(ii) Jumper plugs.



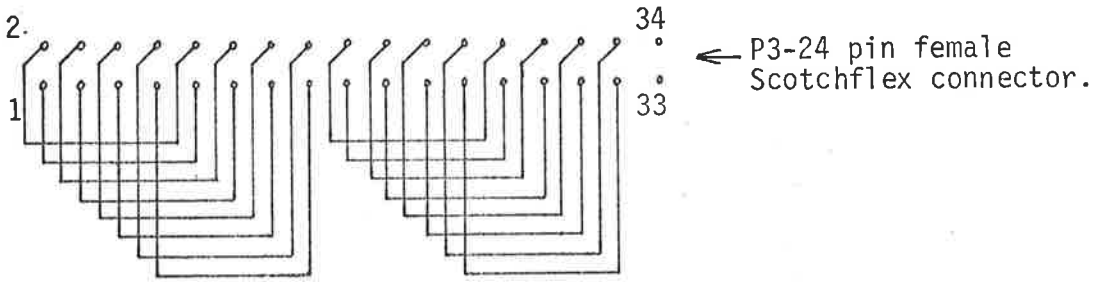
Normal operation	Stage II	Stage III	Stage IA
(WHITE)	(ORANGE)	(BLUE)	(GREEN)

(b) Addressing Changes.

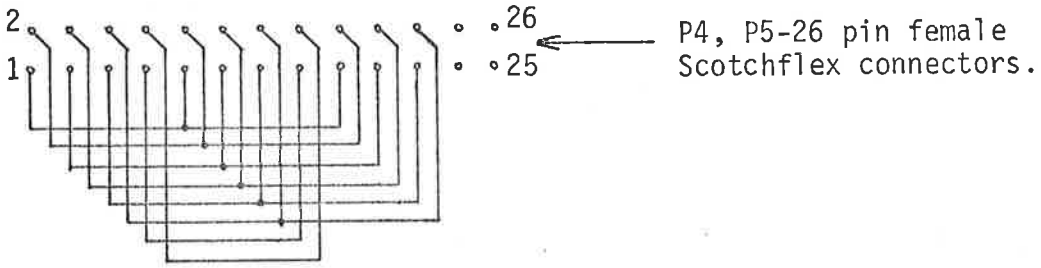
(i) Serial I/O test connector (P7).



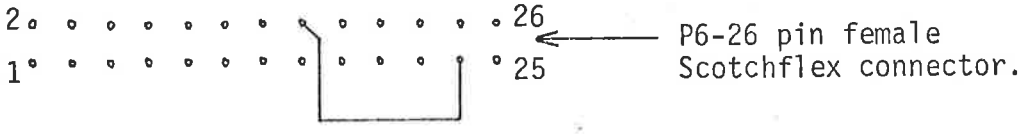
(ii) 8755 port loopback connector (P3).



(iii) 8155 port loopback connectors (P4 and P5).

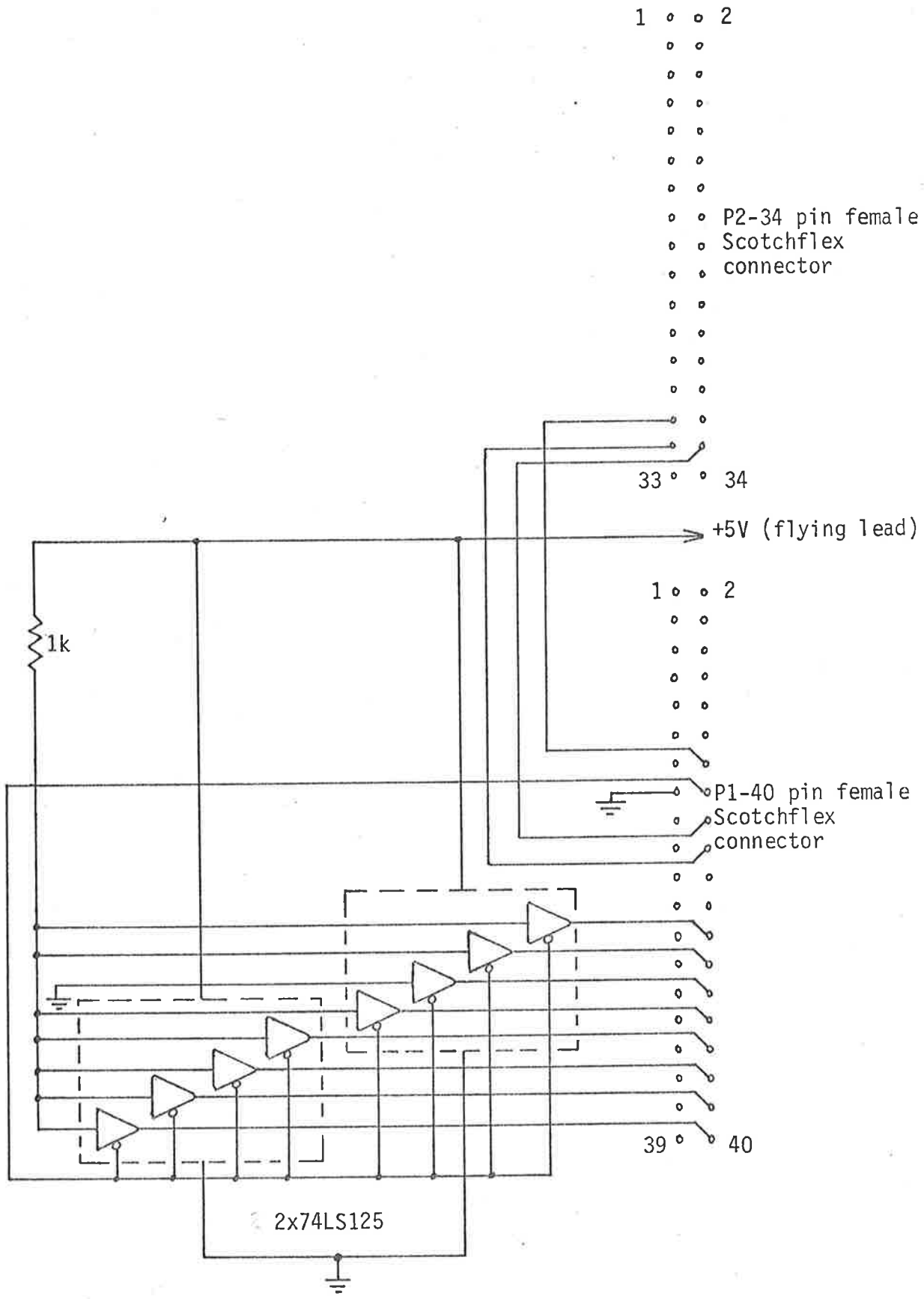


(iv) 8155 timer loopback connector (P6).



(c) Stage III Test Connectors and Hardware.

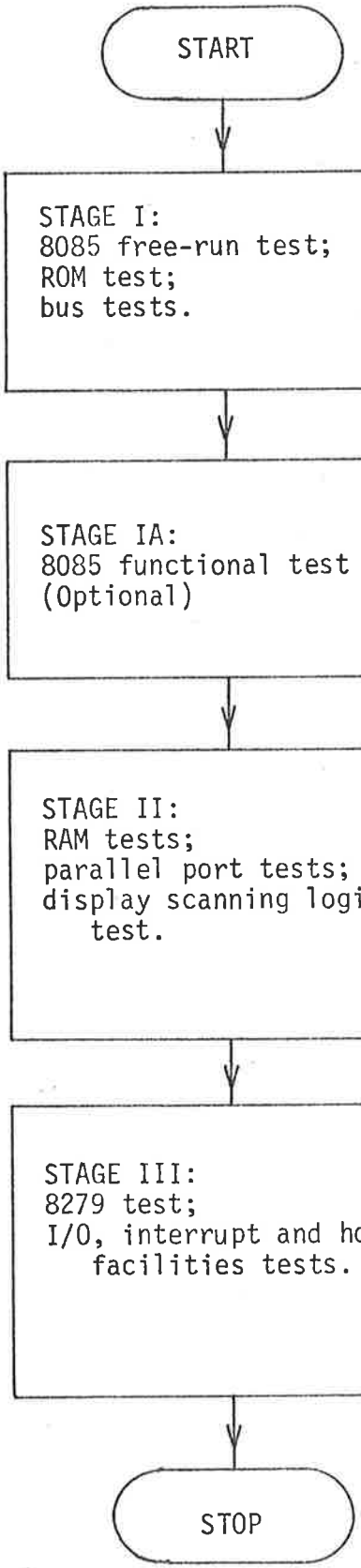
(v) HOLD, RST6.5 and INTR test connectors (P1 and P2).



(c) Stage III Test Connectors and Hardware (continued).

APPENDIX D

OVERALL FLOWCHART FOR THE SDK-85 SIGNATURE ANALYSIS PROCEDURE



Approximate duration

45 minutes

(15 minutes)

30 minutes

5 minutes
(10 minutes if the 8279 functional test is performed.)

80 minutes
(100 minutes with options.)

Overall Flowchart of the SDK-85 Signature Analysis Procedure

APPENDIX E

LISTING OF THE SDK-85 SIGNATURE ANALYSIS PROCEDURE SOFTWARE, STAGES II AND III

(SDK85S.V34)

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    1

  LOC  OBJ        LINE       SOURCE STATEMENT

                     1 ;#############################################################
                     2 ;
                     3 $       TITLE('SDK-85 Self Diagnostic V34')
                     4 ;
                     5 ;#############################################################
                     6 ;
                     7 ;
                     8 ;
                     9 ;          SDK 85 DIAGNOSTIC SOFTWARE
                    10 ;          ==========================
                    11 ;
                    12 ;
                    13 ;              COPYRIGHT (C) 1980
                    14 ;
                    15 ;                 M. J. LIEBELT
                    16 ;       ELECTRICAL ENGINEERING DEPARTMENT
                    17 ;            UNIVERSITY OF ADELAIDE
                    18 ;                  14/04/80
                    19 ;
                    20 ;
                    21 ;   THIS SOFTWARE FOR THE SIGNATURE ANALYSIS AND SELF
                    22 ; DIAGNOSIS OF THE SDK-85 IS STORED IN AN 8755 2K EPROM
                    23 ; AND IS DIVIDED INTO TWO SEPARATE PROGRAMS.  THE FIRST
                    24 ; IS STORED FROM 400H TO 5FFH WITHIN THE EPROM AND IS
                    25 ; EXECUTED IN STAGE II OF THE SIGNATURE ANALYSIS ROUTINE.
                    26 ; THE SECOND IS STORED FROM 000H TO 3FFH AND IS EXECUTED
                    27 ; IN STAGE III OF THE SIGNATURE ANALYSIS ROUTINE, USING
                    28 ; SUBROUTINES STORED FROM 640H TO 7FFH.
                    29 ;
                    30 ;   THE 8755 EPROM CONTAINING THIS TEST SOFTWARE IS TO
                    31 ; BE PLUGGED INTO A SOCKET AT A15 ON THE SDK-85 BOARD.
                    32 ;   FOR NORMAL OPERATION OF THE SDK-85 (I.E. TO EXECUTE
                    33 ; THE MONITOR ON RESET) THE WHITE PLUG MUST BE INSERTED
                    34 ; IN THE ADDRESS SELECTION SOCKET.
                    35 ;   THE STAGE II PROGRAM IS EXECUTED UPON RESET WHEN THE
                    36 ; ORANGE PLUG IS INSERTED INTO THE ADDRESS SELECTION
                    37 ; SOCKET.  THIS EXCHANGES THE CS0/ AND CS1/ CHIP SELECT
                    38 ; LINES TO A14 AND A15, MAPPING A15 INTO THE ADDRESS
                    39 ; RANGE 0000H TO 07FFH.  THE PLUG ALSO TIES ADDRESS INPUT
                    40 ; A10 TO A15 HIGH, SO EXECUTION STARTS FROM ADDRESS 400H.
                    41 ;   THE STAGE III PROGRAM IS EXECUTED ON RESET WHEN THE
                    42 ; BLUE PLUG IS INSERTED INTO THE ADDRESS SELECTION
                    43 ; SOCKET.  THIS ALSO INTERCHANGES CS0/ AND CS1/, BUT
                    44 ; LEAVES THE CONNECTION TO A10 INTACT, SO EXECUTION
                    45 ; STARTS AT 000H ON RESET.
                    46 ;   NOTE, THEREFORE, THAT IN THE FOLLOWING PROGRAMS 8755 #1
                    47 ; PORTS (00H - 03H) ARE THOSE ON A15, WHILE 8755 #2 PORTS
                    48 ; (08H - 0BH) ARE THOSE ON THE MONITOR ROM, A14.
                    49 ;
                    50 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    2
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE        SOURCE STATEMENT

 0000              51 ROM1A   EQU     00H     ;8755 #1 PORT A
 0001              52 ROM1B   EQU     01H     ;8755 #1 PORT B
 0002              53 DDR1A   EQU     02H     ;8755 #1 DDR A
 0003              54 DDR1B   EQU     03H     ;8755 #1 DDR B
 0008              55 ROM2A   EQU     08H     ;8755 #2 PORT A
 0009              56 ROM2B   EQU     09H     ;8755 #2 PORT B
 000A              57 DDR2A   EQU     0AH     ;8755 #2 DDR A
 000B              58 DDR2B   EQU     0BH     ;8755 #2 DDR B
 0020              59 CTRL1   EQU     20H     ;8155 #1 CONTROL REG.
 0021              60 RAM1A   EQU     21H     ;8155 #1 PORT A
 0022              61 RAM1B   EQU     22H     ;8155 #1 PORT B
 0023              62 RAM1C   EQU     23H     ;8155 #1 PORT C
 0024              63 TIM1LO  EQU     24H     ;8155 #1 TIMER LOW BYTE
 0025              64 TIM1HI  EQU     TIM1LO+1;8155 #1 TIMER HIGH BYTE
 0028              65 CTRL2   EQU     28H     ;8155 #2 CONTROL REG.
 0029              66 RAM2A   EQU     29H     ;8155 #2 PORT A
 002A              67 RAM2B   EQU     2AH     ;8155 #2 PORT B
 002B              68 RAM2C   EQU     2BH     ;8155 #2 PORT C
 002C              69 TIM2LO  EQU     2CH     ;8155 #2 TIMER LOW BYTE
 002D              70 TIM2HI  EQU     TIM2LO+1;8155 #2 TIMER HIGH BYTE
                   71
 0010              72 CS2     EQU     10H     ;8205 CS2/ ADDRESS
 0030              73 CS6     EQU     30H     ;8205 CS6/ ADDRESS
 0038              74 CS7     EQU     38H     ;8205 CS7/ ADDRESS
                   75
 1800              76 KDCD    EQU     1800H   ;8279 DATA REGISTER
 1900              77 KDCC    EQU     1900H   ;8279 CONTROL/STATUS REG.
                   78
 2000              79 RAM1    EQU     2000H   ;256 BYTE RAM #1 ADDRESS
 2800              80 RAM2    EQU     2800H   ;256 BYTE RAM #2 ADDRESS
                   81
 2100              82 TOPST   EQU     2100H   ;TOP OF STACK
                   83
 0050              84 EXEC    EQU     50H     ;'EXEC' KEY CODE
 0051              85 NEXT    EQU     51H     ;'NEXT' KEY CODE
                   86
                   87 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    3
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

                   88 ;##########################################################################
                   89 ;
                   90 ;
                   91 ;                  STAGE III
                   92 ;                  =========
                   93 ;
                   94 ;   THIS PROGRAM IS A SELF DIAGNOSTIC, DESIGNED TO BE RUN
                   95 ; AT THE LAST STAGE OF SIGNATURE ANALYSIS OF THE SDK-85. IT
                   96 ; INTERACTS WITH THE KEYBOARD AND DISPLAY TO TEST MOST OF
                   97 ; THE FACILITIES ON THE BOARD NOT TESTED IN STAGES I AND
                   98 ; II.
                   99 ;
                  100 ;
0000               101        ORG   0000H
                   102 ;
                   103 ;           INITIALISATION
                   104 ;
0000 F3            105        DI
0001 3EDF          106        MVI   A,0DFH    ;SET ALL MASKS, CLEAR RST7.5, SOD=1
0003 30            107        SIM
0004 310021        108        LXI   SP,TOPST  ;INITIALISE STACK POINTER
0007 3E00          109        MVI   A,00H     ;PROGRAM RAM PORTS FOR INPUT
0009 D320          110        OUT   CTRL1
000B D328          111        OUT   CTRL2
000D D302          112        OUT   DDR1A     ;PROGRAM ROM PORTS FOR INPUT
000F D303          113        OUT   DDR1B
0011 D30A          114        OUT   DDR2A
0013 D30B          115        OUT   DDR2B
0015 C33F00        116        JMP   PASTIR    ;LEAVE ROOM FOR INTERRUPT
                   117                        ; ROUTINES (THIS INSTRUCTION
                   118                        ; IS IN 15H,16H,17H)
                   119                        ;NOTE - TO EXECUTE THE 8279 TEST
                   120                        ; ROUTINE (KBDTST.V9) THIS INSTR-
                   121                        ; UCTION MUST BE CHANGED TO:
                   122                        ;     JMP   8000H
                   123
                   124
                   125 ;-------------------------------------------------------------------------
                   126 ;
                   127 ;               INTERRUPT ROUTINES
                   128 ;
                   129 ;
0018               130        ORG   0018H     ;RST 3 (INT) ROUTINE
0018 F5            131        PUSH  PSW
0019 3E01          132        MVI   A,01H
001B 320120        133        STA   INTFLG    ;SET 'INT' FLAG
001E F1            134        POP   PSW
001F C9            135        RET
                   136
0024               137        ORG   0024H     ;RST 4.5 (TRAP) ROUTINE
0024 F5            138        PUSH  PSW
0025 3E01          139        MVI   A,01H
0027 320320        140        STA   TRPFLG    ;SET 'TRAP' FLAG
002A F1            141        POP   PSW
002B C9            142        RET
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    4
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE        SOURCE STATEMENT

                  143
 002C             144         ORG   002CH     ;RST 5.5
 002C F5          145 RSTIR:  PUSH  PSW
 002D 3E01        146         MVI   A,01H
 002F 320220      147         STA   RSTFLG    ;SET 'RESTART' FLAG
 0032 F1          148         POP   PSW
 0033 C9          149         RET
                  150
 0034             151         ORG   0034H     ;RST 6.5
 0034 C32C00      152         JMP   RSTIR
                  153
 0038             154         ORG   0038H     ;RST 7 -- THIS VECTOR IS INCLUDED
 0038 C9          155         RET             ; IN CASE THE 8085 RESPONDS TO
                  156                         ; FFH (RST 7) IF IT IS NOT GIVEN
                  157                         ; AN INTERRUPT VECTOR IN TEST 7.
                  158
 003C             159         ORG   003CH     ;RST 7.5
 003C C32C00      160         JMP   RSTIR
                  161 ;
                  162 ;
                  163 ;-------------------------------------------------------------------------
                  164
                  165
 003F 210019      166 PASTIR: LXI   H,KDCC    ;PROGRAM 8279 FOR 8 CHAR, RIGHT ENTRY
 0042 3610        167         MVI   M,10H     ; ENCODED SCAN, 2 KEY LOCKOUT.
 0044 363F        168         MVI   M,3FH     ;CLOCK PRESCALER = 31.
 0046 3690        169         MVI   M,90H     ;WRITE DISPLAY RAM, AUTO-INCR.
 0048 3640        170         MVI   M,40H     ;READ FIFO.
                  171
 004A 36CD        172         MVI   M,0CDH    ;CLEAR FIFO & DISPLAY RAM.
 004C 3E01        173         MVI   A,01H     ;WAIT 1ms FOR DISPLAY CLEAR.
 004E CD4006      174         CALL  DELAY     ;
                  175 ;
                  176 ;              TEST 0
                  177 ;              ------
                  178 ;
                  179 ;  THIS TEST VERIFIES CORRECT DISPLAY OPERATION BY
                  180 ; SHIFTING CHARACTERS ACROSS THE DISPLAY. THE TEST
                  181 ; IS TERMINATED BY ENTERING ANY CHARACTER AT THE
                  182 ; KEYBOARD. AN ERROR CODE IS DISPLAYED IF ANY ERROR
                  183 ; IS DETECTED IN ACCEPTING THE CHARACTER FROM THE
                  184 ; KEYBOARD.
                  185 ;
 0051 25          186 TEST0:  DCR   H         ;(HL) = 1800H = KDCD
 0052 0E18        187 RPTCH:  MVI   C,24D     ;NO. OF CHARS. IN THE TABLE.
 0054 112807      188         LXI   D,CHRTAB  ;ADDRESS OF CHAR. TABLE.
 0057 1A          189 NXTCH:  LDAX  D         ;WRITE CHAR. TO DISPLAY RAM.
 0058 77          190         MOV   M,A
 0059 CD4D06      191         CALL  DLY500    ;WAIT 1/2 SECOND.
 005C CD5806      192         CALL  TSTKBD    ;CHECK FOR KEYBOARD ENTRY.
 005F B7          193         ORA   A         ;TEST RESULT.
 0060 C26B00      194         JNZ   END0      ;EXIT IF ANY ENTRY
 0063 0D          195         DCR   C         ;TEST FOR END OF TABLE.
 0064 CA5200      196         JZ    RPTCH     ;START AGAIN IF SO.
 0067 13          197         INX   D         ;ELSE POINT TO NEXT CHAR
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    5
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

 0068 C35700      198        JMP   NXTCH     ; AND DISPLAY IT.
 006B 24          199 END0:  INR   H         ;(HL) = KDCC.
 006C 3600        200        MVI   M,00H     ;PROGRAM KDC FOR LEFT ENTRY.
 006E B7          201        ORA   A         ;TEST KEYBOARD ENTRY.
 006F F27B00      202        JP    TEST1     ;NEXT TEST IF VALID ENTRY
 0072 2F          203        CMA             ;TAKE 2'S COMP. OF ERROR CODE
 0073 3C          204        INR   A         ;
 0074 47          205        MOV   B,A       ;SAVE ERROR CODE IN B.
 0075 0E01        206        MVI   C,01H     ;PUT TEST NO. IN C.
 0077 CD9F06      207        CALL  ERRDSP    ;DISPLAY ERROR MESSAGE
 007A 76          208        HLT             ;..AND STOP (KEYBOARD ERROR
                  209                        ;IS FATAL).
                  210 ;
                  211 ;
                  212 ;                TEST 1
                  213 ;                ------
                  214 ;
                  215 ;  THIS TEST VERIFIES KEYBOARD OPERATION BY DISPLAYING THE
                  216 ; CODE FOR WHICHEVER KEY IS PRESSED. INITIALLY THE DISPLAY
                  217 ; WILL SHOW THE CODE FOR THE KEY USED TO END TEST 0. THIS
                  218 ; TEST IS ENDED WHEN THE 'NEXT ,' KEY IS PRESSED.
                  219 ;
 007B 4F          220 TEST1: MOV   C,A       ;SAVE ENTERED CHAR IN C.
 007C CDD806      221        CALL  CLRDSP    ;CLEAR THE DISPLAY.
 007F 3685        222        MVI   M,85H     ;WRITE DISPLAY RAM LOC. 5
 0081 25          223        DCR   H         ;(HL) = KDCD
 0082 79          224        MOV   A,C       ;RETRIEVE CHAR.
 0083 E63F        225 DISPCH: ANI  03FH      ;CLR 'CHAR PRESENT' FLAG.
 0085 CDE206      226        CALL  CONVRT    ;CONVERT TO DISPLAY CODE.
 0088 77          227        MOV   M,A       ;WRITE TO DISPLAY.
 0089 CD4D06      228        CALL  DLY500    ;WAIT 500ms.
 008C CDEF06      229 WFIP:  CALL  RDKBD     ;TEST FOR KEYBOARD ENTRY.
 008F B7          230        ORA   A         ;
 0090 CA8C00      231        JZ    WFIP      ;WAIT IF NO ENTRY.
 0093 FE51        232        CPI   NEXT      ;NEXT TEST IF 'NEXT'
 0095 C28300      233        JNZ   DISPCH    ;ELSE DISPLAY THE CHAR.
                  234 ;
                  235 ;                TEST 2
                  236 ;                ------
                  237 ;
                  238 ;   THIS TEST VERIFIES OPERATION OF THE SERIAL I/O LINES.
                  239 ; SOD IS SET, THEN RESET AND SID IS TESTED IN EACH CASE TO
                  240 ; CHECK THAT IT FOLLOWS SOD. IF NOT, AN ERROR MESSAGE IS
                  241 ; DISPLAYED AND A LOOP IS ENTERED TO TOGGLE SOD EVERY 1ms,
                  242 ; TO ASSIST IN TRACING THE FAULT (E.G.WITH A CRO). THIS
                  243 ; LOOP WILL BE LEFT TO PROCEDE TO THE NEXT TEST WHEN THE
                  244 ; 'NEXT ,' KEY IS PRESSED.
                  245 ;   IF NO ERROR OCCURS, TEST 3 IS ENTERED IMMEDIATELY.
                  246 ;   THE TEST REQUIRES THE 25 PIN CANNON TEST PLUG TO BE
                  247 ; INSERTED INTO J7 TO LOOP SERIAL OUTPUT BACK INTO SERIAL
                  248 ; INPUT.
                  249 ;
 0098 0E02        250 TEST2: MVI   C,02H     ;PUT TEST NO. IN C.
 009A 1E80        251        MVI   E,80H     ;SET MSB IN E.
 009C 3EC0        252        MVI   A,0C0H    ;SET SOD.
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    6
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

 009E 30          253 SEROUT: SIM
 009F 57          254         MOV   D,A       ;SAVE SIM BYTE IN D.
 00A0 3E1E        255         MVI   A,30D     ;WAIT 150us FOR SID INPUT
 00A2 3D          256 DLYSID: DCR   A         ; TO SETTLE.
 00A3 C2A200      257         JNZ   DLYSID    ;
 00A6 20          258         RIM             ;TEST SID
 00A7 AA          259         XRA   D         ;COMPARE WITH DATA WRITTEN.
 00A8 A3          260         ANA   E         ;MASK MSB.
 00A9 C2B400      261         JNZ   SERERR    ;ERROR IF MSB NOT 0.
 00AC 7A          262         MOV   A,D       ;RESTORE SIM BYTE.
 00AD AB          263         XRA   E         ;COMPLEMENT SOD BIT.
 00AE F29E00      264         JP    SEROUT    ;REPEAT TEST FOR SOD = 0.
 00B1 C3CD00      265         JMP   TEST3     ;EXIT FROM TEST IF BOTH DONE.
 00B4 CDFF06      266 SERERR: CALL  CLRKBD    ;CLEAR KEYBOARD FIFO.
 00B7 0601        267         MVI   B,01H     ;PUT ERROR CODE IN B.
 00B9 CD9F06      268         CALL  ERRDSP    ;DISPLAY ERROR MESSAGE.
 00BC 7A          269 CHSOD:  MOV   A,D       ;LOAD SIM BYTE.
 00BD AB          270         XRA   E         ;COMPLEMENT SOD.
 00BE 57          271         MOV   D,A       ;SAVE THE BYTE AGAIN.
 00BF 30          272         SIM             ;
 00C0 3E01        273         MVI   A,01H     ;WAIT 1ms.
 00C2 CD4006      274         CALL  DELAY     ;
 00C5 CDEF06      275         CALL  RDKBD     ;CHECK FOR KEYBOARD ENTRY.
 00C8 FE51        276         CPI   NEXT      ;SEE IF 'NEXT' ENTERED.
 00CA C2BC00      277         JNZ   CHSOD     ;CONTINUE WITH TEST IF NOT.
                  278 ;
                  279 ;                TEST 3
                  280 ;                ------
                  281 ;
                  282 ;   THIS TEST VERIFIES THE INPUT CAPABILITIES OF THE I/O
                  283 ; PORTS. TEST PLUGS MUST BE INSERTED INTO SOCKETS J3, J4 AND
                  284 ; J5 TO CONNECT TOGETHER THE A, B AND C PORTS OF EACH PARALLEL
                  285 ; I/O DEVICE.  PORTS A ARE FIRST PROGRAMMED AS OUTPUTS AND A '1'
                  286 ; IS WALKED ACROSS EACH PORT, WHILE THE B (& C) PORTS ARE READ
                  287 ; TO VERIFY THAT THEY CAN READ INPUT DATA.  THE A PORTS ARE
                  288 ; THEN SET TO INPUT AND B TO OUTPUT AND THE TEST IS REPEATED
                  289 ; TO CHECK THE INPUT CAPABILITIES OF THE A PORTS.
                  290 ;   IF ANY PORT FAILS THE READ TEST, A BIT IS SET IN THE
                  291 ; ERROR CODE TO INDICATE THE FAILED DEVICE.  THE
                  292 ; ERROR MESSAGE IS NOT DISPLAYED UNTIL TEST 4 HAS BEEN
                  293 ; PERFORMED.
                  294 ;
 00CD CD0D07      295 TEST3:  CALL  CLRFLG    ;CLEAR 'REPEAT TEST' FLAG.
 00D0 CDFF06      296         CALL  CLRKBD    ;CLEAR KEYBOARD FIFO.
 00D3 010300      297         LXI   B,0003H   ;LOAD TEST AND ERROR CODE.
 00D6 3E01        298 RPT34:  MVI   A,01H     ;SET PORTS A AS OUTPUT,
 00D8 D320        299         OUT   CTRL1     ; OTHERS AS INPUT.
 00DA D328        300         OUT   CTRL2     ;
 00DC 3EFF        301         MVI   A,0FFH    ;
 00DE D302        302         OUT   DDR1A     ;
 00E0 D30A        303         OUT   DDR2A     ;
                  304
 00E2 3E01        305         MVI   A,01H     ;INITIALISE TEST BYTE.
 00E4 57          306 SHFTA:  MOV   D,A       ;SAVE THE BYTE IN D.
 00E5 D321        307         OUT   RAM1A     ;OUTPUT TO A PORTS.
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

00E7 D329        308        OUT   RAM2A     ;
00E9 D300        309        OUT   ROM1A     ;
00EB D308        310        OUT   ROM2A     ;
                 311
00ED DB22        312        IN    RAM1B     ;READ B PORT.
00EF AA          313        XRA   D         ;COMPARE WITH DATA WRITTEN.
00F0 C41407      314        CNZ   ERR1      ;SET ERROR FLAG IF NOT SAME.
00F3 DB23        315        IN    RAM1C     ;READ THE C PORT.
00F5 AA          316        XRA   D         ;COMPARE WITH DATA WRITTEN.
00F6 E63F        317        ANI   3FH       ;BOTTOM 6 BITS ONLY.
00F8 C41407      318        CNZ   ERR1      ;SET ERROR FLAG IF NOT SAME.
                 319
00FB DB2A        320        IN    RAM2B     ;REPEAT FOR RAM 2 PORTS.
00FD AA          321        XRA   D         ;
00FE C41907      322        CNZ   ERR2      ;
0101 DB2B        323        IN    RAM2C     ;
0103 AA          324        XRA   D         ;
0104 E63F        325        ANI   3FH       ;
0106 C41907      326        CNZ   ERR2      ;
                 327
0109 DB01        328        IN    ROM1B     ;ROM 1 PORTS.
010B AA          329        XRA   D         ;
010C C41E07      330        CNZ   ERR3      ;
                 331
010F DB09        332        IN    ROM2B     ;ROM 2 PORTS.
0111 AA          333        XRA   D         ;
0112 C42307      334        CNZ   ERR4      ;
                 335
0115 7A          336        MOV   A,D       ;RESTORE TEST BYTE.
0116 07          337        RLC             ;SHIFT TEST BIT LEFT.
0117 D2E400      338        JNC   SHFTA     ;REPEAT IF 8 BITS NOT DONE.
                 339 ;
                 340 ; NOW REPEAT WITH B PORTS AS OUTPUT; A AS INPUT.
                 341 ;
011A 3E00        342        MVI   A,00H     ;SET ROM A PORTS AS INPUT.
011C D302        343        OUT   DDR1A     ;
011E D30A        344        OUT   DDR2A     ;
0120 3EFF        345        MVI   A,0FFH    ;SET ROM B PORTS AS OUTPUT.
0122 D303        346        OUT   DDR1B     ;
0124 D30B        347        OUT   DDR2B     ;
0126 3E02        348        MVI   A,02H     ;SIMILARLY FOR THE RAM PORTS.
0128 D320        349        OUT   CTRL1     ;
012A D328        350        OUT   CTRL2     ;
                 351
012C 3E01        352        MVI   A,01H     ;RE-INITIALISE THE TEST BYTE.
012E 57          353 SHFTB: MOV   D,A       ;SAVE THE BYTE.
012F D322        354        OUT   RAM1B     ;WRITE OUT TO B PORTS.
0131 D32A        355        OUT   RAM2B     ;
0133 D301        356        OUT   ROM1B     ;
0135 D309        357        OUT   ROM2B     ;
                 358
0137 DB21        359        IN    RAM1A     ;
0139 AA          360        XRA   D         ;
013A C41407      361        CNZ   ERR1      ;
013D DB29        362        IN    RAM2A     ;
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

 013F AA          363        XRA   D         ;
 0140 C41907      364        CNZ   ERR2      ;
 0143 DB00        365        IN    ROM1A     ;
 0145 AA          366        XRA   D         ;
 0146 C41E07      367        CNZ   ERR3      ;
 0149 DB08        368        IN    ROM2A     ;
 014B AA          369        XRA   D         ;
 014C C42307      370        CNZ   ERR4      ;
                  371
 014F 7A          372        MOV   A,D       ;RETRIEVE THE TEST BYTE.
 0150 07          373        RLC             ;SHIFT THE TEST BIT LEFT.
 0151 D22E01      374        JNC   SHFTB     ;REPEAT UNTIL 8 BITS DONE.
                  375
                  376
                  377
                  378 ;
                  379 ;                TEST 4
                  380 ;                ------
                  381 ;
                  382 ;   THIS TEST CHECKS THE OPERATION OF THE TIMER ON
                  383 ; RAM CHIP 1 AND THE CPU TRAP INPUT (THE TIMER OUTPUT
                  384 ; IS PERMANENTLY CONNECTED TO THE 8085 TRAP INPUT ON
                  385 ; THE SDK-85 BOARD).
                  386 ;   THE TIMER IS PROGRAMMED TO GO LOW AFTER 128 CYCLES
                  387 ; FOR 128 CYCLES. A TRAP SHOULD OCCUR WHEN THE TIMER
                  388 ; OUTPUT GOES HIGH AGAIN, AND IF A TRAP IS NOT DETECT-
                  389 ; ED WITHIN 300D CYCLES AN ERROR FLAG IS SET. IF THERE
                  390 ; ARE ANY ERRORS FROM EITHER TEST 3 OR TEST 4 THE ERROR
                  391 ; CODE IS DISPLAYED AND BOTH TESTS ARE REPEATED AFTER
                  392 ; WAITING 1ms. THIS IS NECESSARY TO ENABLE TRACING OF THE
                  393 ; TIMER OUTPUT IF THE TRAP TEST FAILS. THE TESTS REPEAT
                  394 ; UNTIL 'NEXT ,' IS ENTERED AT THE KEYBOARD.
                  395 ;   IF THERE ARE NO ERRORS THE TEST 5 ROUTINE IS ENTERED
                  396 ; IMMEDIATELY.
                  397 ;
 0154 AF          398 TEST4:  XRA   A         ;CLEAR 'TRAP' FLAG.
 0155 320320      399         STA   TRPFLG    ;
 0158 3E01        400         MVI   A,01H     ;PROGRAM THE TIMER (128 CYCLES
 015A D325        401         OUT   TIM1HI    ; HIGH, 128 CYCLES LOW).
 015C 3E00        402         MVI   A,00H     ;
 015E D324        403         OUT   TIM1LO    ;
 0160 3EC0        404         MVI   A,0C0H    ;START THE TIMER (AND SET ALL
 0162 D320        405         OUT   CTRL1     ; PORTS TO INPUT).
 0164 3E14        406         MVI   A,20D     ;WAIT 300 CYCLES.
 0166 3D          407 WFTRP:  DCR   A         ;
 0167 C26601      408         JNZ   WFTRP     ;
 016A 3E80        409         MVI   A,80H     ;STOP THE TIMER AFTER
 016C D320        410         OUT   CTRL1     ; TERMINAL COUNT.
 016E 3A0320      411         LDA   TRPFLG    ;LOAD TRAP FLAG.
 0171 57          412         MOV   D,A       ;SAVE IT IN D.
 0172 3A0020      413         LDA   RPTFLG    ;SEE IF THE TEST IS BEING
 0175 B7          414         ORA   A         ; REPEATED.
 0176 C29101      415         JNZ   ITR34     ;IF SO, SKIP THE TEST FOR ERRORS.
                  416
 0179 7A          417         MOV   A,D       ;RETRIEVE THE TRAP FLAG.
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    9
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

 017A B7          418         ORA   A         ;TEST IT.
 017B C28301      419         JNZ   IOERT     ;SKIP IF FLAG SET.
 017E 0E04        420         MVI   C,04H     ;ELSE SET TEST NO. TO 4
 0180 C38801      421         JMP   IOERR     ; AND DISPLAY ERROR MSG.
 0183 78          422 IOERT:  MOV   A,B       ;TEST FOR ANY ERRORS FROM
 0184 B7          423         ORA   A         ; THE INPUT TEST (3).
 0185 CA9E01      424         JZ    TEST5     ;SKIP TO NEXT TEST IF NONE.
 0188 CD0707      425 IOERR:  CALL  SETFLG    ;SET 'REPEAT TEST' FLAG.
 018B CDFF06      426         CALL  CLRKBD    ;CLEAR THE KEYBOARD FIFO.
 018E CD9F06      427         CALL  ERRDSP    ;DISPLAY THE ERROR CODE.
 0191 3E01        428 ITR34:  MVI   A,01H     ;DELAY 1ms (FOR THE TIMER
 0193 CD4006      429         CALL  DELAY     ; TO RESET).
 0196 CDEF06      430         CALL  RDKBD     ;TEST FOR KEYBOARD ENTRY.
 0199 FE51        431         CPI   NEXT      ;'NEXT' ENTERED?
 019B C2D600      432         JNZ   RPT34     ;REPEAT TESTS IF NOT.
                  433
                  434
                  435
                  436
                  437 ;
                  438 ;                TEST 5
                  439 ;                ------
                  440 ;
                  441 ;   TEST 5 CHECKS THE OPERATION OF THE RST 6.5 INPUT OF
                  442 ; THE 8085.  THE I/O CONNECTORS MUST BE CHANGED, TO LOOP
                  443 ; THE ROM 1 A PORT BIT 0 (J3 PIN 31) BACK TO THE RST 6.5
                  444 ; INPUT OF THE BOARD (J1 PIN 20).  THIS CONNECTOR WILL ALSO
                  445 ; CONNECT BIT 1 OF THE PORT (J3/32) TO THE INTR INPUT (J1/18)
                  446 ; FOR USE IN TEST 7, AND BIT 2 OF THE PORT (J3/29) TO THE
                  447 ; HOLD INPUT (J1/14) FOR USE IN TEST 10.
                  448 ;   AFTER THE CONNECTOR HAS BEEN INSERTED THE THREE LINKS
                  449 ; SHORTING THE RST6.5, INT AND HOLD INPUTS TO GROUND MUST BE
                  450 ; REMOVED.  NOTE THAT THE PORT OUTPUTS ARE SET TO 0 BEFORE THE
                  451 ; CONNECTOR IS INSERTED SO THAT WHEN THE LINKS ARE REMOVED
                  452 ; THE THREE INPUTS ARE HELD LOW.  THIS IS PARTICULARLY
                  453 ; IMPORTANT FOR THE HOLD INPUT.
                  454 ;   THE TEST NUMBER IS FIRST DISPLAYED TO PROMPT THE CHANGE
                  455 ; OF CONNECTORS.  WHEN 'EXEC' IS ENTERED THE TEST PROCEEDS
                  456 ; WITH THE INTERRUPT BIT FIRST SET TO 0, THEN 1.  A RIM
                  457 ; INSTRUCTION IS EXECUTED IN EACH CASE TO VERIFY THAT THE
                  458 ; RST 6.5 INPUT PIN IS FUNCTIONING, AND THEN THE RST 6.5
                  459 ; MASK IS CLEARED TO ALLOW THE INTERRUPT TO OCCUR.
                  460 ;   IF ANY OF THESE TESTS FAIL AN ERROR MESSAGE IS DISP-
                  461 ; LAYED AND THE TEST IS REPEATED UNTIL 'NEXT' IS ENTERED.
                  462 ; IF NO ERRORS OCCUR TEST 6 COMMENCES IMMEDIATELY.
                  463 ;
 019E 3E00        464 TEST5:  MVI   A,00H     ;WRITE 0'S TO THE ROM A PORT.
 01A0 D300        465         OUT   ROM1A     ;
 01A2 3E07        466         MVI   A,07H     ;ENABLE THE BOTTOM THREE BITS
 01A4 D302        467         OUT   DDR1A     ; AS OUTPUTS.
 01A6 CDFF06      468         CALL  CLRKBD    ;CLEAR THE KEYBOARD BUFFER.
 01A9 3E05        469         MVI   A,05H     ;DISPLAY IMPENDING TEST NO.
 01AB CDBF06      470         CALL  DSPTST    ;
 01AE CD0D07      471         CALL  CLRFLG    ;CLEAR 'REPEAT TEST' FLAG.
 01B1 CDEF06      472 WAIT5:  CALL  RDKBD     ;WAIT FOR KBD. ENTRY.
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

 01B4 FE50        473        CPI   EXEC      ;'EXEC' ?
 01B6 C2B101      474        JNZ   WAIT5     ;WAIT IF NOT.
                  475
 01B9 010500      476        LXI   B,0005H   ;LOAD ERROR AND TEST NO.
 01BC 3E0D        477        MVI   A,0DH     ;UNMASK RST 6.5.
 01BE 30          478        SIM             ;
 01BF 20          479 RPT5:  RIM             ;TEST RST 6.5 INPUT.
 01C0 E620        480        ANI   20H       ;
 01C2 C41407      481        CNZ   ERR1      ;ERROR IF NOT 0.
 01C5 3E01        482        MVI   A,01H     ;SET THE INTERRUPT BIT.
 01C7 D300        483        OUT   ROM1A     ;
 01C9 20          484        RIM             ;TEST THE RST 6.5 INPUT.
 01CA E620        485        ANI   20H       ;
 01CC CC1907      486        CZ    ERR2      ;ERROR IF NOT SET.
 01CF AF          487        XRA   A         ;CLEAR 'RST' INT. FLAG.
 01D0 320220      488        STA   RSTFLG    ;
 01D3 FB          489        EI              ;ENABLE INTERRUPT TO OCCUR.
 01D4 00          490        NOP             ;
 01D5 00          491        NOP             ;
 01D6 F3          492        DI              ;DISABLE INTERRUPTS.
 01D7 3E00        493        MVI   A,00H     ;REMOVE THE INTERRUPT.
 01D9 D300        494        OUT   ROM1A     ;
 01DB 3A0220      495        LDA   RSTFLG    ;SEE IF INTERRUPT OCCURRED.
 01DE B7          496        ORA   A         ;
 01DF CC1E07      497        CZ    ERR3      ;SET ERROR FLAG IF NOT.
 01E2 3A0020      498        LDA   RPTFLG    ;SEE IF THE TEST IS BEING
 01E5 B7          499        ORA   A         ; REPEATED.
 01E6 C2F701      500        JNZ   ITR5      ;SKIP THE TEST FOR ERRORS IF SO.
 01E9 78          501        MOV   A,B       ;TEST THE ERROR FLAG.
 01EA B7          502        ORA   A         ;
 01EB CAFF01      503        JZ    TEST6     ;NEXT TEST IF NO ERRORS.
 01EE CD0707      504        CALL  SETFLG    ;SET THE 'REPEAT TEST' FLAG.
 01F1 CDFF06      505        CALL  CLRKBD    ;CLEAR THE KEYBOARD BUFFER.
 01F4 CD9F06      506        CALL  ERRDSP    ;DISPLAY ERROR CODE.
 01F7 CDEF06      507 ITR5:  CALL  RDKBD     ;TEST FOR 'NEXT'
 01FA FE51        508        CPI   NEXT      ; ENTERED.
 01FC C2BF01      509        JNZ   RPT5      ;REPEAT THE TEST IF NOT.
                  510 ;
                  511 ;              TEST 6
                  512 ;              ------
                  513 ;
                  514 ;   THIS TEST CHECKS OPERATION OF THE RST 7.5 CPU
                  515 ; INTERRUPT AND THE 'VECT INTR' KEY.  THE TEST NO.
                  516 ; IS DISPLAYED AND THE RST 7.5 MASK IS CLEARED. A
                  517 ; LOOP IS THEN ENTERED TO WAIT FOR EITHER A 'NEXT'
                  518 ; ENTRY AT THE KEYBOARD OR A RST 7.5 INTERRUPT
                  519 ; INPUT.
                  520 ;   IF THE RST 7.5 INPUT IS DETECTED, INTERRUPTS ARE
                  521 ; ENABLED AND A TEST IS MADE TO SEE IF THE RST 7.5
                  522 ; INTERUPT OCCURS.  IF NOT, OR IF THE KEYBOARD ENTRY
                  523 ; IS DETECTED BEFORE THE RST 7.5 INPUT (WHICH WOULD
                  524 ; HAPPEN IF THE 'VECT INTR' KEY HAD NO EFFECT AND THE
                  525 ; 'NEXT' KEY WERE PRESSED TO STOP THE TEST) AN ERROR
                  526 ; MESSAGE IS DISPLAYED UNTIL 'NEXT' IS ENTERED AGAIN
                  527 ; (THE TEST IS NOT REPEATED).
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                  528 ;   IF NO ERRORS ARE DETECTED THE TEST 7 ROUTINE IS
                  529 ; ENTERED IMMEDIATELY.
                  530 ;
01FF CDFF06       531 TEST6:  CALL  CLRKBD    ;CLEAR KEYBOARD BUFFER.
0202 010600       532         LXI   B,0006H   ;LOAD TEST AND ERROR NO.
0205 79           533         MOV   A,C       ;DISPLAY PROMPT FOR TEST 6.
0206 CDBF06       534         CALL  DSPTST    ;
0209 3E0B         535         MVI   A,0BH     ;UNMASK RST 7.5.
020B 30           536         SIM             ;
020C AF           537         XRA   A         ;CLEAR 'RST' FLAG.
020D 320220       538         STA   RSTFLG    ;
0210 CDEF06       539 WAIT6:  CALL  RDKBD     ;TEST FOR 'NEXT' ENTRY.
0213 FE51         540         CPI   NEXT      ;
0215 CA2B02       541         JZ    EXIT6     ;EXIT WITH ERROR IF SO.
0218 20           542         RIM             ;TEST FOR RST 7.5.
0219 E640         543         ANI   40H       ;
021B CA1002       544         JZ    WAIT6     ;CONTINUE WAITING IF
                  545                         ; NOTHING HAS HAPPENED.
021E FB           546         EI              ;ALLOW INTERRUPT TO OCCUR.
021F 00           547         NOP             ;
0220 00           548         NOP             ;
0221 F3           549         DI              ;DISABLE INTERRUPTS.
0222 3A0220       550         LDA   RSTFLG    ;SEE IF INTERRUPT OCCURED.
0225 B7           551         ORA   A         ;
0226 C23A02       552         JNZ   TEST7     ;PROCEED TO TEST 7 IF SO.
0229 0601         553         MVI   B,01H     ;ELSE SET ERROR NO. TO 2.
022B 04           554 EXIT6:  INR   B         ;UPDATE ERROR NO.
022C CDFF06       555         CALL  CLRKBD    ;CLEAR KEYBOARD BUFFER.
022F CD9F06       556         CALL  ERRDSP    ;DISPLAY ERROR CODE.
0232 CDEF06       557 WT6:    CALL  RDKBD     ;WAIT FOR 'NEXT'.
0235 FE51         558         CPI   NEXT      ;
0237 C23202       559         JNZ   WT6       ;
                  560 ;
                  561 ;              TEST 7
                  562 ;              ------
                  563 ;
                  564 ;   TEST 7 IS AN OPTIONAL TEST TO EXERCISE THE 'INT'
                  565 ;  FACILITIES OF THE BOARD.  THE TEST REQUIRES AN
                  566 ; EXTERNAL BUFFER CONNECTED TO THE SDK-85 DATA BUS
                  567 ; BUFFERS TO SUPPLY THE 'RST 3' VECTOR IN RESPONSE
                  568 ; TO INTA/.
                  569 ;   THE TEST NUMBER IS FIRST DISPLAYED AND THE KEY-
                  570 ; BOARD IS MONITORED FOR INPUT.  'EXEC' WILL CAUSE
                  571 ; THE TEST TO PROCEED, WHEREAS 'NEXT' WILL ABORT
                  572 ; THIS TEST AND THE NEXT TEST ROUTINE WILL BE ENTER-
                  573 ; ED IMMEDIATELY.
                  574 ;   THE TEST SIMPLY ASSERTS THE 'INT' INPUT TO THE
                  575 ; BOARD VIA BIT 1 OF ROM 1 PORT A (REFER TO NOTE
                  576 ; ON TEST 5).  HAVING ASSERTED 'INT', INTERRUPTS
                  577 ; ARE ENABLED AND A TEST IS MADE TO DETERMINE
                  578 ; WHETHER AN RST 3 OCCURS.  IF SO, THE TEST 8
                  579 ; ROUTINE IS ENTERED.  IF NOT, AN ERROR MESSAGE
                  580 ; IS DISPLAYED AND THE TEST IS REPEATED UNTIL THE
                  581 ; 'NEXT' KEY IS PRESSED.
                  582 ;
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   12
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE        SOURCE STATEMENT

 023A CD0D07      583 TEST7:  CALL  CLRFLG    ;CLEAR 'REPEAT TEST' FLAG.
 023D CDFF06      584         CALL  CLRKBD    ;CLEAR KEYBOARD BUFFER.
 0240 3E07        585         MVI   A,07H     ;DISPLAY TEST NUMBER.
 0242 CDBF06      586         CALL  DSPTST    ;
 0245 CDEF06      587 WAIT7:  CALL  RDKBD     ;WAIT FOR INPUT.
 0248 FE51        588         CPI   NEXT      ;'NEXT' ?
 024A CA8A02      589         JZ    TEST8     ;SKIP TO NEXT TEST IF SO.
 024D FE50        590         CPI   EXEC      ;'EXEC' ?
 024F C24502      591         JNZ   WAIT7     ;GO BACK AND WAIT IF NOT.
                  592
 0252 CDFF06      593         CALL  CLRKBD    ;CLEAR KEYBOARD AGAIN.
 0255 3E0F        594         MVI   A,0FH     ;MASK ALL RST INTERUPTS.
 0257 30          595         SIM             ;
 0258 AF          596 RPT7:   XRA   A         ;CLEAR 'INT' FLAG.
 0259 320120      597         STA   INTFLG    ;
 025C 3E02        598         MVI   A,02H     ;ASSERT 'INT'.
 025E D300        599         OUT   ROM1A     ;
 0260 FB          600         EI              ;ALLOW INTERRUPT.
 0261 00          601         NOP             ;
 0262 00          602         NOP             ;
 0263 F3          603         DI              ;DISABLE INTERRUPTS.
 0264 3E00        604         MVI   A,00H     ;REMOVE THE INTERRUPT.
 0266 D300        605         OUT   ROM1A     ;
 0268 3A0020      606         LDA   RPTFLG    ;SEE IF REPEAT TEST.
 026B B7          607         ORA   A         ;
 026C C28202      608         JNZ   ITR7      ;SKIP TEST FOR ERROR IF SO.
 026F 3A0120      609         LDA   INTFLG    ;SEE IF INTERRUPT OCCURRED.
 0272 B7          610         ORA   A         ;
 0273 C28A02      611         JNZ   TEST8     ;SKIP TO TEST 8 IF SO.
 0276 CD0707      612         CALL  SETFLG    ;SET 'REPEAT TEST' FLAG.
 0279 CDFF06      613         CALL  CLRKBD    ;CLEAR KEYBOARD BUFFER.
 027C 010701      614         LXI   B,0107H   ;LOAD THE ERROR CODE...
 027F CD9F06      615         CALL  ERRDSP    ;...AND DISPLAY IT.
 0282 CDEF06      616 ITR7:   CALL  RDKBD     ;TEST FOR 'NEXT'.
 0285 FE51        617         CPI   NEXT      ;
 0287 C25802      618         JNZ   RPT7      ;REPEAT TEST IF NOT.
                  619 ;
                  620 ;              TEST 8
                  621 ;              ------
                  622 ;
                  623 ;   TEST 8 IS A TEST FOR THE TIMER IN THE SECOND 8155
                  624 ; (IF IT IS PRESENT).  THE TEST NUMBER IS DISPLAYED,
                  625 ; AND THE KEYBOARD IS MONITORED FOR INPUT. 'NEXT'
                  626 ; WILL ABORT THE TEST AND 'EXEC' WILL CAUSE IT TO
                  627 ; PROCEED.
                  628 ;   THE TEST REQIRES A JUMPER PLUG TO BE INSERTED INTO
                  629 ; J5 TO LOOP THE TIMER OUTPUT OF THE 8155 (PIN 23) BACK
                  630 ; TO THE MSB OF ITS A PORT (PIN 23).
                  631 ;   THE TIMER IS PROGRAMMED TO GO LOW FOR 128 CYCLES
                  632 ; AFTER 128 CYCLES.  THE OUTPUT IS THEN MONITORED TO
                  633 ; ENSURE THAT IT FOLLOWS THE SEQUENCE 1->0->1 WITHIN
                  634 ; THE SPECIFIED TIME.  IF ANY ERROR IS DETECTED AN
                  635 ; ERROR MESSAGE IS DISPLAYED AND THE TEST IS REPEATED
                  636 ; AT 1ms. INTERVALS UNTIL THE 'NEXT' KEY IS PRESSED.
                  637 ;
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   13
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE        SOURCE STATEMENT

 028A CDFF06      638 TEST8:  CALL  CLRKBD    ;CLEAR KEYBOARD BUFFER.
 028D 3E08        639         MVI   A,08H     ;DISPLAY TEST NO.
 028F CDBF06      640         CALL  DSPTST    ;
 0292 CDEF06      641 WAIT8:  CALL  RDKBD     ;WAIT FOR INPUT.
 0295 FE51        642         CPI   NEXT      ;'NEXT' => NEXT TEST.
 0297 CAF402      643         JZ    TEST9     ;
 029A FE50        644         CPI   EXEC      ;'EXEC' => PROCEED.
 029C C29202      645         JNZ   WAIT8     ;OTHERWISE WAIT.
                  646
 029F CDFF06      647         CALL  CLRKBD    ;CLEAR KEYBOARD AGAIN.
 02A2 010800      648         LXI   B,0008H   ;LOAD TEST AND ERROR NO.
 02A5 DB29        649         IN    RAM2A     ;READ TIMER OUTPUT STATE.
 02A7 B7          650         ORA   A         ;TEST MSB (TIMER OUT/).
 02A8 F2D402      651         JP    ER81      ;ERROR IF NOT = 1.
 02AB 3E00        652         MVI   A,00H     ;PROGRAM TIMER (HIGH 128
 02AD D32C        653         OUT   TIM2LO    ; CYCLES, LOW 128 CYCLES).
 02AF 3E01        654         MVI   A,01H     ;
 02B1 D32D        655         OUT   TIM2HI    ;
 02B3 3EC0        656         MVI   A,0C0H    ;START THE TIMER.
 02B5 D328        657         OUT   CTRL2     ;
                  658
 02B7 1607        659         MVI   D,07H     ;TIMEOUT COUNTER.
 02B9 15          660 WTF0:   DCR   D         ;TEST THE COUNTER.
 02BA CAD302      661         JZ    ER82      ;ERROR IF COUNT IS UP.
 02BD DB29        662         IN    RAM2A     ;TEST TIMER OUTPUT.
 02BF B7          663         ORA   A         ;
 02C0 FAB902      664         JM    WTF0      ;WAIT IF NOT 0 YET.
                  665
 02C3 1607        666         MVI   D,07H     ;TIMEOUT COUNTER.
 02C5 15          667 WTF1:   DCR   D         ;TEST THE COUNTER.
 02C6 CAD202      668         JZ    ER83      ;ERROR IF COUNT IS UP.
 02C9 DB29        669         IN    RAM2A     ;READ TIMER OUT/.
 02CB B7          670         ORA   A         ;
 02CC F2C502      671         JP    WTF1      ;WAIT UNTIL 1 AGAIN.
 02CF C3F402      672         JMP   TEST9     ;NO ERRORS-GO TO TEST 9.
 02D2 04          673 ER83:   INR   B         ;B:=3.
 02D3 04          674 ER82:   INR   B         ;B:=2.
 02D4 04          675 ER81:   INR   B         ;B:=1.
 02D5 CDFF06      676         CALL  CLRKBD    ;CLEAR THE KEYBOARD BUFFER.
 02D8 CD9F06      677         CALL  ERRDSP    ;DISPLAY ERROR CODE.
 02DB 3E01        678 RPT8:   MVI   A,01H     ;WAIT FOR 1MS AND RESTART THE TIMER.
 02DD CD4006      679         CALL  DELAY     ;
 02E0 3E00        680         MVI   A,00H     ;
 02E2 D32C        681         OUT   TIM2LO    ;
 02E4 3E01        682         MVI   A,01H     ;
 02E6 D32D        683         OUT   TIM2HI    ;
 02E8 3EC0        684         MVI   A,0C0H    ;START THE TIMER.
 02EA D328        685         OUT   CTRL2     ;
 02EC CDEF06      686         CALL  RDKBD     ;REPEAT THE TEST UNTIL
 02EF FE51        687         CPI   NEXT      ; 'NEXT' IS ENTERED.
 02F1 C2DB02      688         JNZ   RPT8      ;
                  689 ;
                  690 ;              TEST 9
                  691 ;              ------
                  692 ;
```

```
 LOC  OBJ          LINE       SOURCE STATEMENT

                    693 ;   THIS TEST IS INTENDED TO EXERCISE EXTERNAL MEMORY
                    694 ; (IF ANY) AND THE DATA BUS BUFFERS.  THE TEST NUMBER
                    695 ; IS DISPLAYED AND THE TEST STARTS IMMEDIATELY.
                    696 ;   THE 8205 CS7/ OUTPUT IS PULSED TO SIGNAL THE
                    697 ; START OF THE TEST (FOR USE WITH A SIGNATURE ANALYSER
                    698 ; - IF REQUIRED) AND THE ENTIRE EXTERNAL MEMORY SPACE
                    699 ; (8000H TO FFFFH) IS WRITTEN WITH 00H AND THEN READ.
                    700 ; THE PROCESS IS THEN REPEATED, WRITING FFH.  FINALLY
                    701 ; CS6/ IS PULSED TO SIGNAL THE END OF THE TEST SEQU-
                    702 ; ENCE.
                    703 ;   THE TEST IS REPEATED UNTIL THE 'NEXT' KEY IS
                    704 ; PRESSED.
                    705 ;
 02F4 CDFF06        706 TEST9:  CALL  CLRKBD    ;CLEAR THE KEYBOARD BUFFER.
 02F7 3E09          707         MVI   A,09H     ;DISPLAY THE TEST NO.
 02F9 CDBF06        708         CALL  DSPTST    ;
 02FC 0600          709         MVI   B,00H     ;DATA TO BE WRITTEN.
 02FE DB38          710 RPT9:   IN    CS7       ;PULSE CS7/ WITH RD/ AND
 0300 D338          711         OUT   CS7       ; WR/ TO FLAG START OF TEST.
 0302 210080        712 WRFF:   LXI   H,8000H   ;START OF EXTERNAL MEMORY.
 0305 70            713 NXTAD:  MOV   M,B       ;WRITE TO THE ADDRESS.
 0306 7E            714         MOV   A,M       ;READ FROM THE ADDRESS.
 0307 23            715         INX   H         ;POINT TO NEXT LOCATION.
 0308 7C            716         MOV   A,H       ;TEST FOR UPPER LIMIT (0000H).
 0309 B5            717         ORA   L         ;
 030A C20503        718         JNZ   NXTAD     ;
                    719
 030D 78            720         MOV   A,B       ;COMPLEMENT THE TEST BYTE.
 030E 2F            721         CMA             ;
 030F 47            722         MOV   B,A       ;
 0310 B7            723         ORA   A         ;TEST FOR 0 (SECOND TIME
                    724                         ; THROUGH THE TEST).
 0311 FA0203        725         JM    WRFF      ;NOT 0-REPEAT WITH FFH.
 0314 DB30          726         IN    CS6       ;00-SIGNAL END OF TEST
 0316 D330          727         OUT   CS6       ;
 0318 CDEF06        728         CALL  RDKBD     ;TEST FOR KEYBOARD ENTRY.
 031B FE51          729         CPI   NEXT      ;'NEXT' ?
 031D C2FE02        730         JNZ   RPT9      ;REPEAT THE TEST IF NOT.
                    731 ;
                    732 ;               TEST 10
                    733 ;               -------
                    734 ;
                    735 ;   THE FINAL TEST EXERCISES THE HOLD INPUT OF THE CPU.
                    736 ; THE 'FINAL TEST' MESSAGE IS DISPLAYED AND THE HOLD
                    737 ; INPUT IS ASSERTED.  IF THERE IS NO FAULT THIS WILL
                    738 ; CAUSE THE SYSTEM TO HANG.  IF THE HOLD IS NOT SUCCESS-
                    739 ; FUL, HOWEVER, EXECUTION CONTINUES AND AN ERROR MESSAGE
                    740 ; IS WRITTEN TO THE DISPLAY.  THE CPU THEN HALTS AND THE
                    741 ; HOLD AND HLDA LINES MAY BE TRACED TO FIND THE SOURCE
                    742 ; OF THE FAULT.
                    743 ;   THIS TEST REQUIRES BIT 2 OF PORT A ON A15 (J3/29) TO
                    744 ; BE CONNECTED TO THE SDK85 HOLD INPUT (J1/14).  REFER TO
                    745 ; THE NOTE ON TEST 5.
                    746 ;
 0320 3E16          747 LAST:   MVI   A,16H     ;DISPLAY '.... .. FOR LAST TEST.
```

```
 LOC  OBJ        LINE        SOURCE STATEMENT

 0322 CDBF06      748         CALL  DSPTST    ;
 0325 3E04        749         MVI   A,04H     ;ASSERT HOLD.
 0327 D300        750         OUT   ROM1A     ;
 0329 00          751         NOP             ;
 032A 00          752         NOP             ;THE CPU SHOULD BE HUNG BY NOW.
 032B 011010      753         LXI   B,1010H   ;DISPLAY ERR MESSAGE IF NOT.
 032E CD9F06      754         CALL  ERRDSP    ;
 0331 76          755         HLT             ;STOP.
                  756
                  757 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   16
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

                  758 ;*********************************************************************************
                  759 ;
                  760 ;
                  761 ;              SUBROUTINES FOR STAGE III
                  762 ;              -------------------------
                  763 ;
                  764 ;
                  765 ;   THE SUBROUTINES FOR STAGE III ARE STORED IN THE TOP 448 BYTES
                  766 ; OF THE ROM BECAUSE THERE IS NOT ENOUGH ROOM LEFT IN THE BOTTOM 1K,
                  767 ; WHERE THE MAIN PROGRAM IS STORED.  THE SIGNATURE ANALYSIS DIAGNOSTIC
                  768 ; PROGRAM (STAGE II) FITS IN THE 512 BYTES STARTING AT 0400H.
                  769 ;   LOCATIONS 600H TO 63FH ARE RESERVED FOR THE USE OF THE 8085 SELF
                  770 ; TEST PROGRAM, WHICH MAY BE STORED IN EXTERNAL ROM (8000-83FFH) AND
                  771 ; EXECUTED BEFORE STAGE II (REFER TO THE LISTING OF SLFTST.V8).
                  772 ;
                  773 ;
0640              774         ORG   0640H
                  775 ;
                  776 ;   1 MILLISECOND DELAY ROUTINE.
                  777 ;
                  778 ; THIS ROUTINE GENERATES A DELAY OF (A) MILLISECONDS.
                  779 ;
0640 D5           780 DELAY:  PUSH  D          ;
0641 16D6         781 LOOP2:  MVI   D,214D     ;14*214=2996 CYCLES.
0643 15           782 LOOP1:  DCR   D          ;
0644 C24306       783         JNZ   LOOP1      ;
0647 3D           784         DCR   A          ;
0648 C24106       785         JNZ   LOOP2      ;
064B D1           786         POP   D          ;
064C C9           787         RET              ;
                  788 ;
                  789 ;
                  790 ;   HALF SECOND DELAY ROUTINE.
                  791 ;
                  792 ; THIS ROUTINE GENERATES A DELAY OF 500ms USING THE 1ms
                  793 ; DELAY ROUTINE.
                  794 ;
064D 3EFA         795 DLY500: MVI   A,250D     ;
064F CD4006       796         CALL  DELAY      ;
0652 3EFA         797         MVI   A,250D     ;
0654 CD4006       798         CALL  DELAY      ;
0657 C9           799         RET              ;
                  800 ;
                  801 ;
                  802 ;   KEYBOARD TEST ROUTINE.
                  803 ;
                  804 ; THIS ROUTINE PERFORMS THE INITIAL TEST ON THE OPERATION
                  805 ; OF THE KEYBOARD CONTROLLER.  BOTH THE NO. OF CHARACTERS
                  806 ; IN THE FIFO AND THE RST 5.5 INPUT ARE TESTED TO DETERMINE
                  807 ; WHETHER ANY KEYBOARD ENTRIES HAVE BEEN MADE.  IF BOTH
                  808 ; TESTS ARE NEGATIVE THE ROUTINE RETURNS WITH (A) = 00.
                  809 ; IF ONE TEST SUCCEEDS AND THE OTHER FAILS THE ROUTINE RET-
                  810 ; URNS WITH AN ERROR FLAG SET (A<0).  IF BOTH TESTS SUC-
                  811 ; CEED THE RST 5.5 INTERRUPT IS ALLOWED TO OCCUR AND A
                  812 ; CHECK IS MADE TO DETERMINE WHETHER IT WAS SUCCESSFUL,
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   17
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

                  813 ; AN ERROR FLAG BEING SET IF NOT.  FINALLY, THE CHARACTER
                  814 ; IS READ FROM THE FIFO AND ANOTHER ERROR IS FLAGGED IF
                  815 ; ANY MORE CHARACTERS ARE LEFT IN THE FIFO.
                  816 ;
 0658 E5          817 TSTKBD: PUSH  H          ;
 0659 210019      818         LXI   H,KDCC     ;8279 CONTROL REG. ADRS.
 065C 7E          819         MOV   A,M        ;READ FIFO STATUS INTO A.
 065D E607        820         ANI   07H        ;MASK NO. OF CHARS. IN FIFO.
 065F 47          821         MOV   B,A        ;SAVE IN B.
 0660 20          822         RIM              ;TEST RST 5.5 STATUS.
 0661 E610        823         ANI   10H        ;
 0663 80          824         ADD   B          ;ADD IN NO. OF FIFO CHARS.
 0664 CA8D06      825         JZ    RETURN     ;RETURN IF NO INT., NO CHARS.
 0667 FE10        826         CPI   10H        ;INT. WITH NO CHARS.?
 0669 CA8F06      827         JZ    KBDE1      ;SET ERROR FLAG IF SO.
 066C FA9306      828         JM    KBDE2      ;CHARS. AND NO INT.?
                  829
 066F AF          830         XRA   A          ;INT. AND CHARS.
 0670 320220      831         STA   RSTFLG     ;CLEAR 'RST' INT. FLAG.
 0673 3E0E        832         MVI   A,0EH      ;UNMASK RST 5.5.
 0675 30          833         SIM              ;
 0676 FB          834         EI               ;ALLOW INT. TO OCCUR.
 0677 00          835         NOP              ;
 0678 00          836         NOP              ;
 0679 F3          837         DI               ;DISABLE INTERRUPTS.
 067A 3A0220      838         LDA   RSTFLG     ;SEE IF INT. OCCURRED.
 067D B7          839         ORA   A          ;
 067E CA9706      840         JZ    KBDE3      ;ERROR IF NOT.
 0681 25          841         DCR   H          ;(HL)= KDCD.
 0682 46          842         MOV   B,M        ;READ CHAR. INTO B.
 0683 24          843         INR   H          ;(HL) = KDCC.
 0684 7E          844         MOV   A,M        ;READ STATUS AGAIN.
 0685 E607        845         ANI   07H        ;MASK NO. OF CHARS.
 0687 C29B06      846         JNZ   KBDE4      ;ERROR IF NOT 0.
 068A 78          847         MOV   A,B        ;PUT CHAR. INTO A...
 068B F640        848         ORI   40H        ;SET FLAG FOR CHAR. PRESENT.
 068D E1          849 RETURN: POP   H          ;...AND RETURN.
 068E C9          850         RET              ;
 068F 3EFF        851 KBDE1:  MVI   A,-1       ;SET ERROR CODE=1
 0691 E1          852         POP   H          ;
 0692 C9          853         RET              ;
 0693 3EFE        854 KBDE2:  MVI   A,-2       ;SET ERROR CODE=2.
 0695 E1          855         POP   H          ;
 0696 C9          856         RET              ;
 0697 3EFD        857 KBDE3:  MVI   A,-3       ;SET ERROR CODE=3
 0699 E1          858         POP   H          ;
 069A C9          859         RET              ;
 069B 3EFC        860 KBDE4:  MVI   A,-4       ;SET ERROR CODE=4
 069D E1          861         POP   H          ;
 069E C9          862         RET              ;
                  863 ;
                  864 ;
                  865 ;   ERROR DISPLAY ROUTINE
                  866 ;
                  867 ; THIS ROUTINE DISPLAYS THE 2 DIGIT ERROR CODE PASSED
```

```
 LOC  OBJ          LINE       SOURCE STATEMENT

                    868 ; IN BC.  THE MSD (IN C) IS THE NUMBER OF THE TEST IN
                    869 ; WHICH THE ERROR WAS DETECTED AND THE LSD (IN B) IS
                    870 ; THE ERROR CODE.
                    871 ;
 069F F5            872 ERRDSP: PUSH  PSW       ;
 06A0 E5            873         PUSH  H         ;
 06A1 210019        874         LXI   H,KDCC    ;8279 CONTROL REG.
 06A4 CDD806        875         CALL  CLRDSP    ;CLEAR THE DISPLAY.
 06A7 3690          876         MVI   M,90H     ;WRITE DISP. RAM (AUTO-INC.).
 06A9 25            877         DCR   H         ;(HL) = KDCD.
 06AA 3668          878         MVI   M,68H     ;WRITE 'E' TO LOC. 0.
 06AC 36FA          879         MVI   M,0FAH    ;  '   'r'
 06AE 36FA          880         MVI   M,0FAH    ;  '   'r'
 06B0 36FF          881         MVI   M,0FFH    ;  '   BLANK.
 06B2 79            882         MOV   A,C       ;CONVERT NUMBER TO DISPLAY
 06B3 CDE206        883         CALL  CONVRT    ; CODE.
 06B6 77            884         MOV   M,A       ;WRITE DISPLAY CODE.
 06B7 78            885         MOV   A,B       ;REPEAT WITH SECOND DIGIT.
 06B8 CDE206        886         CALL  CONVRT    ;
 06BB 77            887         MOV   M,A       ;
 06BC E1            888         POP   H         ;RETURN.
 06BD F1            889         POP   PSW       ;
 06BE C9            890         RET             ;
                    891 ;
                    892 ;
                    893 ;   TEST NUMBER DISPLAY ROUTINE.
                    894 ;
                    895 ; THIS ROUTINE DISPLAYS THE NUMBER PASSED IN A TO
                    896 ; INDICATE THE TEST ABOUT TO BE PERFORMED.  THE
                    897 ; DISPLAY WILL SHOW:
                    898 ;                   .... .N
                    899 ;                         WHERE N IS THE NUMBER
                    900 ; OF THE TEST.
                    901 ;
 06BF D5            902 DSPTST: PUSH  D         ;
 06C0 E5            903         PUSH  H         ;
 06C1 210019        904         LXI   H,KDCC    ;8279 CONTROL REG.
 06C4 CDD806        905         CALL  CLRDSP    ;CLEAR THE DISPLAY.
 06C7 3690          906         MVI   M,90H     ;WRITE DISP. RAM (AUTO-INC.).
 06C9 25            907         DCR   H         ;(HL) = KDCD.
 06CA CDE206        908         CALL  CONVRT    ;CONVERT NO. TO DISP. CODE.
 06CD 16F7          909         MVI   D,0F7H    ;CODE FOR '.'
 06CF 72            910         MOV   M,D       ;WRITE '.' IN FIRST 5
 06D0 72            911         MOV   M,D       ; LOCATIONS.
 06D1 72            912         MOV   M,D       ;
 06D2 72            913         MOV   M,D       ;
 06D3 72            914         MOV   M,D       ;
 06D4 77            915         MOV   M,A       ;WRITE THE NO. IN THE LAST
                    916                         ; LOCATION.
 06D5 E1            917         POP   H         ;
 06D6 D1            918         POP   D         ;
 06D7 C9            919         RET             ;
                    920 ;
                    921 ;
                    922 ;   CLEAR DISPLAY ROUTINE
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

                  923 ;
                  924 ; THIS ROUTINE WRITES A 'CLEAR DISPLAY' COMMAND TO THE 8279
                  925 ; CONTROL REGISTER AND WAITS 1ms FOR THE CLEAR OPERATION TO
                  926 ; COMPLETE (IT SHOULD TAKE 160us).  ON ENTRY HL MUST CONTAIN
                  927 ; THE ADDRESS OF THE 8279 CONTROL REGISTER (1900H).  THIS
                  928 ; VALUE IS NOT MODIFIED.
                  929 ;
 06D8 F5          930 CLRDSP: PUSH  PSW       ;
 06D9 36DC        931         MVI   M,0DCH    ;WRITE 'CLEAR DISPLAY' CMD TO KDCC.
 06DB 3E01        932         MVI   A,01H     ;WAIT FOR 1ms.
 06DD CD4006      933         CALL  DELAY     ;
 06E0 F1          934         POP   PSW       ;
 06E1 C9          935         RET             ;
                  936 ;
                  937 ;
                  938 ;   DISPLAY CODE CONVERSION ROUTINE.
                  939 ;
                  940 ; THIS ROUTINE CONVERTS THE NUMBER IN A INTO THE
                  941 ; CODE TO BE WRITTEN TO THE DISPLAY CONTROLLER TO
                  942 ; DISPLAY THAT NUMBER.  IF (A)>15, THE CODE RET-
                  943 ; URNED WILL BE THAT OF THE CORRESPONDING CHARACTER
                  944 ; IN THE TABLE (CHRTAB).  FOR EXAMPLE, IF (A) = 17D
                  945 ; THE CODE RETURNED WILL BE 9BH, WHICH IS THE DISP-
                  946 ; LAY CODE FOR 'H', THE 17TH CHARACTER IN THE TABLE.
                  947 ; IF (A)>24D THE RESULT WILL BE UNPREDICTABLE.
                  948 ;
 06E2 E5          949 CONVRT: PUSH  H         ;
 06E3 C5          950         PUSH  B         ;
 06E4 212807      951         LXI   H,CHRTAB  ;ADRS. OF CHAR. CODE TABLE.
 06E7 0600        952         MVI   B,00H     ;PUT THE NO. TO BE DISP-
 06E9 4F          953         MOV   C,A       ; LAYED INTO BC.
 06EA 09          954         DAD   B         ;ADD OFFSET TO BASE ADRS.
 06EB 7E          955         MOV   A,M       ;FETCH THE CODE.
 06EC C1          956         POP   B         ;RETURN.
 06ED E1          957         POP   H         ;
 06EE C9          958         RET             ;
                  959 ;
                  960 ;
                  961 ;   READ KEYBOARD ROUTINE
                  962 ;
                  963 ; THIS ROUTINE READS THE FIFO STATUS TO SEE IF ANY CHARACTERS
                  964 ; ARE IN THE FIFO, RETURNING WITH (A) = 0 IF NOT.  IF A CHAR-
                  965 ; ACTER IS AVAILABLE, IT IS PLACED IN A AND BIT 6 IS SET TO
                  966 ; SHOE THAT A CHARACTER WAS READ. THE RST 5.5 INTERRUPT IS
                  967 ; NOT USED.
                  968 ;
 06EF E5          969 RDKBD:  PUSH  H         ;
 06F0 210019      970         LXI   H,KDCC    ;8279 CONTROL REG.
 06F3 7E          971         MOV   A,M       ;READ FIFO STATUS.
 06F4 E60F        972         ANI   0FH       ;MASK NO. OF CHARACTERS (FIFO
                  973                         ; FULL => 8 CHARS).
 06F6 CAFD06      974         JZ    RTRN      ;RETURN WITH A=0 IF NONE.
 06F9 25          975         DCR   H         ;(HL) = KDCD.
 06FA 7E          976         MOV   A,M       ;READ A CHARACTER.
 06FB F640        977         ORI   40H       ;SET FLAG FOR CHAR. PRESENT.
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

06FD E1          978 RTRN:  POP   H          ;RETURN.
06FE C9          979        RET             ;
                 980 ;
                 981 ;
                 982 ;   KEYBOARD BUFFER CLEAR ROUTINE
                 983 ;
                 984 ; THIS ROUTINE WRITES A 'CLEAR FIFO' COMMAND TO THE
                 985 ; 8279 CONTROL REG. TO DISCARD ALL PREVIOUS KEYBOARD
                 986 ; ENTRIES.
                 987 ;
06FF E5          988 CLRKBD: PUSH  H          ;
0700 210019      989        LXI   H,KDCC    ;8279 CONTROL REG.
0703 36C2        990        MVI   M,0C2H    ;CLEAR FIFO COMMAND.
0705 E1          991        POP   H          ;RETURN.
0706 C9          992        RET             ;
                 993 ;
                 994 ;
                 995 ;   'REPEAT FLAG' SET AND CLEAR ROUTINES.
                 996 ;
                 997 ; THESE ROUTINES SET AND RESET THE FLAG RPTFLG WHICH
                 998 ; WHEN SET, ININDICATES THAT THE CURRENT TEST IS BEING
                 999 ; REPEATED, FOLLOWING DETECTION OF AN ERROR IN THE TEST.
                1000 ;
0707 F5         1001 SETFLG: PUSH  PSW        ;
0708 3E01       1002        MVI   A,01H     ;
070A C30F07     1003        JMP   STORE     ;JUMP TP STORE 1 IN RPTFLG
                1004
070D F5         1005 CLRFLG: PUSH  PSW        ;
070E AF         1006        XRA   A         ;CLEAR A.
070F 320020     1007 STORE:  STA   RPTFLG    ;STORE IN RPTFLG
0712 F1         1008        POP   PSW       ;RETURN.
0713 C9         1009        RET             ;
                1010 ;
                1011 ;
                1012 ;   SET ERROR FLAG ROUTINES
                1013 ;
                1014 ; THE FOLLOWING ROUTINES (ERR1 - ERR4) SET A BIT IN
                1015 ; THE B REGISTER TO INDICATE THAT AN ERROR HAS BEEN
                1016 ; DETECTED.
                1017 ;
0714 3E01       1018 ERR1:   MVI   A,01H     ;SET BIT 0.
0716 C32507     1019        JMP   SETBIT    ;
0719 3E02       1020 ERR2:   MVI   A,02H     ;SET BIT 1.
071B C32507     1021        JMP   SETBIT    ;
071E 3E04       1022 ERR3:   MVI   A,04H     ;SET BIT 2.
0720 C32507     1023        JMP   SETBIT    ;
0723 3E08       1024 ERR4:   MVI   A,08H     ;SET BIT 3.
0725 B0         1025 SETBIT: ORA   B         ;SET THE BIT IN B.
0726 47         1026        MOV   B,A       ;
0727 C9         1027        RET             ;RETURN.
                1028
                1029
                1030
                1031
                1032 ;
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   21
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE        SOURCE STATEMENT

                 1033 ;          DISPLAY CHARACTER TABLE
                 1034 ;          ----------------------
                 1035 ;
 0728 0C         1036 CHRTAB: DB    0CH,9FH,4AH,0BH     ;0,1,2,3
 0729 9F
 072A 4A
 072B 0B
 072C 99         1037         DB    99H,29H,28H,8FH     ;4,5,6,7
 072D 29
 072E 28
 072F 8F
 0730 08         1038         DB    08H,09H,88H,38H     ;8,9,A,b
 0731 09
 0732 88
 0733 38
 0734 6C         1039         DB    6CH,1AH,68H,0E8H    ;C,d,E,F
 0735 1A
 0736 68
 0737 E8
 0738 98         1040         DB    98H,7CH,0C8H,1CH    ;H,L,P,U
 0739 7C
 073A C8
 073B 1C
 073C FA         1041         DB    0FAH,78H,0F7H,0FFH  ;r,t,.,Space
 073D 78
 073E F7
 073F FF
                 1042
                 1043
                 1044
                 1045
                 1046
                 1047 ;
                 1048 ;              RAM LOCATIONS
                 1049 ;              -------------
                 1050 ;
 2000            1051         ORG   2000H
 2000            1052 RPTFLG: DS    1
 2001            1053 INTFLG: DS    1
 2002            1054 RSTFLG: DS    1
 2003            1055 TRPFLG: DS    1
 2004            1056         DS    TOPST-$       ;STACK
                 1057
                 1058 $EJECT
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

                 1059 ;##########################################################################
                 1060 ;
                 1061 ;
                 1062 ;
                 1063 ;
                 1064 ;
                 1065 ;
                 1066 ;
                 1067 ;
                 1068 ;                   STAGE II
                 1069 ;                   ========
                 1070 ;
                 1071 ;
                 1072 ;   THIS PROGRAM EXERCISES THE ON BOARD RAM AND I/O PORTS
                 1073 ;  OF THE SDK-85 FOR VERIFICATION USING SIGNATURE ANALYSIS.
                 1074
                 1075
                 1076
                 1077
                 1078 ;   TSTIO MACRO TESTS THE <NOBITS> WIDE OUTPUT
                 1079 ;  PORT AT ADDRESS <IOADR> BY WALKING A 1 ACROSS
                 1080 ;  THE PORT.  EACH TIME NEW DATA IS WRITTEN TO A
                 1081 ;  PORT BY THE 'OUT' INSTRUCTION, THE NEW DATA IS
                 1082 ;  FIRST CLOCKED INTO THE SIGNATURE ANALYSER BY
                 1083 ;  THE POSITIVE EDGE OF RD/ WHEN THE NEXT INST-
                 1084 ;  RUCTION IS FETCHED.  THIS ALLOWS AT LEAST 720ns
                 1085 ;  FOR THE DATA TO SETTLE AT THE OUTPUTS.
                 1086 ;
                 1087 TSTIO   MACRO IOADR,NOBITS
                 1088         LOCAL NXTBIT
                 1089
-                1090         MVI   C,NOBITS  ;;NO. OF BITS TO TEST.
-                1091 NXTBIT: OUT   IOADR     ;;WRITE TEST PATTERN.
-                1092         RLC             ;;SHIFT TEST BIT LEFT.
-                1093         DCR   C         ;;STOP IF ALL BITS TESTED.
-                1094         JNZ   NXTBIT    ;;
-                1095         XRA   A         ;;CLEAR THE PORT AGAIN.
-                1096         OUT   IOADR     ;;
-                1097         INR   A         ;;SET TEST BIT AGAIN.
-                1098
                 1099         ENDM
                 1100
                 1101 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   23
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

 0400            1102        ORG   0400H
                 1103
 0400 F3         1104        DI
 0401 3E0F       1105        MVI   A,0FH     ;PROGRAM RAM PORTS
 0403 D320       1106        OUT   CTRL1     ; AS OUTPUTS.
 0405 D328       1107        OUT   CTRL2     ;
 0407 3EFF       1108        MVI   A,0FFH    ;PROGRAM ROM PORTS
 0409 D302       1109        OUT   DDR1A     ; AS OUTPUTS.
 040B D303       1110        OUT   DDR1B     ;
 040D D30A       1111        OUT   DDR2A     ;
 040F D30B       1112        OUT   DDR2B     ;
                 1113 ;
                 1114 ;  SET ALL OUTPUTS TO 0.
                 1115 ;
 0411 AF         1116        XRA   A
 0412 D321       1117        OUT   RAM1A
 0414 D322       1118        OUT   RAM1B
 0416 D323       1119        OUT   RAM1C
 0418 D329       1120        OUT   RAM2A
 041A D32A       1121        OUT   RAM2B
 041C D32B       1122        OUT   RAM2C
 041E D300       1123        OUT   ROM1A
 0420 D301       1124        OUT   ROM1B
 0422 D308       1125        OUT   ROM2A
 0424 D309       1126        OUT   ROM2B
                 1127 ;
                 1128 ;  INITIALISE 8279 KEYBOARD/DISPLAY CONTROLLER
                 1129 ;
 0426 210019     1130        LXI   H,KDCC    ;ADRS OF 8279 CONTROL REG.
 0429 3608       1131        MVI   M,08H     ;PROGRAM FOR 16 CHAR. LEFT
                 1132                        ; ENTRY DISPLAY, ENCODED SCAN
                 1133                        ; 2 KEY LOCKOUT KEYBOARD.
 042B 3630       1134        MVI   M,30H     ;SET CLOCK CTR TO 16.
 042D 3690       1135        MVI   M,90H     ;WRITE DISPLAY RAM, AUTO-
                 1136                        ; INCREMENT.
 042F 36CD       1137        MVI   M,0CDH    ;CLEAR DISPLAY RAM AND FIFO.
 0431 3EC8       1138        MVI   A,200D    ;WAIT 1ms FOR DISPLAY TO
 0433 3D         1139 WTCLR: DCR   A         ; CLEAR.
 0434 C23304     1140        JNZ   WTCLR     ;
                 1141 ;
                 1142 ;   TEST THE DISPLAY CONTROLLER OUTPUTS BY STORING
                 1143 ;  WALKING 1'S AND 0'S IN THE DISPLAY RAM. THE
                 1144 ;  DISPLAY SHOULD SHOW:
                 1145 ;                      8.8.8.8. 8.8.
                 1146 ;
 0437 25         1147        DCR   H         ;(HL) = 1800H
 0438 B7         1148        ORA   A         ;CLEAR CARRY.
 0439 3E01       1149        MVI   A,01H     ;
 043B 77         1150 WALK1: MOV   M,A       ;WRITE DATA TO DISP. RAM.
 043C 17         1151        RAL             ;SHIFT THE BIT LEFT.
 043D D23B04     1152        JNC   WALK1     ;REPEAT UNTIL 8 BITS
                 1153                        ; ARE TESTED.
 0440 3EFE       1154        MVI   A,0FEH    ;CARRY IS SET:REPEAT
 0442 77         1155 WALK0: MOV   M,A       ; WITH WALKING 0.
 0443 17         1156        RAL             ;
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   24
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE        SOURCE STATEMENT

 0444 DA4204     1157        JC    WALK0     ;
                 1158 ;
                 1159 ;
                 1160 ;       RAM TEST
                 1161 ;       --------
                 1162 ;
 0447 DB38       1163 LOOP:  IN    CS7       ;PULSE CS7/ (WITH RD/ AND WR/)
 0449 D338       1164        OUT   CS7       ; TO SIGNAL START OF RAM #1
                 1165                        ; TEST (AND END OF I/O TEST).
                 1166                                .
                 1167 ;  THE RAM TEST SUBROUTINE IS ACCESSED BY A JMP
                 1168 ;  INSTRUCTION, WITH THE RETURN ADDRESS STORED IN
                 1169 ;  DE, TO AVOID USING UNTESTED RAM FOR SUBROUTINE
                 1170 ;  LINKAGE.
 044B 115404     1171        LXI   D,RET1    ;SAVE RETURN ADDRESS.
 044E 210020     1172        LXI   H,RAM1    ;START ADDRESS OF RAM #1.
 0451 C3EC04     1173        JMP   RAMTST    ;JUMP TO RAM TEST S/R.
 0454 DB30       1174 RET1:  IN    CS6       ;PULSE CS6/ -END OF RAM #1
 0456 D330       1175        OUT   CS6       ; TEST AND START OF RAM
                 1176                        ; #2 TEST.
 0458 116104     1177        LXI   D,RET2    ;SAVE RETURN ADDRESS.
 045B 210028     1178        LXI   H,RAM2    ;START ADDRESS OF RAM #2.
 045E C3EC04     1179        JMP   RAMTST    ;JUMP TO RAM TEST S/R.
 0461 DB10       1180 RET2:  IN    CS2       ;PULSE CS2/ -END OF RAM #2
 0463 D310       1181        OUT   CS2       ; TEST AND START OF I/O
                 1182                        ; TEST.
                 1183 ;
                 1184 ;       OUTPUT PORT TEST
                 1185 ;       ----------------
                 1186 ;
                 1187 ;  THE OUTPUT CAPABILITIES OF THE I/O PORTS AND THE
                 1188 ;  INTEGRITY OF THE CONNECTIONS TO THE I/O CONNECTORS
                 1189 ;  ARE TESTED BY WALKING A '1' ACROSS ALL 10 PORTS.
                 1190 ;
 0465 3E01       1191        MVI   A,01H     ;INITIALISE THE TEST BIT.
                 1192
                 1193        TSTIO ROM1A,8   ;I/O TEST MACRO CALLS.
                 1194+
 0467 0E08       1195+       MVI   C,8
 0469 D300       1196+??0001: OUT  ROM1A
 046B 07         1197+       RLC
 046C 0D         1198+       DCR   C
 046D C26904     1199+       JNZ   ??0001
 0470 AF         1200+       XRA   A
 0471 D300       1201+       OUT   ROM1A
 0473 3C         1202+       INR   A
                 1203+
                 1204        TSTIO ROM1B,8
                 1205+
 0474 0E08       1206+       MVI   C,8
 0476 D301       1207+??0002: OUT  ROM1B
 0478 07         1208+       RLC
 0479 0D         1209+       DCR   C
 047A C27604     1210+       JNZ   ??0002
 047D AF         1211+       XRA   A
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

047E D301       1212+       OUT   ROM1B
0480 3C         1213+       INR   A
                1214+
                1215        TSTIO ROM2A,8
                1216+
0481 0E08       1217+       MVI   C,8
0483 D308       1218+??0003: OUT   ROM2A
0485 07         1219+       RLC
0486 0D         1220+       DCR   C
0487 C28304     1221+       JNZ   ??0003
048A AF         1222+       XRA   A
048B D308       1223+       OUT   ROM2A
048D 3C         1224+       INR   A
                1225+
                1226        TSTIO ROM2B,8
                1227+
048E 0E08       1228+       MVI   C,8
0490 D309       1229+??0004: OUT   ROM2B
0492 07         1230+       RLC
0493 0D         1231+       DCR   C
0494 C29004     1232+       JNZ   ??0004
0497 AF         1233+       XRA   A
0498 D309       1234+       OUT   ROM2B
049A 3C         1235+       INR   A
                1236+
                1237        TSTIO RAM1A,8
                1238+
049B 0E08       1239+       MVI   C,8
049D D321       1240+??0005: OUT   RAM1A
049F 07         1241+       RLC
04A0 0D         1242+       DCR   C
04A1 C29D04     1243+       JNZ   ??0005
04A4 AF         1244+       XRA   A
04A5 D321       1245+       OUT   RAM1A
04A7 3C         1246+       INR   A
                1247+
                1248        TSTIO RAM1B,8
                1249+
04A8 0E08       1250+       MVI   C,8
04AA D322       1251+??0006: OUT   RAM1B
04AC 07         1252+       RLC
04AD 0D         1253+       DCR   C
04AE C2AA04     1254+       JNZ   ??0006
04B1 AF         1255+       XRA   A
04B2 D322       1256+       OUT   RAM1B
04B4 3C         1257+       INR   A
                1258+
                1259        TSTIO RAM1C,6
                1260+
04B5 0E06       1261+       MVI   C,6
04B7 D323       1262+??0007: OUT   RAM1C
04B9 07         1263+       RLC
04BA 0D         1264+       DCR   C
04BB C2B704     1265+       JNZ   ??0007
04BE AF         1266+       XRA   A
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   26
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

 04BF D323       1267+       OUT   RAM1C
 04C1 3C         1268+       INR   A
                 1269+
                 1270        TSTIO RAM2A,8
                 1271+
 04C2 0E08       1272+       MVI   C,8
 04C4 D329       1273+??0008: OUT   RAM2A
 04C6 07         1274+       RLC
 04C7 0D         1275+       DCR   C
 04C8 C2C404     1276+       JNZ   ??0008
 04CB AF         1277+       XRA   A
 04CC D329       1278+       OUT   RAM2A
 04CE 3C         1279+       INR   A
                 1280+
                 1281        TSTIO RAM2B,8
                 1282+
 04CF 0E08       1283+       MVI   C,8
 04D1 D32A       1284+??0009: OUT   RAM2B
 04D3 07         1285+       RLC
 04D4 0D         1286+       DCR   C
 04D5 C2D104     1287+       JNZ   ??0009
 04D8 AF         1288+       XRA   A
 04D9 D32A       1289+       OUT   RAM2B
 04DB 3C         1290+       INR   A
                 1291+
                 1292        TSTIO RAM2C,6
                 1293+
 04DC 0E06       1294+       MVI   C,6
 04DE D32B       1295+??0010: OUT   RAM2C
 04E0 07         1296+       RLC
 04E1 0D         1297+       DCR   C
 04E2 C2DE04     1298+       JNZ   ??0010
 04E5 AF         1299+       XRA   A
 04E6 D32B       1300+       OUT   RAM2C
 04E8 3C         1301+       INR   A
                 1302+
 04E9 C34704     1303        JMP   LOOP      ;REPEAT TESTS INDEFINITELY.
                 1304 ;
                 1305 ;
                 1306 ;       RAM TEST SUBROUTINE.
                 1307 ;
                 1308 ;  THIS ROUTINE TESTS THE 256 BYTE BLOCK OF RAM
                 1309 ;  STARTING AT (HL) AS FOLLOWS:
                 1310 ;    1. THE RAM IS FILLED WITH 00H.
                 1311 ;    2. FFH IS WRITTEN INTO EACH LOCATION, AFTER
                 1312 ;       READING THE LOCATION (TO VERIFY IT STILL CONTAINS
                 1313 ;       00H), FROM THE TOP DOWN.
                 1314 ;    3. 00H IS AGAIN WRITTEN FROM BOTTOM UP, AFTER CHECKING
                 1315 ;       FOR FFH.
                 1316 ;    4. EACH LOCATION IS TESTED BY WRITING AND READING THREE
                 1317 ;       BYTES (0FH,33H,55H) TO CHECK THE INDEPENDENCE
                 1318 ;       OF EACH BIT.
                 1319 ;
                 1320 ;  THE DATA READ BACK IS VERIFIED BY THE SIGNATURE
                 1321 ;  ANALYSER.
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   27
SDK-85 Self Diagnostic V34

 LOC  OBJ        LINE       SOURCE STATEMENT

                 1322 ;
 04EC 0E00       1323 RAMTST: MVI   C,00H     ;INITIALISE BYTE CTR.
 04EE 3600       1324 NXTLOC: MVI   M,00H     ;WRITE 0'S IN ALL
 04F0 0C         1325         INR   C         ; LOCATIONS.
 04F1 CAF804     1326         JZ    READ0     ;STOP IF 256 DONE.
 04F4 23         1327         INX   H         ;ELSE INCREMENT ADRS
 04F5 C3EE04     1328         JMP   NXTLOC    ; AND CONTINUE.
                 1329 ;
                 1330 ;NOW READ BACK THE DATA AND WRITE FFH.
                 1331 ;
 04F8 7E         1332 READ0:  MOV   A,M       ;DUMMY READ.
 04F9 36FF       1333         MVI   M,0FFH    ;STORE 1'S.
 04FB 0C         1334         INR   C         ;STOP AFTER 256
 04FC CA0305     1335         JZ    READ1     ; LOCATIONS.
 04FF 2B         1336         DCX   H         ;DECREMENT ADRS AND
 0500 C3F804     1337         JMP   READ0     ; REPEAT.
                 1338 ;
                 1339 ;READ BACK FF'S AND WRITE 00'S AGAIN.
                 1340 ;
 0503 7E         1341 READ1:  MOV   A,M       ;DUMMY READ.
 0504 3600       1342         MVI   M,00H     ;STORE 0'S.
 0506 0C         1343         INR   C         ;
 0507 CA0E05     1344         JZ    BITCHK    ;
 050A 23         1345         INX   H         ;INCREMENT ADDRESS.
 050B C30305     1346         JMP   READ1     ;
                 1347 ;
                 1348 ;NOW TEST THE INDEPENDENCE OF EACH BIT IN EACH BYTE
                 1349 ;BY WRITING A UNIQUE BINARY SEQUENCE TO EACH BIT.
                 1350 ;
 050E 360F       1351 BITCHK: MVI   M,0FH     ;WRITE AND READ
 0510 7E         1352         MOV   A,M       ; 00001111.
 0511 3633       1353         MVI   M,33H     ;WRITE AND READ
 0513 7E         1354         MOV   A,M       ; 00110011.
 0514 3655       1355         MVI   M,55H     ;WRITE AND READ
 0516 7E         1356         MOV   A,M       ; 01010101.
 0517 3600       1357         MVI   M,00H     ;CLEAR THE BYTE.
 0519 0C         1358         INR   C         ;256 BYTES DONE?
 051A CA2105     1359         JZ    ENDTST    ;EXIT IF SO.
 051D 2B         1360         DCX   H         ;ELSE DECREMENT AND
 051E C30E05     1361         JMP   BITCHK    ; REPEAT.
                 1362 ;
                 1363 ; RETURN BY JUMPING TO THE ADDRESS SAVED
                 1364 ; IN DE.
                 1365 ;
 0521 EB         1366 ENDTST: XCHG            ;(DE)->(HL)
 0522 E9         1367         PCHL            ;(HL)->(PC)
                 1368 ;
                 1369 ;
                 1370         END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS
```

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0 MODULE PAGE 28
SDK-85 Self Diagnostic V34

USER SYMBOLS

BITCHK	A 050E	CHRTAB	A 0728	CHSOD	A 00BC	CLRDSP	A 06D8	CLRFLG	A 070D	CLRKBD	A 06FF
CONVRT	A 06E2	CS2	A 0010	CS6	A 0030	CS7	A 0038	CTRL1	A 0020	CTRL2	A 0028
DDR1A	A 0002	DDR1B	A 0003	DDR2A	A 000A	DDR2B	A 000B	DELAY	A 0640	DISPCH	A 0083
DLY500	A 064D	DLYSID	A 00A2	DSPTST	A 06BF	END0	A 006B	ENDTST	A 0521	ER81	A 02D4
ER82	A 02D3	ER83	A 02D2	ERR1	A 0714	ERR2	A 0719	ERR3	A 071E	ERR4	A 0723
ERRDSP	A 069F	EXEC	A 0050	EXIT6	A 022B	INTFLG	A 2001	IOERR	A 0188	IOERT	A 0183
ITR34	A 0191	ITR5	A 01F7	ITR7	A 0282	KBDE1	A 068F	KBDE2	A 0693	KBDE3	A 0697
KBDE4	A 069B	KDCC	A 1900	KDCD	A 1800	LAST	A 0320	LOOP	A 0447	LOOP1	A 0643
LOOP2	A 0641	NEXT	A 0051	NXTAD	A 0305	NXTCH	A 0057	NXTLOC	A 04EE	PASTIR	A 003F
RAM1	A 2000	RAM1A	A 0021	RAM1B	A 0022	RAM1C	A 0023	RAM2	A 2800	RAM2A	A 0029
RAM2B	A 002A	RAM2C	A 002B	RAMTST	A 04EC	RDKBD	A 06EF	READ0	A 04F8	READ1	A 0503
RET1	A 0454	RET2	A 0461	RETURN	A 068D	ROM1A	A 0000	ROM1B	A 0001	ROM2A	A 0008
ROM2B	A 0009	RPT34	A 00D6	RPT5	A 01BF	RPT7	A 0258	RPT8	A 02DB	RPT9	A 02FE
RPTCH	A 0052	RPTFLG	A 2000	RSTFLG	A 2002	RSTIR	A 002C	RTRN	A 06FD	SERERR	A 00B4
SEROUT	A 009E	SETBIT	A 0725	SETFLG	A 0707	SHFTA	A 00E4	SHFTB	A 012E	STORE	A 070F
TEST0	A 0051	TEST1	A 007B	TEST2	A 0098	TEST3	A 00CD	TEST4	A 0154	TEST5	A 019E
TEST6	A 01FF	TEST7	A 023A	TEST8	A 028A	TEST9	A 02F4	TIM1HI	A 0025	TIM1LO	A 0024
TIM2HI	A 002D	TIM2LO	A 002C	TOPST	A 2100	TRPFLG	A 2003	TSTIO	+ 0000	TSTKBD	A 0658
WAIT5	A 01B1	WAIT6	A 0210	WAIT7	A 0245	WAIT8	A 0292	WALK0	A 0442	WALK1	A 043B
WFIP	A 008C	WFTRP	A 0166	WRFF	A 0302	WT6	A 0232	WTCLR	A 0433	WTF0	A 02B9
WTF1	A 02C5										

ASSEMBLY COMPLETE, NO ERRORS

APPENDIX F

SDK-85 SIGNATURE ANALYSIS PROCEDURE OPERATING INSTRUCTIONS

SDK-85 SIGNATURE ANALYSIS PROCEDURE

(For an SDK-85, using V1.2 of the SDK-85 monitor, with V34 of the SA test program (SDK85S.V34, Appendix E) and V8 of the 8085 self test program (SLFTST.V8, Appendix G).)

The signature analysis routine for the SDK-85 is divided into three stages:

Stage I — is the basic stage, designed to verify the CPU, ROM and bus integrity while the CPU free-runs through its entire address range.

Stage II — is an intermediate stage designed to test RAM, output ports and those bus signals not testable at Stage I.

Stage III — is the self-diagnostic stage in which the processor executes a program stored in the test ROM to verify some of its more complex operations, as well as to more fully test its peripheral chips.

In each stage the procedure must be followed through step by step {(1), (2), (3) etc.}, performing the specified test or taking the specified signature at each test. If the result of a test is incorrect, the likely cause of the fault will be indicated in the procedure. EACH FAULT MUST BE CORRECTED AS SOON AS IT IS IDENTIFIED AND THE ENTIRE SIGNATURE ANALYSIS ROUTINE MUST BE STARTED AGAIN - FROM STAGE I STEP (1). The only exceptions to this rule occur in Stage III, in which some errors may be stepped over to proceed to later tests, as described in the procedure for Stage III.

To accommodate signature analysis, the SDK-85 must have been modified to include an address selection DIL socket. The appropriate plug (white, orange or blue) must be inserted in this socket at each stage, as specified in step (1) of each stage. For normal operation (i.e. execution of the SDK-85 monitor stored in the ROM, A14) the WHITE plug should be inserted.

In addition a free-run adapter (holding buffers B1 and B2 and a zero insertion force socket for the 8085 (A11)) is needed for Stage 1.

A number of test connectors - P3, P4, P5 and P1/P2 (including a buffer to apply an interrupt vector to the 8085 data bus) - are required to execute all tests at Stage III.

IT IS ASSUMED THAT ALL TEST HARDWARE (ADAPTERS AND CONNECTORS) IS WORKING CORRECTLY THROUGHOUT THE S.A. ROUTINE. If there is any doubt this (simple) external hardware should be tested separately.

Signature analyser connections to the system are generally specified in the form

CLOCK (⌝) to < pin number >

START (⌝) to < pin number >

STOP (⌟) to < pin number >

where (⌟) or (⌝) indicates that the given signature analyser input should trigger on the negative or positive edge (respectively).

<pin number> will be specified in the form:

IC or connector number - pin number

(e.g. A7-1 is pin 1 of IC A7).

Signature analyser connections to the system may not be changed while power is applied to the system (but positive/negative edge trigger switches, of course, may be). The power must be turned off, new connections made, power turned on and RESET button pressed to proceed with the test.

NOTES:

(1) The procedure in some steps may involve instructions to proceed to subsequent steps under certain test conditions. These must be obeyed if the given test condition is satisfied. If there is no instruction to the contrary, proceed to the following step unless a fault is identified, in which case it must be repaired and the routine recommenced.

(2) Where instruction to replace a device is qualified by an asterisk (e.g. replace A11*) it should be noted that the faulty behaviour of the device may be due to a short circuit on one or more of its output lines, rather than an internal fault.

In such cases either use a current tracing device to isolate any short circuits before replacing the device or replace the device and look for short circuits on the output lines if the fault occurs again.

(3) Some pins at which signatures are to be taken are marked with a double asterisk (e.g. A11-32**). In these cases two signatures are

to be taken -

one with the CLOCK input set to trigger on the positive edge

one with the CLOCK input set to trigger on the negative edge

with the START and STOP inputs as specified in the first instance.

The signatures expected are noted in the form

positive edge / negative edge

and both must be correct.

After taking these signatures, the CLOCK edge select switch must be returned to the originally specified position.

(4) It is possible that, due to timing margin differences and spurious effects within the system, some signatures will appear a little unstable. However, if one occurs much more frequently than any other then it can be taken as a valid signature at that point.

(5) The procedure is neither foolproof nor complete, so the conclusions drawn by following the procedure may be incorrect in some cases.

If, after following the specified procedure,a fault is not cured then more conventional debugging techniques must be used - with or without the help of any information that may have been obtained by following the signature analysis procedure up to the point of its failure.

Link Placement

In Stages I and II, and up to step (6) of Stage III, when three links must be removed, the following links on the SDK-85 board must be inserted or removed as specified below.

Link	Status
1-2	Out if A9 is installed, In otherwise.
3-4	In if buffer A5 in installed, Out otherwise.
3-5	Out if buffer A5 is installed, In otherwise.
6-8	Out if buffer A3 is installed, In otherwise.
7-8	In if buffer A3 is installed, Out otherwise.
9-10	In
11-12	In
13-14	Out
15-16	Out
17-18	In
18-19	Out
20-21	In
22-23	In
23-24	Out
25-26	Out
25-27	In
28-29	In if A14 is 8755, Out if A14 is 8755A.
29-30	Out if A14 is 8755, In if A14 is 8755A.
31-32	In if A15 is 8755, Out if A15 is 8755A.
32-33	Out if A15 is 8755, In if A15 is 8755A.

SDK-85 SIGNATURE ANALYSIS

Stage I

(1) INITIALISATION:

Remove 8085 (A11) from its socket on the SDK-85 board and place in the free-run adapter (Appendix C).

Plug the 40 pin plug from the adapter into the 8085 socket.

Ensure the WHITE plug is inserted in the address selection socket.

Connect the signature analyser leads as follows:

GROUND	to GND (Adapter test pin 1)
CLOCK (↓)	to ALE (Adapter test pin 4)
START (↓)	to A15 (Adapter test pin 2)
STOP (↓)	to A15 (Adapter test pin 3)

(2) Apply power and, using a logic probe, voltmeter, or CRO verify that:

Pin A11 - 20 is at GND (0V)

Pin A11 - 40 is at V_{cc} (5V)

Pin A11 - 36 is Hi, and goes Lo when the RESET key is pressed.

If not, check the connections to the 8085 through the adapter cable and the socket on the SDK-85 board.

(3) Again using logic probe, CRO a voltmeter, check that pin A11 - 3 is Lo, and goes Hi when the RESET button is pressed. If not replace A11.

(4) Verify the following signatures:

Pin	Signal Name	Signature
A11 - 5	SID	0000
- 6	TRAP	0000
- 7	RST 7.5	0000
- 8	RST 6.5	0000
- 9	RST 5.5	0000
- 10	INTR	0000
- 35	RDY	0001
- 39	HOLD	0000

If any errors then the free-run adapter is faulty.

(5) Take signatures at the following pins:

Pin	Signal Name	Signature
A11 - 28	A15	755U
- 27	A14	3827
- 26	A13	3C96
- 25	A12	HAP7
- 24	A11	1293
- 23	A10	HPP0
- 22	A9	2H70
- 21	A8	HC89

If any errors then replace A11*.

(6)

Pin	Signal Name	Signature
B1 - 18	BA7	52F8
- 16	BA6	UPFH
- 14	BA5	OAFA
- 12	BA4	5H21
B2 - 18	BA3	7F7F
- 16	BA2	CCCC
- 14	BA1	5555
- 12	BAØ	UUUU

NOTE: B1 and B2 are buffers on the test adapter.

If any errors then go to (7).

else to to (8).

(7)

Pin	Signal Name	Signature
A11 - 19	AD7	52F8
- 18	AD6	UPFH
- 17	AD5	OAFA
- 16	AD4	5H21
- 15	AD3	7F7F
- 14	AD2	CCCC
- 13	AD1	5555
- 12	ADØ	UUUU

If any errors then replace A11

else there is a short circuit on the system data bus.

(8) Verify the signature at all of the listed pins.

Pins	Signature
A13-19, A14-19, A15-19, A16-19, A17-19, A4-12, A4-14, A6-22	5258
A13-18, A14-18, A15-18, A16-18, A17-18, A4-9, A4-11, A6-20	UPFH
A13-17, A14-17, A15-17, A16-17, A17-17, A4-7, A4-5, A6-18	OAFA
A13-16, A14-16, A15-16, A16-16, A17-16, A4-4, A4-2, A6-16	5H21
A13-15, A14-15, A15-15, A16-15, A17-15, A7-12, A7-14, A6-9	7F7F
A13-14, A14-14, A15-14, A16-14, A17-14, A7-9, A7-11, A6-7	CCCC
A13-13, A14-13, A15-13, A16-13, A17-13, A7-4, A7-2, A6-5	5555
A13-12, A14-12, A15-12, A16-12, A17-12, A7-7, A7-5, A6-3	UUUU
A14-21, A15-21, A13-21	HC89
A14-22, A15-22	2H7O
A14-23, A15-23	HPPO

If the signature at any pin is wrong then there is an open circuit on the address/data bus line to that pin from A11. Trace and repair the line.

(9)

Pin	Signal Name	Signature
A10 - 7	CS7/	A68C
- 9	CS6/	A277
- 10	CS5/	9840
- 11	CS4/	8P4P
- 12	CS3/	5P18
- 13	CS2/	2828
- 14	CS1/	02H7
- 15	CS0/	3ADF

If any errors then go to (10)

else go to (11).

(10)

Pin	Signal Name	Signature
A10 - 1	A11	1293
- 2	A12	HAP7
- 3	A13	3C96
- 4	A14	3827
- 5	A15	755U
- 6	V_{CC}	0001
- 8	GND	0000
- 10	V_{CC}	0001

If any errors then there is an open circuit to the input pin of A10

else replace A10*.

(11)

Pin	Signal Name	Signature
A17 - 8	CS5/	9840
A16 - 8	CS4/	8P4P
A15 - 1	CS1/	02H7
A14 - 1	CS0/	3APF
A13 - 22	CS3/	5P18

If any errors then there is an open circuit in the line to that pin from A10.

(12) Change the signature analyser connections:

CLOCK (⌐) to RD/ (A11 - 32)

START (¬) to A15 (TP2)

STOP (¬) to A15 (TP3)

Verify the following signatures:

Pin	Signal Name	Signature
A11 - 11	INTA/	0001
- 31	WR/	0001
- 32**	RD/	0000/0001
- 34	IO/M	0000
- 38	HLDA	0000

If any errors then replace A11*

else go to (13)

(13) Check all of the following signatures:

Pins	Signal Name	Signature
A13-9, A14-4, A15-4, A16-4, A17-4, A9-10	RESET	0000
A14-2, A15-2,	CE	0001
A13-10, A14-9, A15-9, A16-9, A17-9**	RD/	0000/0001
A14-5, A15-5	V_{DD}	0000
A14-8, A15-8	IOR/	0001
A14-10, A15-10, A16-10,	WR/	0001
A14-7, A15-7, A16-7	IO/$\overline{M}$	0000
A2-7**	RD/	0000/0001
A2-8, A4-8, A7-8, A9-7	GND	0000
A2-15	GND	0000
A2-16, A4-16, A7-16, A9-14	V_{cc}	0001
A1-2, A3-1, A5-1, A6-2	GND	0000
A1-11, ,	V_{cc}	0001
A1-12, A3-8, A5-8, A6-12, A8-7	GND	0000
A1-13, , A6-13	V_{cc}	0001
A1-14, , A6-14	V_{cc}	0001
A1-24, A3-16, A5-16, A6-24, A8-14	V_{cc}	0001
A3-15, A5-15,	GND	0000
A3-4,	HOLD	0000
A3-9,	HLDA	0000
A3-12, , A8-1	INTA/	0001
A5-12	RST6.5	0000
, A8-4	A15	755U
, A8-12**	RD/	0000/0001
, A8-13**	RD/	0000/0001

If any errors then the connection to the pin in error is open circuited.

(14) Verify the following signatures; remedy the fault specified if the signature at any pin is incorrect else go to (15)

Pin	Signature	Fault if incorrect signature(s)
A13-4	0000	A13 faulty - replace*
A16-6	0001	A16 faulty - replace*
A8-11**	0001/0000	A8 faulty - replace*
A8-5**	0001/0000	Open cct from A8-11 to A8-5
A8-6**	755P/0001	A8 faulty - replace*
A8-2**	755P/0001	Open cct from A8-6 to A8-2
A8-3**	755U/0000	A8 faulty - replace*
A7-15**	755U/0000	Open cct from A8-3 to A7-15
A4-15**	755U/0000	Open cct from A8-3 to A4-15

(15) Change the signature analyser connections to the following:

CLOCK (⌐) to CLK (A13-3)

START (⌐) to RD/ (A11-32)

STOP (⌐) to RD/ (A11-32)

Take signature at A11-37**

If signature(s) are 0000/000U then go to (16)

else replace A11* or there is an open circuit from A11-37 to A3-3.

(16)

Pin	Signal Name	Signature
A13-3**	CLK	0000/000U
A14-3**	CLK	0000/000U
A15-3**	CLK	0000/000U
A16-3**	CLK	0000/000U
A17-3**	CLK	0000/000U
18-9**	CLK	0000/000U
A8-19**	CLK	0000/000U
A5-7**	CLK	0000/000U
J5-24**	CLK	0000/000U

If any signature is wrong then there is an open circuit from A11-37 to that pin.

(17)

Pin	Signature(s)	Fault if bad signature
A5-6**	0000/000U	A5 faulty - replace*
J1-4**	0000/000U	Open circuit from A5-6 to J1-4
A8-8**	000U/0000	A8 faulty - replace*
A9-3**	000U/0000	Open circuit from A8-8 to A9-3
A9-1	000U	Bad connection from V_{CC} to A9-1
A9-2	0000	Open circuit from A11-38 to A9-2
A9-11	0000	Bad connection from GND to A9-11
A9-12	0000	Bad connection from GND to A9-12
A9-13	0000	Bad connection from GND to A9-13
A9-9	000U	A9 faulty - replace*
A9-4	000U	Open circuit from A9-9 to A9-4
A9-5	0000	A9 faulty - replace*
A1-1	0000	Open circuit from A9-5 to A1-1
A2-1	0000	Open circuit from A9-5 to A2-1
A4-1	0000	Open circuit from A9-5 to A4-1
A6-1	0000	Open circuit from A9-5 to A6-1
A7-1	0000	Open circuit from A9-5 to A7-1

If no errors then go to (18).

(18) Change signature analyse connections to:

CLOCK (↓) to ALE (Adapter TP4)

START (↓) to A15 (Adapter TP2)

STOP (↑) to A15 (Adapter TP3)

Pin	Signature(s)	Fault if incorrect signature
A14-11**	0000/755U	Open circuit from A11-30 to A14-11
A15-11**	0000/755U	Open circuit from A11-30 to A15-11
A16-11**	0000/755U	Open circuit from A11-30 to A16-11
A17-11**	0000/755U	Open circuit from A11-30 to A17-11
A2-9**	0000/755U	Open circuit from A11-30 to A2-9
A6-11**	00UU/755U	Open circuit from A11-30 to A6-11
A2-10**	0000/755U†	A2 faulty - replace*
J1-10**	0000/755U†	Open circuit from A2-10 to J1-10

If no errors then go to (19).

† These signatures are prone to instability.

(19) Set signature analyser up as follows:

CLOCK (⏜) to ALE (Adapter TP4)

START (⏜) to A15 (Adapter TP2)

STOP (⏜) to A15 (Adapter TP3)

Pin	Signal Name	Signature
J1-26	B-D7	52F8
J1-28	B-D6	UPFH
J1-30	B-D5	OAFA
J1-32	B-D4	5H21
J1-34	B-D3	7F7F
J1-36	B-D2	CCCC
J1-38	B-D1	5555
J1-40	B-DØ	UUUU

If no errors then go to (21)

else go to (20).

(20)

Pin	Signal Name	Signature
A4-13	B-D7	52F8
A4-10	B-D6	UPFH
A4-6	B-D5	OAFA
A4-3	B-D4	5H21
A7-13	B-D3	7F7F
A7-10	B-D2	CCCC
A7-3	B-D1	5555
A7-6	B-DØ	UUUU

If any errors then replace A7* and/or A4*

else the fault is an open circuit on a line from A7/A4 to J1.

(21) Change the signature analyser connections to:

CLOCK (⏊) to RD/ (A11-32)

START (⏉) to A15 (TP2)

STOP (⏉) to A15 (TP3)

Pin	Signature(s)	Fault if incorrect signatures
A2-6**	0000/0001	A2 faulty - replace*
J2-6**	0000/0001	Open circuit from A2-6 to J2-6
A3-3	0000	A3 faulty - replace*
A3-10	0000	A3 faulty - replace*
A3-13	0001	A3 faulty - replace*
A5-13	0000	A5 faulty - replace*
J1-12	0000	Open circuit from A3-10 to J1-12
J1-16	0001	Open circuit from A3-13 to J1-16

If no errors then go to (22).

(22)

Pin	Signal Name	Signatures
J2-10	B-A15	755U
J2-12	B-A14	3827
J2-14	B-A13	3C96
J2-16	B-A12	HAP7
J2-18	B-A11	1293
J2-20	B-A10	HPP0
J2-22	B-A9	2H70
J2-24	B-A8	HC89

If no errors then go to (25)

else go to (23).

(23)

Pin	Signal Name	Signatures
A1-21	B-A15	755U
A1-19	B-A14	3827
A1-17	B-A13	3C96
A1-15	B-A12	HAP7
A1-10	B-A11	1293
A1-8	B-A10	HPP0
A1-6	B-A9	2H70
A1-4	B-A8	HC89

If any errors then to to (24)

else there is an open circuit in line(s) from A1 to J2.

(24)

Pin	Signal Name	Signatures
A1-22	BA15	755U
A1-20	BA14	3827
A1-18	BA13	3C96
A1-16	BA12	HAP7
A1-9	BA11	1293
A1-7	BA10	HPP0
A1-5	BA9	2H70
A1-3	BA8	HC89

If any errors then there is an open circuit on line(s) from A11 to A1

else replace A1*.

(25)

Pin	Signal Name	Signature
J2-26	B-A7	52F8
J2-28	B-A6	UPFH
J2-30	B-A5	OAFA
J2-32	B-A4	5H21
J2-34	B-A3	7F7F
J2-36	B-A2	CCCC
J2-38	B-A1	5555
J2-40	B-AØ	UUUU

If any errors then go to (26)

else go to (27).

(26)

Pin	Signal Name	Signature
A6-21	B-A7	52F8
A6-19	B-A6	UPFH
A6-17	B-A5	OAFA
A6-15	B-A4	5H21
A6-10	B-A3	7F7F
A6-8	B-A2	CCCC
A6-6	B-A1	5555
A6-4	B-AØ	UUUU

If any errors then replace A6*

else there is an open circuit on line(s) from A6 to J2.

(27) Change the signature analyser connections to the following:

CLOCK (↓) to RD/ (A11-32)

START (↓) to CSØ/ (A14-1)

STOP (↓) to CSØ/ (A14-1)

Pin	Signal Name	Signature
A14-12	DØ	PHAC
-13	D1	H75P
-14	D2	31CF
-15	D3	F041
-16	D4	UAU4
-17	D5	07UU
-18	D6	F894
-19	D7	UP74

If any errors then replace A14.

(28) Change the signature analyser connections to the following:

CLOCK (↓) to RD/ (A11-32)

START (↓) to CS1/ (A10-14)

STOP (↓) to CS1/ (A10-14)

Pin	Signal Name	Signature
A15-12	DØ	6PF3
-13	D1	2398
-14	D2	P4A5
-15	D3	0A0C
-16	D4	C37F
-17	D5	HHU8
-18	D6	4FCC
-19	D7	01FC

If any errors then replace A15.

Stage II

(1) INITIALISATION

Replace the 8085 (A11) in its socket on the SDK-85 board.

Insert the ORANGE plug into the address selection socket.

Connect the signature analyser leads as follows:

GROUND (⏚) to WR/ (A11-31)
START (⏜) to CS7/ (A10-7)
STOP (⏚) to CS6/ (A10-9).

(2) Apply power to the SDK-85 and, using logic probe, CRO or voltmeter verify that

Pin A11-20 is at GND (0V)
Pin A11-40 is at V_{cc} (5V)
Pin A11-36 is LO, and goes HI when the reset key is pressed.

If not, check the appropriate connection to A11 from the power supply and/or the RESET key.

(3) Check the effect of the RESET key at the following pins and if the effect is not as specified, remedy the fault indicated.

Pin	RESET not pressed	RESET pressed	Fault if not correct
A11-3	LO	HI	A11 faulty - replace*
A13-9	LO	HI	Open circuit from A11-3 to A13-9
A14-4	LO	HI	Open circuit from A11-3 to A14-4
A15-4	LO	HI	Open circuit from A11-3 to A15-4
A16-4	LO	HI	Open circuit from A11-3 to A16-4
A17-4	LO	HI	Open circuit from A11-3 to A17-4
A12-2	LO	HI	Open circuit from A11-3 to A12-2
A12-14	LO	HI	Open circuit from A11-3 to A12-14
A5-4	LO	HI	Open circuit from A11-3 to A5-4
A9-10	LO	HI	Open circuit from A11-3 to A9-10
A9-9	HI	LO	A9 faulty - replace*
A9-4	HI	LO	Open circuit from A9-9 to A9-4
A9-5	LO	HI	A9 faulty - replace*
A1-1	LO	HI	Open circuit from A9-5 to A1-1
A2-1	LO	HI	Open circuit from A9-5 to A2-1
A4-1	LO	HI	Open circuit from A9-5 to A4-1
A6-1	LO	HI	Open circuit from A9-5 to A6-1
A7-1	LO	HI	Open circuit from A9-5 to A7-1
A5-3	LO	HI	A5 faulty - replace*
J1-22	LO	HI	Open circuit from A5-3 to J1-22

If no errors then go to (4)

(4) Check the signature at A11-40 (V_{cc})

If signature - 1HCP then go to (5)

else replace A11* - WR/ output error.

(5) Check the following signatures

Pin	Signal Name	Signature
A11-6	TRAP	1HCP
A11-7	RST 7.5	1HCP
A11-8	RST 6.5	0000
A11-9	RST 5.5	0000
A11-10	INTR	0000
A11-35	RDY	1HC P
A11-39	HOLD	0000

If any errors then there is a bad connection to this pin from logic elsewhere on the board (which was checked at Stage I).

(6)

Pin	Signature(s)	Fault if incorrect signature
A13-11**	0000/1HCP	Open circuit from A11-31 to A13-11
A14-10**	0000/1HCP	Open circuit from A11-31 to A14-10
A15-10**	0000/1HCP	Open circuit from A11-31 to A15-10
A16-10**	0000/1HCP	Open circuit from A11-31 to A16-10
A17-10**	0000/1HCP	Open circuit from A11-31 to A17-10
A2-12**	0000/1HCP	Open circuit from A11-31 to A2-12
A2-13**	0000/1HCP	A2 faulty - replace*
J2-4**	0000/1HCP	Open circuit from A2-13 to J2-4

If no errors then go to (7).

(7) Change the signature analyser connections to the following

CLOCK (⌐‾) to ALE (A11-30)

START (⌐‾) to CS7/ (A10-7)

STOP (‾⌐) to CS6/ (A10-9)

Pin	Signal Name	Signature
A11-29	SØ	PH34
A11-33	S1	61FO
A11-34	IO/$\overline{M}$	HOA8

If any errors then replace A11*

else go to (8).

(8)

Pin	Signature	Fault if incorrect signature
A14-7	HOA8	Open circuit from A11-34 to A14-7
A15-7	HOA8	Open circuit from A11-34 to A15-7
A16-7	HOA8	Open circuit from A11-34 to A16-7
A17-7	HOA8	Open circuit from A11-34 to A17-7
A2-4	HOA8	Open circuit from A11-34 to A2-4
A2-3	HOA8	A2 faulty - replace*
A3-7	PH34	Open circuit from A11-29 to A3-7
A3-6	PH34	A3 faulty - replace*
A5-9	61FO	Open circuit from A11-33 to A5-9
A5-10	61FO	A5 faulty - replace*
J1-6	61FO	Open circuit from A5-10 to J1-6
J1-8	PH34	Open circuit from A3-6 to J1-8
J2-8	HOA8	Open circuit from A2-3 to J2-8

If no errors then go to (9).

(9) Change the signature analyser connections to the following:

CLOCK (⌐↓) to RD/ (A11-32)

START (⌐↓) to CS7/ (A10-7)

STOP (¬↑) to CS6/ (A10-9)

If A16-40 = 9PHH then go to (10)

else replace A11

(10)

Pin	Signal Name	Signature
A16-19	D7	11F4
A16-18	D6	2107
A16-17	D5	44FP
A16-16	D4	AU6A
A16-15	D3	4583
A16-14	D2	554A
A16-13	D1	2515
A16-12	DØ	735F

If any errors then replace A16

else go to (11).

(11) Change the signature analyser connections to the following:

CLOCK (⌐↓) to RD/ (A11-32)

START (⌐↓) to CS6/ (A10-9)

STOP (¬↑) to CS2/ (A10-13)

If A17-40 = 9PHH then go to (12)

else replace A11.

(12)

Pin	Signal Name	Signature
A17-19	D7	11F4
A17-18	D6	2107
A17-17	D5	PPU9
A17-16	D4	055F
A17-15	D3	37U9
A17-14	D2	UU7F
A17-13	D1	2515
A17-12	DØ	H96A

If any errors then replace A17

else go to (13).

(13) Change the signature analyser connections to the following:

CLOCK (↓) to RD/ (A11-32)

START (↓) to CS2/ (A10-13)

STOP (↑) to CS7/ (A10-7)

If A11-40 = 28P9 then go to (14)

else replace A11.

(14) For each of the following output port signatures, first take the signatures at the specified connector pin (J3, J4, J5).

If this is correct then proceed to the next pin

else check the signatures at the corresponding I/O port (A14, A15, A16, A17).

If this is correct then the fault is an open circuit between the I/O port and the connectors

else the I/O port is faulty, replace the chip.*

Connector Pin	I/O Port Pin	Signature
J3-1	A14-38	4195
J3-2	A14-39	2483
J3-3	A14-36	6P58
J3-4	A14-37	FAHF
J3-5	A14-34	O5UU
J3-6	A14-35	2FOC
J3-7	A14-32	H6F4
J3-8	A14-33	UUAH
J3-9	A14-30	804A
J3-10	A14-31	4500
J3-11	A14-28	8H66
J3-12	A14-29	251A
J3-13	A14-26	8917
J3-14	A14-27	C312
J3-15	A14-24	PU32
J3-16	A14-25	8CHP
J3-17	A15-38	O56U
J3-18	A15-39	2POA
J3-19	A15-36	FP9H
J3-20	A15-37	C79H
J3-21	A15-34	54H1
J3-22	A15-35	4PA9
J3-23	A15-32	74AO
J3-24	A15-33	68P9
J3-25	A15-30	7743
J3-26	A15-31	O2PP
J3-27	A15-28	F92A
J3-28	A15-29	A192
J3-29	A15-26	A6UP
J3-30	A15-27	954H
J3-31	A15-24	92AA
J3-32	A15-25	7U25

(continued on next page)

(14) Continued

Connector Pin	I/O Port Pin	Signature
J4-1	A16-2	3761
J4-2	A16-5	FP6P
J4-3	A16-39	4FP3
J4-4	A16-1	C099
J4-5	A16-37	FH48
J4-6	A16-38	719A
J4-7	A16-35	6F39
J4-8	A16-36	82H8
J4-9	A16-33	7031
J4-10	A16-34	1FP0
J4-11	A16-31	426H
J4-12	A16-32	1884
J4-13	A16-29	6258
J4-14	A16-30	36F4
J4-15	A16-27	H063
J4-16	A16-28	8HA0
J4-17	A16-25	F860
J4-18	A16-26	3190
J4-19	A16-23	2UH2
J4-20	A16-24	305U
J4-21	A16-21	A7A2
J4-22	A16-22	P94U
J5-1	A17-2	U8UH
J5-2	A17-5	1UU1
J5-3	A17-39	7F8U
J5-4	A17-1	7PU9
J5-5	A17-37	UC17
J5-6	A17-38	47U6
J5-7	A17-35	7215
J5-8	A17-36	7PP4
J5-9	A17-33	5FF7
J5-10	A17-34	OAC9
J5-11	A17-31	HU7H
J5-12	A17-32	63CP
J5-13	A17-29	5C31
J5-14	A17-30	CPC6
J5-15	A17-27	9463
J5-16	A17-28	1128
J5-17	A17-25	F962
J5-18	A17-26	3192
J5-19	A17-23	8A80
J5-20	A17-24	C115
J5-21	A17-21	189F
J5-22	A17-22	4031

If no errors then go to (15).

(15) Change the signature analyser connections to the following:

CLOCK (⌐↓) to SLØ (A13-32)

START (¬↓) to SL3 (A13-35)

STOP (¬↓) to SL3 (A13-35)

Pin	Signal Name	Signature(s)
A13-40**	V_{CC}	00UP/00UP
A13-35**	SL3	000U/000U
A13-34**	SL2	0033/0033
A13-33**	SL1	0055/0055
A13-32**	SLØ	0000/00UP

If any errors then replace A13*

else go to (16).

(16)

Pin	Signal Name	Signature(s)
A12-1	SL2	0033
A12-3	SL1	0055
A12-13**	SLØ	0000/00UP
A12-15	SL2	0033

If any errors then there is an open circuit on line(s) from A13 to A12

else go to (17).

(17) To perform this test on the open collector outputs of A12, it is necessary to connect a 10K pullup resistor from the signature analyser data probe to V_{CC}.

Pin	Signal Name	Signature(s)
A12-4**	1Y3	OOUP/OOPU
A12-5**	1Y2	OOPU/OOUP
A12-6**	1Y1	OOUP/OOHF
A12-7**	1YØ	OOHF/OOUP
A12-9**	2YØ	0077/OOUP
A12-10**	2Y1	OOUP/0077
A12-11**	2Y2	OOCA/OOUP
A12-12**	2Y3	OOUP/OOCA

If any errors then replace A12*

else go to (18).

(18) KEYBOARD TEST

Verify that the following signatures are obtained at the Return Line inputs to A13 with the specified keys pressed.

In the tables below '-' denotes the V_{cc} signature (OOUP).

A. With CLOCK (⊓) connected to SLØ (A13-32)

Key Pressed	A13 Input Pin							
	A13-38	A13-39	A13-1	A13-2	A13-5	A13-6	A13-7	A13-8
None	-	-	-	-	-	-	-	-
'0'	0077	-	-	-	-	-	-	-
'1'	-	0077	-	-	-	-	-	-
'2'	-	-	0077	-	-	-	-	-
'3'	-	-	-	0077	-	-	-	-
'4'	-	-	-	-	0077	-	-	-
'5'	-	-	-	-	-	0077	-	-
'6'	-	-	-	-	-	-	0077	-
'7'	-	-	-	-	-	-	-	0077
'EXEC'	OOCA	-	-	-	-	-	-	-
'NEXT'	-	OOCA	-	-	-	-	-	-
'GO'	-	-	OOCA	-	-	-	-	-
'SUBST MEM'	-	-	-	OOCA	-	-	-	-
'EXAM REG'	-	-	-	-	OOCA	-	-	-
'SINGLE STEP'	-	-	-	-	-	OOCA	-	-

B. With CLOCK (↘) connected to SLØ (A13-32)

Key Pressed	A13 Input Pin							
	A13-38	A13-39	A13-1	A13-2	A13-5	A13-6	A13-7	A13-8
'8'	0077	-	-	-	-	-	-	-
'9'	-	0077	-	-	-	-	-	-
'A'	-	-	0077	-	-	-	-	-
'B'	-	-	-	0077	-	-	-	-
'C'	-	-	-	-	0077	-	-	-
'D'	-	-	-	-	-	0077	-	-
'E'	-	-	-	-	-	-	0077	-
'F'	-	-	-	-	-	-	-	0077

If any errors then there is a fault in the interconnection from A12 to the keyboard, the keyboard itself OR from the keyboard to A13. The appropriate line may be traced with an ohm-meter

else go to (19).

(19) Examine the display, which should show

8. 8. 8. 8. 8. 8.

with all segments of approximately equal brilliance.

A. If one segment of one display only does not light then that display is faulty OR the connection to that segment is bad.

B. If all segments of a display are off then there is a fault in the corresponding digit driver circuit (Q11 - Q15 and associated resistors).

C. If one segment is off in each display then there is
either a fault in the segment driver circuit (Q3 - Q4 and associated resistors)
or the segment output from A13 is stuck high.
The cause of the fault can be isolated using a CRO.

D. If all segments in one digit are brighter than the others then the corresponding digit driver is stuck on - isolate the fault with a CRO.

E. If one segment in each display is brighter than the others then
either the corresponding segment drive is stuck on
or the segment output from A13 is stuck low.

F. If two segments in each digit are brighter than the others then there is a short circuit between the two segment drivers or between two segment outputs of A13.

G. If all segments of two digits are brighter than the others then there is a short circuit between the two digit drivers.

H. If any other (irregular) pattern is observed then replace A13.

Stage III

(1) INITIALISATION

Insert the BLUE plug in the address selection socket.

Insert the 25 pin CANNON test plug in the serial I/O socket (J7).

Insert test connector P3 into J3.

Insert test connector P4 into J4.

Insert test connector P5 into J5.

(2) Apply power to the system and press RESET.

The display should go blank and the character sequence

0 1 2 3 4 5 6 7 8 9 A B C D E F H L P U r t . <blank>

should shift from left to right across the display continuously.

If any errors then replace A13.

(3) Press 'NEXT' key.

Display	Procedure
L	Go to (4)
Err 11	If A13-4 is high then replace A13* else if A11-8 is low then replace A11 else there is a bad connection from A13-4 to A11-8.
Err 12	If A11-8 is high then replace A11 else if A13-4 is low then replace A13* else there is a bad connection from A13-4 to A11-8.
Err 13	Replace A11.
Err 14	Press RESET and repeat the test. If the same error occurs then replace A13.
Any other display	Replace A13.

(4) Press each key in the sequence given below and verify that the correct character is displayed.

Key	Display
'Ø'	0
'1'	1
'2'	2
'3'	3
'4'	4
'5'	5
'6'	6
'7'	7
'8'	8
'9'	9
'A'	A
'B'	B
'C'	C
'D'	D
'E'	E
'F'	F
'SINGLE STEP'	t
'GO'	p
'SUBST MEM'	u
'EXAM REG'	r
'EXEC'	H

If any errors then replace A13.

(5) Press 'NEXT' key

Display	Procedure
.....5	Go to (6).
Err 21	Serial I/O error. The processor will loop, producing a 1KHz square wave at SOD (A11-4). If the square wave is not present at A11-4 then replace A11* else if the square wave is present at A11-5 then replace A11 else there is a fault in the interface circuit - trace with a CRO and fix the fault. To proceed with tests press 'NEXT' and go to (6).
Err 3X Err 4X	X is any hex digit. Parallel I/O / timer error. The second hex digit (X) indicates which device(s) failed the parallel I/O test with a bit set for each device which failed. Bit Ø set - replace A16 Bit 1 set - replace A17 Bit 2 set - replace A15 Bit 3 set - replace A14. If no device failed the I/O test the second digit will be 'Ø'. If all failed it will be 'F'. If the first digit is 4, a TRAP error is indicated as well. The processor will loop repeating the I/O and trap tests, producing a low pulse of duration 43µs at A16-6 every 2.4ms. If pulses do not appear at A16-6 then replace A16* else if pulses appear at A11-6 then replace A11 else there is an open circuit from A16-6 to A11-6. In either case, to proceed to next test press 'NEXT' and the display should show .....5. Go to (6).

(6) Remove connectors P3, P4 and P5 from J3, J4 and J5.

Insert connector P1 into J1 and P2 into J3.

Insert connector P6 into J5.

Remove link 2Ø-21 from SDK-85 board.

Remove link 4-3 from SDK-85 board.

Remove link 7-8 from SDK-85 board.

WITHOUT REMOVING POWER OR PRESSING RESET.

Press 'EXEC' to execute the RST 6.5 test.

Display	Procedure
.... .6	Go to (7).
Err 5X	X is 0, 1, 2, 3, 4, 5, 6 or 7. X indicates the type of failure detected in the operation of RST 6.5 as follows: Bit Ø set - RST 6.5 input detected as high initially. Bit 1 set - RST 6.5 input detected as low after being set high. Bit 2 set - RST 6.5 interrupt did not occur Bit 3 is always clear. The processor will enter a loop repeating the RST 6.5 test which will produce a train of high pulses at A15-24 (155µs in 4.4ms) If there is no pulse train at A15-24 then replace A15* else if there is a pulse train at A11-8 then replace A11 else the signal from A15-24 is not propogating from A15-24 to A11-8. Trace the signal with a CRO to indicate the cause. Replace A5* if it appears at A5-12 but not A5-13. To proceed to the next test press 'NEXT'. The display should show .... .6. Go to (7).

(7) Press 'VECT INTR' key.

Display	Procedure
.... .7	Go to (8).
Err 62	An active transition was detected at the RST 7.5 input, but the interrupt failed. Replace A11.
.... .6	No active transition at the RST 7.5 input was detected. Press 'VECT INTR' and observe the logic state at A11-7. If A11-7 goes low when key is pressed then replace A11 else the key, or connection to A11-7 is faulty. To proceed to the next test press 'NEXT' and display shows 'Err 61'. Press 'NEXT' again and the display shows '.... .7' Go to (8).

(8) The 'INTR' test is optional, and need not be executed if the hardware to produce the interrupt vector is not present.

If the test is not to be executed then press 'NEXT' (display should show .... .8)

Go to (9).

else to execute the test press 'EXEC'.

Display	Procedure
.... .8	Go to (9).
Err 71	Interrupt failure. The processor will loop, repeating the interrupt test. A train of positive pulses 12μs wide at 67μs intervals should appear at A15-25 through connectors P1 and P2 to A11-10. A11 should produce a train of INTA/pulses ½μs wide at 67μs intervals at A11-11 through A3 and J1 to the external interrupt vector hardware. This should apply a RST 3 vector through J1, A4/A7, to A11. This path is too difficult to debug completely but the following procedure may be followed: A. If INTA/pulses are produced at A11-11 then go to C. B. If INTR pulses are present at A11-10 then replace A11* else if no pulses at A15-25 then replace A15* else there is an open circuit from A15-25 to A11-10. C. If INTA/pulses are present at A3-12 then go to D else there is an open circuit from A11-11 to A3-12. D. If INTA/pulses are present at J1-16 then go to E else if no pulses present at A3-13 then replace A13* else there is an open circuit from A3-13 to J1-16. E. Replace A11 and repeat the test(s). If the same error occurs then replace A4 and A7. To continue to the next test press 'NEXT' and the display will show '.... .8' Go to (9).

(9) Test 8 is a test for the time on the optimal expansion RAM chip A17.

If the test is not to be executed then press 'NEXT' and the display will show 9

Go to (10)

else press 'EXEC'.

Display	Procedure
.... .9	Go to (10).
Err 81	Timer output was read in as initially low. The processor will loop repeating the test every 1ms, which should produce a low pulse of duration 40μs every 1ms at A17-6. If no pulse train present at A17-6 then replace A17* else if pulse train is present at A17-28 then replace A17 else there is an open circuit from A17-6 to A17-28.
Err 82	Timer output remained high for too long after the timer was started. Proceed as for Err 81.
Err 83	Timer output remained low for too long after the high to low transition. Proceed as for Err 81. To proceed to next test press 'NEXT' and the display will show .... .9. Go to (10).

(10) While the display shows .... .9 the processor is looping, writing to and reading from each address in the range 8000H to FFFFH.

The processor writes 00H to each location, then reads it back for each of the 32K locations. The process is then repeated, writing and reading FFH.

The test is designed to exercise any memory entered to the SDK-85 board, as well as the read logic associated with buffers A4 and A7.

A start pulse at A10-7 and a stop pulse at A10-9 are provided (clocked on RD/ or WR/) to enable the use of a signature analyser to verify the data read or written.

The test procedure will depend on the logic attached to the board, so no specific procedure is given.

To proceed to the next test press 'NEXT'.

Go to (11).

(11)

Display	Procedure
.... ..	The final test has succeeded and the processor has entered a permanent HOLD state. (See note (1) below.) Testing of the SDK-85 is complete.
Err HH	The attempt to place the processor in a hold state has failed and the processor is now halted. A. If A11-39 is high then replace A11 else to to B. B. If A3-4 is high then there is an open circuit from A3-4 to A11-39 else go to C. C. If A3-3 is high then replace A3* else go to D. D. If A15-26 is low then replace A15* else there is an open circuit from A15-16 to A3-4.
	Note (1): With the processor in the hold state it is possible to verify the hold acknowledge logic as follows: A. If A11-33 is high then go to B else replace A11*. B. If A3-9 is high then go to C else there is an open circuit from A11-39 to A3-9. C. If A9-2 is high then go to D else open circuit from A11-39 to A9-2. D. If J1-12 is high then go to F else go to E. E. If A3-10 is low then replace A3* else open circuit from A3-10 to J1-12. F. If A9-5 is high then go to G else replace A9*. G. Check that A1-1, A2-1, A4-1, A6-1 and A7-1 are all high. If any pin is not, there is an open circuit from A9-5 to that pin.

APPENDIX G

LISTING OF THE 8085 FUNCTIONAL TEST PROGRAM

(SLFTST.V8)

```
asm80 :f1:slftst.v8 mod85 macrofile pagewidth(110)


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    1

  LOC  OBJ        LINE       SOURCE STATEMENT

                     1 ;*******************************************************************************
                     2 ;
                     3 $       TITLE   ('8085 Self-test Program V8')
                     4 ;
                     5 ;*******************************************************************************
                     6 ;
                     7 ;                          8085 SELF TEST PROGRAM
                     8 ;                          ======================
                     9 ;
                    10 ;                            COPYRIGHT (C) 1980
                    11 ;
                    12 ;                              M. J. LIEBELT
                    13 ;                         ELECTRICAL ENGINEERING DEPT.
                    14 ;                           UNIVERSITY OF ADELAIDE
                    15 ;                                11/07/80
                    16 ;
                    17 ;
                    18 ;*******************************************************************************
                    19 ;
                    20 ;
                    21 ;   SLFTST IS A SELF'TEST PROGRAM FOR THE 8085 MICROPROCESSOR, INTENDED TO
                    22 ; VERIFY THE INTEGRITY OF THE CHIP BY EXERCISING ITS FUNCTIONAL UNITS (ALU,
                    23 ; ACCUMULATOR, REGISTER ARRAY, ETC.).  THE PROGRAM IS DIVIDED INTO A SERIES
                    24 ; OF TESTS, EACH DESIGNED TO TEST THE OPERATION OF ONE FUNCIONAL UNIT, OR A
                    25 ; SMALL GROUP OF RELATED FUNCTIONAL UNITS.  THE OPERATION OF THE FUNCTIONAL
                    26 ; UNITS IS VERIFIED BY PLACING DATA (WHICH DEPENDS ON CORRECT OPERATION OF TH
                    27 ; FUNCTIONAL UNIT) ONTO THE DATA AND ADDRESS BUSSES.  THIS DATA IS OBSERVED
                    28 ; USING A SIGNATURE ANALYSER CLOCKED ON RD/, WR/ OR ALE.
                    29 ;
                    30 ;   THE TEST IS TO BE EXECUTED IMMEDIATELY AFTER STAGE I OF THE SDK-85 SIGNAT
                    31 ; URE ANALYSIS ROUTINE.  IT IS EXECUTED ON RESET WHEN THE GREEN PLUG IS INSER
                    32 ; INTO THE ADDRESS SELECTION SOCKET ON THE SDK-85 BOARD, WHICH INTERCHANGES
                    33 ; THE CS0/ AND CS1/ LINES TO DEVICES A14 AND A15.  IT ALSO TIES ADDRESS LINES
                    34 ; A9 AND A10 ON THE EXPANSION ROM (A15) HIGH, WHICH CAUSES THE PROCESSOR TO
                    35 ; START EXECUTING AT LOCATION 600H IN A15.  THE FIRST FEW LOCATIONS FROM 600H
                    36 ; CONTAIN A JUMP TO THE EXTERNAL ROM AT 8000H, WHICH CONTAINS THE SELF TEST
                    37 ; PROGRAM PROPER.
                    38 ;   LOCATIONS 600H TO 63FH IN A15 ALSO CONTAIN A CHAIN OF 'RST' INSTRUCTIONS
                    39 ; USED IN THE COURSE OF EXECUTING THIS PROGRAM.
                    40 ;
                    41 ;
                    42
0030                43 CS6     EQU     30H
0038                44 CS7     EQU     38H
                    45
                    46 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    2
8085 Self-test Program V8

 LOC  OBJ        LINE        SOURCE STATEMENT

 0600             47          ORG     0600H          ;IN A15.
                  48
                  49
 0600 F3          50 START:   DI
 0601 C30280      51          JMP     SLFTST         ;JUMP TO MAIN PROGRAM.
                  52
                  53
 0608             54          ORG     START+08H      ;RST 1 LOCATION
 0608 C7          55          RST     0
                  56
 0610             57          ORG     START+10H      ;RST 2 LOCATION
 0610 CF          58          RST     1
                  59
 0618             60          ORG     START+18H      ;RST 3 LOCATION
 0618 D7          61          RST     2
                  62
 0620             63          ORG     START+20H      ;RST 4 LOCATION
 0620 DF          64          RST     3
                  65
 0628             66          ORG     START+28H      ;RST 5 LOCATION
 0628 E7          67          RST     4
                  68
 0630             69          ORG     START+30H      ;RST 6 LOCATION
 0630 EF          70          RST     5
                  71
 0638             72          ORG     START+38H      ;RST 7 LOCATION
 0638 F7          73          RST     6
                  74
                  75 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   3
8085 Self-test Program V8

 LOC  OBJ        LINE        SOURCE STATEMENT

 8001             76          ORG     8001H
 8001 40          77          DB      40H             ;DATA FOR ARITHMETIC INSTRUCTIONS
                  78                                  ; TEST.
                  79 ;
                  80 ;   START AND STOP PULSES FOR THE SIGNATURE ANALYSER ARE PROVIDED BY
                  81 ; EXECUTING IN AND OUT INSTRUCTIONS AT 38H (8205 CS7/) AND 30H (8205 CS6/)
                  82 ; FOR START AND STOP RESPECTIVELY.  EXTERNAL LOGIC IS REQUIRED TO DECODE
                  83 ; THE CHIP SELECT LINES WITH THE IO/M STATUS LINE TO PRODUCE THE START
                  84 ; AND STOP PULSES. BOTH IN AND OUT ARE EXECUTED TO ALLOW EITHER RD/ OR
                  85 ; WR/ TO BE USED FOR THE SIGNATURE ANALYSER CLOCK.
                  86 ;
 8002 DB38        87 SLFTST: IN       CS7             ;STOP PULSES.
 8004 D338        88         OUT      CS7             ;
 8006 DB30        89         IN       CS6             ;START PULSES.
 8008 D330        90         OUT      CS6             ;
                  91 ;
                  92 ;
                  93 ;   START OF THE SELF TEST PROGRAM PROPER.
                  94 ;
                  95 ;
                  96 ;           ACCUMULATOR TEST
                  97 ;           ----------------
                  98 ;
                  99 ;   THE ACCUMULATOR TEST STORES AND READS 256 BYTES OF DATA FROM A, WRITING
                 100 ; THE DATA READ FROM A TO THE DATA BUS BUFFERS FOR VERIFICATION.
                 101 ;
 800A 3E00       102         MVI     A,00H
 800C 3C         103 L1:     INR     A
 800D D300       104         OUT     00H
 800F C20C80     105         JNZ     L1
                 106 ;
                 107 ;         REGISTER ARRAY TEST
                 108 ;         -------------------
                 109 ;
                 110 ;   THE REGISTER ARRAY TEST STORES AND READS 256 BYTES OF DATA IN EACH OF THE
                 111 ; REGISTERS B,C,D,E,H & L.  THE DATA IS OBSERVED BY WRITING REGISTER PAIRS TO
                 112 ; THE ADDRESS BUS (MOV M,A ; STAX B ; STAX D).
                 113 ;
 8012 3E00       114         MVI     A,00H
 8014 0601       115         MVI     B,01H
 8016 0E02       116         MVI     C,02H
 8018 1603       117         MVI     D,03H
 801A 1E04       118         MVI     E,04H
 801C 2605       119         MVI     H,05H
 801E 2E06       120         MVI     L,06H
 8020 77         121 L2:     MOV     M,A
 8021 64         122         MOV     H,H
 8022 77         123         MOV     M,A
 8023 24         124         INR     H
 8024 3C         125         INR     A
 8025 C22080     126         JNZ     L2
 8028 77         127 L3:     MOV     M,A
 8029 6D         128         MOV     L,L
 802A 77         129         MOV     M,A
 802B 2C         130         INR     L
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    4
8085 Self-test Program V8

 LOC  OBJ        LINE       SOURCE STATEMENT

 802C 3C          131        INR     A
 802D C22880      132        JNZ     L3
 8030 12          133 L4:    STAX    D
 8031 52          134        MOV     D,D
 8032 12          135        STAX    D
 8033 14          136        INR     D
 8034 3C          137        INR     A
 8035 C23080      138        JNZ     L4
 8038 12          139 L5:    STAX    D
 8039 5B          140        MOV     E,E
 803A 12          141        STAX    D
 803B 1C          142        INR     E
 803C 3C          143        INR     A
 803D C23880      144        JNZ     L5
 8040 02          145 L6:    STAX    B
 8041 40          146        MOV     B,B
 8042 02          147        STAX    B
 8043 04          148        INR     B
 8044 3C          149        INR     A
 8045 C24080      150        JNZ     L6
 8048 02          151 L7:    STAX    B
 8049 49          152        MOV     C,C
 804A 02          153        STAX    B
 804B 0C          154        INR     C
 804C 3C          155        INR     A
 804D C24880      156        JNZ     L7
                  157 ;
                  158 ;           TMP TEST
                  159 ;           --------
                  160 ;
                  161 ;   THE TEMPORARY REGISTER TEST WRITES 256 BYTES OF DATA TO THE REGISTER
                  162 ; AND READS IT BACK ONTO THE INTERNAL DATA BUS, THEN THROUGH
                  163 ; THE DATA BUS BUFFERS TO THE EXTERNAL DATA BUS.
                  164 ;
 8050 0600        165        MVI     B,00H
 8052 25          166 L8:    DCR     H               ;CHANGE CONTENTS OF TMP.
 8053 70          167        MOV     M,B             ;B -> TMP -> DATA BUS.
 8054 04          168        INR     B
 8055 C25280      169        JNZ     L8
                  170 ;
                  171 ;       SP AND DECREMENTER TEST
                  172 ;       -----------------------
                  173 ;
                  174 ;   THIS TEST DECREMENTS THE STACK POINTER THROUGH ALL 65536 VALUES, AT
                  175 ; THE SAME TIME TESTING THE OPERATION OF THE DECREMENTER ON EACH OF THESE
                  176 ; VALUES.  STACK POINTER CONTENTS ARE TESTED BY PERFORMING A 'PUSH H'
                  177 ; OPERATION.
                  178 ;
 8058 310000      179        LXI     SP,0000H
 805B 3E00        180        MVI     A,00H
 805D 47          181        MOV     B,A
 805E E5          182 L9:    PUSH    H
 805F 2B          183        DCX     H
 8060 3C          184        INR     A
 8061 C25E80      185        JNZ     L9
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    5
8085 Self-test Program V8

 LOC  OBJ        LINE        SOURCE STATEMENT

 8064 04          186          INR     B
 8065 C25E80      187          JNZ     L9
                  188 ;
                  189 ;            ALU TEST
                  190 ;            --------
                  191 ;
                  192 ;   THE ALU IS TESTED BY EXECUTING EACH POSSIBLE ALU OPERATION WITH ALL
                  193 ; POSSIBLE VALUES OF THE TWO INPUT OPERANDS (AND, WHERE RELEVANT, OF CY).
                  194 ; THE EFFECT OF EACH OPERATION ON A AND THE FLAGS IS EXAMINED BY EXECUTING
                  195 ; A 'PUSH PSW'.
                  196 ;
 8068 37          197          STC                     ;CHECK CARRY CAN BE SET.
 8069 F5          198          PUSH    PSW             ;
 806A 3F          199          CMC                     ;CHECK CARRY CAN BE RESET.
 806B F5          200          PUSH    PSW             ;
                  201
 806C 3E00        202          MVI     A,00H           ;INITIALISE FIRST OPERAND.
 806E 47          203          MOV     B,A             ;INITIALISE SECOND OPERAND.
 806F 4F          204 L10:     MOV     C,A             ;SAVE THE FIRST OPERAND IN C.
 8070 79          205 L10A:    MOV     A,C
 8071 80          206          ADD     B
 8072 F5          207          PUSH    PSW
 8073 CA7780      208          JZ      L10B            ;THESE INSTRUCTIONS ARE INCLUDED HERE TO TEST
 8076 00          209          NOP                     ; THAT WHEN THE ALU SETS FLAGS, THEY ARE
 8077 DA7B80      210 L10B:    JC      L10C            ; CORRECTLY TESTED BY CONDITIONAL JUMP
 807A 00          211          NOP                     ; INSTRUCTIONS.
 807B FA7F80      212 L10C:    JM      L10D            ;
 807E 00          213          NOP                     ;
 807F EA8380      214 L10D:    JPE     L10E            ;
 8082 00          215          NOP                     ;
 8083 79          216 L10E:    MOV     A,C
 8084 90          217          SUB     B
 8085 F5          218          PUSH    PSW
 8086 79          219          MOV     A,C
 8087 A0          220          ANA     B
 8088 F5          221          PUSH    PSW
 8089 79          222          MOV     A,C
 808A A8          223          XRA     B
 808B F5          224          PUSH    PSW
 808C 79          225          MOV     A,C
 808D B0          226          ORA     B
 808E F5          227          PUSH    PSW
 808F 79          228          MOV     A,C
 8090 B8          229          CMP     B
 8091 F5          230          PUSH    PSW
 8092 37          231          STC
 8093 79          232          MOV     A,C
 8094 88          233          ADC     B
 8095 F5          234          PUSH    PSW
 8096 37          235          STC
 8097 3F          236          CMC
 8098 79          237          MOV     A,C
 8099 88          238          ADC     B
 809A F5          239          PUSH    PSW
 809B 37          240          STC
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE  PAGE   6
8085 Self-test Program V8

  LOC  OBJ        LINE       SOURCE STATEMENT

  809C 79          241       MOV     A,C
  809D 98          242       SBB     B
  809E F5          243       PUSH    PSW
  809F 37          244       STC
  80A0 3F          245       CMC
  80A1 79          246       MOV     A,C
  80A2 98          247       SBB     B
  80A3 F5          248       PUSH    PSW
  80A4 04          249       INR     B               ;INCREMENT OPERAND #2.
  80A5 C27080      250       JNZ     L10A            ;REPEAT 2 OPERAND OPERATIONS FOR ALL 256
                   251                               ; VALUES OF OP. #2.
                   252
  80A8 37          253       STC                     ;SINGLE OPERAND INSTRUCTIONS.
  80A9 79          254       MOV     A,C
  80AA 17          255       RAL
  80AB F5          256       PUSH    PSW
  80AC 37          257       STC
  80AD 3F          258       CMC
  80AE 79          259       MOV     A,C
  80AF 17          260       RAL
  80B0 F5          261       PUSH    PSW
  80B1 37          262       STC
  80B2 79          263       MOV     A,C
  80B3 1F          264       RAR
  80B4 F5          265       PUSH    PSW
  80B5 37          266       STC
  80B6 3F          267       CMC
  80B7 79          268       MOV     A,C
  80B8 1F          269       RAR
  80B9 F5          270       PUSH    PSW
  80BA 37          271       STC
  80BB 79          272       MOV     A,C
  80BC 07          273       RLC
  80BD F5          274       PUSH    PSW
  80BE 37          275       STC
  80BF 3F          276       CMC
  80C0 79          277       MOV     A,C
  80C1 07          278       RLC
  80C2 F5          279       PUSH    PSW
  80C3 37          280       STC
  80C4 79          281       MOV     A,C
  80C5 0F          282       RRC
  80C6 F5          283       PUSH    PSW
  80C7 37          284       STC
  80C8 3F          285       CMC
  80C9 79          286       MOV     A,C
  80CA 0F          287       RRC
  80CB F5          288       PUSH    PSW
  80CC 79          289       MOV     A,C
  80CD 2F          290       CMA
  80CE F5          291       PUSH    PSW
  80CF 79          292       MOV     A,C
  80D0 3D          293       DCR     A
  80D1 F5          294       PUSH    PSW
  80D2 79          295       MOV     A,C
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    7
8085 Self-test Program V8

 LOC  OBJ        LINE       SOURCE STATEMENT

 80D3 3C          296        INR     A
 80D4 F5          297        PUSH    PSW
 80D5 C26F80      298        JNZ     L10
                  299 ;
                  300 ;          DAA TEST
                  301 ;          --------
                  302 ;
                  303 ;   THE DAA INSTRUCTION IS EXECUTED WITH ALL POSSIBLE VALUES OF A,CY AND AC.
                  304 ; THE ALU TEST WILL HAVE VERIFIED THAT CY AND AC ARE CORRECTLY SET AS THE
                  305 ; RESULT OF OTHER ALU OPERATIONS.
                  306 ;
 80D8 0E0F        307        MVI     C,0FH           ;DATA TO ENABLE AC SET.
 80DA 3E00        308        MVI     A,00H
 80DC 47          309 L11:   MOV     B,A
 80DD B7          310        ORA     A               ;CY=0, AC=0
 80DE 27          311        DAA
 80DF F5          312        PUSH    PSW
 80E0 B7          313        ORA     A               ;AC=0
 80E1 37          314        STC                     ;CY=1
 80E2 78          315        MOV     A,B
 80E3 27          316        DAA
 80E4 F5          317        PUSH    PSW
 80E5 B7          318        ORA     A               ;CY=0
 80E6 79          319        MOV     A,C
 80E7 3C          320        INR     A               ;AC=1
 80E8 78          321        MOV     A,B
 80E9 27          322        DAA
 80EA F5          323        PUSH    PSW
 80EB 79          324        MOV     A,C
 80EC 3C          325        INR     A               ;AC=1
 80ED 37          326        STC                     ;CY=1
 80EE 78          327        MOV     A,B
 80EF 27          328        DAA
 80F0 F5          329        PUSH    PSW
 80F1 78          330        MOV     A,B
 80F2 3C          331        INR     A
 80F3 C2DC80      332        JNZ     L11
                  333 ;
                  334 ;          WZ REGISTER TEST
                  335 ;          ----------------
                  336 ;
                  337 ;   A WALKING BIT TEST IS EXECUTED TO VERIFY THAT REGISTERS W & Z MAY
                  338 ; BE LOADED FROM THE DATA BUS.  EIGHT 'OUT' INSTRUCTIONS ARE ALSO EXECUTED
                  339 ; TO CHECK THAT W & Z ARE CORRECTLY LOADED IN PARALLEL WITH THE SAME DATA.
                  340 ; NOTE THAT THE 'LHLD' INSTRUCTIONS REFERENCE ROM LOCATIONS (IN A15) ONLY,
                  341 ; SO THE DATA READ IN IS PREDICTABLE.
                  342 ;
 80F6 2A0100      343        LHLD    0001H
 80F9 2A0200      344        LHLD    0002H
 80FC 2A0400      345        LHLD    0004H
 80FF 2A0800      346        LHLD    0008H
 8102 2A1000      347        LHLD    0010H
 8105 2A2000      348        LHLD    0020H
 8108 2A4000      349        LHLD    0040H
 810B 2A8000      350        LHLD    0080H
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    8
8085 Self-test Program V8

 LOC  OBJ        LINE       SOURCE STATEMENT

 810E 220001      351        SHLD    0100H
 8111 220002      352        SHLD    0200H
 8114 220004      353        SHLD    0400H
 8117 220008      354        SHLD    0800H
 811A 220010      355        SHLD    1000H
 811D 220020      356        SHLD    2000H
 8120 220040      357        SHLD    4000H
 8123 220080      358        SHLD    8000H
 8126 D301        359        OUT     01H
 8128 D302        360        OUT     02H
 812A D304        361        OUT     04H
 812C D308        362        OUT     08H
 812E D310        363        OUT     10H
 8130 D320        364        OUT     20H
 8132 D340        365        OUT     40H
 8134 D380        366        OUT     80H
                  367 ;
                  368 ;   INCREMENTER/DECREMENTER NO-OPERATION TEST
                  369 ;   ----------------------------------------
                  370 ;
                  371 ;   THE ABILITY OF THE INCREMENTER/DECREMENTER TO PASS DATA UNMODIFIED
                  372 ; FROM THE ADDRESS LATCH TO THE REGISTER ARRAY IS TESTED BY EXECUTING
                  373 ; 65536 'SPHL' INSTRUCTIONS.
                  374 ;
 8136 3E00        375        MVI     A,00H
 8138 47          376        MOV     B,A
 8139 F9          377 L12:   SPHL                    ;HL -> AL -> SP.
 813A 23          378        INX     H
 813B E5          379        PUSH    H               ;OBSERVE SP.
 813C 3C          380        INR     A
 813D C23981      381        JNZ     L12
 8140 04          382        INR     B
 8141 C23981      383        JNZ     L12
                  384 ;
                  385 ;   BC AND DE ACCESS FROM INCREMENTER/DECREMENTER
                  386 ;   ----------------------------------------
                  387 ;
                  388 ;   THE ABILITY OF THE INCREMENTER/DECREMENTER TO CORRECTLY STORE DATA
                  389 ; IN BC AND DE IS TESTED BY EXECUTING 65536 'INX B', 'STAX B' AND
                  390 ; 'DCX D', 'STAX D' OPERATIONS.
                  391 ;
 8144 010000      392        LXI     B,0000H         ;INITIALISE BC.
 8147 110000      393        LXI     D,0000H         ;INITIALISE DE.
 814A 3E00        394        MVI     A,00H           ;COUNTER.
 814C 2600        395        MVI     H,00H           ;   '
 814E 03          396 L12A:  INX     B               ;WRITE TO BC FROM INC./DEC.
 814F 1B          397        DCX     D               ;WRITE TO DE FROM INC./DEC.
 8150 02          398        STAX    B               ;PLACE (BC) ON ADDRESS BUS.
 8151 12          399        STAX    D               ;PLACE (DE) ON ADDRESS BUS.
 8152 3D          400        DCR     A               ;REPEAT 65536 TIMES.
 8153 C24E81      401        JNZ     L12A            ;
 8156 25          402        DCR     H               ;
 8157 C24E81      403        JNZ     L12A            ;
                  404 ;
                  405 ;   READING SP THROUGH MUX
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    9
8085 Self-test Program V8

 LOC  OBJ        LINE       SOURCE STATEMENT

                  406 ;    ----------------------
                  407 ;
                  408 ;   THE ABILITY OF THE SP TO BE READ ONTO THE INTERNAL DATA BUS IS
                  409 ; TESTED BY THE EXECUTION OF 65536 'DAD SP' INSTRUCTIONS, THE RESULTS
                  410 ; OF WHICH ARE EXAMINED BY WRITING HL ONTO THE ADDRESS BUS ('MOV M,A').
                  411 ;
 815A 3E00        412         MVI     A,00H
 815C 47          413         MOV     B,A
 815D 210000      414 L13:    LXI     H,0000H
 8160 39          415         DAD     SP
 8161 77          416         MOV     M,A
 8162 3B          417         DCX     SP
 8163 3C          418         INR     A
 8164 C25D81      419         JNZ     L13
 8167 04          420         INR     B
 8168 C25D81      421         JNZ     L13
                  422 ;
                  423 ;   WRITING SP THROUGH MUX
                  424 ;   ----------------------
                  425 ;
                  426 ;   THE ABILITY OF THE SP TO BE LOADED FROM THE DATA BUS IS TESTED BY
                  427 ; A WALKING BIT TEST.  THE SP CONTENTS ARE EXAMINED BY EXECUTION OF A
                  428 ; 'PUSH' INSTRUCTION.
                  429 ;
 816B 310100      430         LXI     SP,0001H
 816E E5          431         PUSH    H
 816F 310200      432         LXI     SP,0002H
 8172 E5          433         PUSH    H
 8173 310400      434         LXI     SP,0004H
 8176 E5          435         PUSH    H
 8177 310800      436         LXI     SP,0008H
 817A E5          437         PUSH    H
 817B 311000      438         LXI     SP,0010H
 817E E5          439         PUSH    H
 817F 312000      440         LXI     SP,0020H
 8182 E5          441         PUSH    H
 8183 314000      442         LXI     SP,0040H
 8186 E5          443         PUSH    H
 8187 318000      444         LXI     SP,0080H
 818A E5          445         PUSH    H
 818B 310001      446         LXI     SP,0100H
 818E E5          447         PUSH    H
 818F 310002      448         LXI     SP,0200H
 8192 E5          449         PUSH    H
 8193 310004      450         LXI     SP,0400H
 8196 E5          451         PUSH    H
 8197 310008      452         LXI     SP,0800H
 819A E5          453         PUSH    H
 819B 310010      454         LXI     SP,1000H
 819E E5          455         PUSH    H
 819F 310020      456         LXI     SP,2000H
 81A2 E5          457         PUSH    H
 81A3 310040      458         LXI     SP,4000H
 81A6 E5          459         PUSH    H
 81A7 310080      460         LXI     SP,8000H
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   10
8085 Self-test Program V8

 LOC  OBJ        LINE        SOURCE STATEMENT

 81AA E5          461          PUSH    H
                  462 ;
                  463 ;    FLAG/JUMP/CALL/RETURN TEST
                  464 ;    --------------------------
                  465 ;
                  466 ;   THE FLAGS AND FLAG TESTING MECHANISM ARE TESTED BY LOADING THE FLAG
                  467 ; REGISTER FROM A TABLE (WALKING 1'S) AND TESTING THE RESPONSE OF ALL
                  468 ; AVAILABLE JUMP, CALL AND RETURN INSTRUCTIONS, AS INDICATED BY THE
                  469 ; SEQUENCE OF INSTRUCTION FETCH ADDRESSES PLACED ON THE ADDRESS BUS.
                  470 ;
 81AB 217883      471          LXI     H,FLGTAB              ;TABLE OF PSW'S TO POP.
 81AE 0605        472          MVI     B,(TABEND-FLGTAB)/2   ;NO. OF TESTS.
 81B0 F9          473 L14:     SPHL                          ;RESET SP TO NEXT TABLE ENTRY.
 81B1 23          474          INX     H                     ;UPDATE TABLE POINTER.
 81B2 23          475          INX     H                     ;
 81B3 F1          476          POP     PSW                   ;LOAD THE FLAGS.
 81B4 F5          477          PUSH    PSW                   ;CHECK THAT DATA IS READ BACK FROM TH
                  478                                        ; FLAGS CORRECTLY.
 81B5 CAB981      479          JZ      L15
 81B8 00          480          NOP
 81B9 C2BD81      481 L15:     JNZ     L16
 81BC 00          482          NOP
 81BD DAC181      483 L16:     JC      L17
 81C0 00          484          NOP
 81C1 D2C581      485 L17:     JNC     L18
 81C4 00          486          NOP
 81C5 FAC981      487 L18:     JM      L19
 81C8 00          488          NOP
 81C9 F2CD81      489 L19:     JP      L20
 81CC 00          490          NOP
 81CD EAD181      491 L20:     JPE     L21
 81D0 00          492          NOP
 81D1 E2D581      493 L21:     JPO     L22
 81D4 00          494          NOP
 81D5 C3D981      495 L22:     JMP     L23
 81D8 00          496          NOP
 81D9 310000      497 L23:     LXI     SP,0000H
 81DC CCE081      498          CZ      L24
 81DF 00          499          NOP
 81E0 C4E481      500 L24:     CNZ     L25
 81E3 00          501          NOP
 81E4 DCE881      502 L25:     CC      L26
 81E7 00          503          NOP
 81E8 D4EC81      504 L26:     CNC     L27
 81EB 00          505          NOP
 81EC FCF081      506 L27:     CM      L28
 81EF 00          507          NOP
 81F0 F4F481      508 L28:     CP      L29
 81F3 00          509          NOP
 81F4 ECF881      510 L29:     CPE     L30
 81F7 00          511          NOP
 81F8 E4FC81      512 L30:     CPO     L31
 81FB 00          513          NOP
 81FC CD0082      514 L31:     CALL    L32
 81FF 00          515          NOP
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

8200 318A83      516 L32:   LXI     SP,RETTAB
8203 C8          517        RZ
8204 33          518        INX     SP
8205 33          519        INX     SP
8206 C0          520 L33:   RNZ
8207 33          521        INX     SP
8208 33          522        INX     SP
8209 D8          523 L34:   RC
820A 33          524        INX     SP
820B 33          525        INX     SP
820C D0          526 L35:   RNC
820D 33          527        INX     SP
820E 33          528        INX     SP
820F F8          529 L36:   RM
8210 33          530        INX     SP
8211 33          531        INX     SP
8212 F0          532 L37:   RP
8213 33          533        INX     SP
8214 33          534        INX     SP
8215 E8          535 L38:   RPE
8216 33          536        INX     SP
8217 33          537        INX     SP
8218 E0          538 L39:   RPO
8219 33          539        INX     SP
821A 33          540        INX     SP
821B C9          541 L40:   RET
821C 33          542        INX     SP
821D 33          543        INX     SP
821E 05          544 L41:   DCR     B
821F C2B091      545        JNZ     L14
                 546 ;
                 547 ;   NOW TEST THAT THE FLAGS, WHEN WRITTEN FROM THE DATA BUS (POP PSW), ARE
                 548 ; CORRECTLY READ BY THE ALU (DAA).  CY AND AC ARE THE ONLY FLAGS CONCERNED,
                 549 ; AND WHEN THE DAA INSTRUCTION IS EXECUTED THE ACCUMULATOR WILL BE SET TO
                 550 ; 00, 06, 60 OR 66, DEPENDING ON THE VALUES OF CY AND AC.
                 551 ;
8222 318283      552        LXI     SP,FLGTB2              ;ADDRESS OF DATA TABLE.
8225 0604        553        MVI     B,(TB2END-FLGTB2)/2    ;NO. OF DATA BYTE PAIRS.
8227 F1          554 L41A:  POP     PSW                    ;GET NEXT FLAG DATA FROM TABLE.
8228 27          555        DAA                            ;READ CY AND AC.
8229 F5          556        PUSH    PSW                    ;EXAMINE THE RESULT.
822A 33          557        INX     SP                     ;SP POINTS TO NEXT PAIR OF BYTES IN T
822B 33          558        INX     SP                     ; DATA TABLE.
822C 05          559        DCR     B                      ;TEST FOR END OF TABLE.
822D C22782      560        JNZ     L41A                   ;
                 561 ;
                 562 ;   INTERRUPT MASK AND ENABLE FLAG TEST
                 563 ;   ----------------------------------
                 564 ;
                 565 ;   THE INTERRUPT CONTROL CIRCUITS ARE TESTED AS FOLLOWS:
                 566 ;     -ALL 8 POSSIBLE INTERRUPT MASKS ARE WRITTEN (WITH THE WRITE ENABLE BIT
                 567 ;      SET) AND READ BACK;
                 568 ;     -AN ATTEMPT IS MADE TO WRITE TO THE MASK REGISTER WITHOUT THE ENABLE BI
                 569 ;      SET AND A CHECK IS MADE TO SEE THAT IT IS NOT MODIFIED;
                 570 ;     -THE INTERRUPT ENABLE BIT IS TESTED TO SEE THAT IT CAN BE SET, RESET AN
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

                  571 ;      CORRECTLY READ.
                  572 ;
                  573 ;   INTERRUPT PROCESSING IS NOT TESTED HERE AS IT REQUIRES THE USE OF EXTERNA
                  574 ; (STILL UNTESTED) HARDWARE.
                  575 ;
 8230 F3          576        DI
 8231 3E08        577        MVI     A,08H
 8233 30          578 L42:   SIM
 8234 3C          579        INR     A
 8235 47          580        MOV     B,A
 8236 20          581        RIM                     ;CHECK IE AND MASKS
 8237 D300        582        OUT     00H
 8239 78          583        MOV     A,B             ;GET NEXT MASK.
 823A FE10        584        CPI     10H             ;CHECK FOR LAST.
 823C C23382      585        JNZ     L42
 823F AF          586        XRA     A               ;CLEAR A.
 8240 30          587        SIM                     ;ATTEMPT MASK CLEAR.
 8241 20          588        RIM                     ;CHECK NO EFFECT - ALL MASKS SHOULD BE SET.
 8242 D300        589        OUT     00H
 8244 FB          590        EI                      ;SET IE.
 8245 20          591        RIM
 8246 D300        592        OUT     00H
 8248 F3          593        DI
                  594 ;
                  595 ;   INSTRUCION DECODER TEST
                  596 ;   ----------------------
                  597 ;
                  598 ;   ALL INSTRUCTIONS WHICH HAVE NOT BEEN EXECUTED ALREADY
                  599 ; ARE NOW EXECUTED.
                  600 ;
                  601 ;
                  602 ;    (A) - ARITHMETIC INSTRUCTIONS.
                  603 ;
                  604 ;   TO TEST THE OPERATION OF THE ARITHMETIC INSTRUCTIONS, THE REGISTERS
                  605 ; ARE LOADED (FROM A TABLE) WITH DATA WHICH IS MEANINGFUL FOR THE PARTICULAR
                  606 ; OPERATION BEING TESTED (E.G. NONE OF THE REGISTERS IS LOADED WITH FFH FOR
                  607 ; THE 'OR' TEST).
                  608 ;
 8249 319C83      609        LXI     SP,DATA1
 824C F1          610        POP     PSW
 824D C1          611        POP     B
 824E D1          612        POP     D
 824F E1          613        POP     H
 8250 80          614        ADD     B
 8251 81          615        ADD     C
 8252 82          616        ADD     D
 8253 83          617        ADD     E
 8254 84          618        ADD     H
 8255 85          619        ADD     L
 8256 87          620        ADD     A
 8257 86          621        ADD     M
 8258 F5          622        PUSH    PSW
 8259 AF          623        XRA     A
 825A 88          624        ADC     B
 825B 89          625        ADC     C
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

 825C 8A          626        ADC     D
 825D 8B          627        ADC     E
 825E 8C          628        ADC     H
 825F 8D          629        ADC     L
 8260 8F          630        ADC     A
 8261 8E          631        ADC     M
 8262 F5          632        PUSH    PSW
 8263 3EFF        633        MVI     A,0FFH
 8265 A0          634        ANA     B
 8266 A1          635        ANA     C
 8267 A2          636        ANA     D
 8268 A3          637        ANA     E
 8269 A4          638        ANA     H
 826A A5          639        ANA     L
 826B A6          640        ANA     M
 826C A7          641        ANA     A
 826D F5          642        PUSH    PSW
 826E AF          643        XRA     A
 826F A8          644        XRA     B
 8270 A9          645        XRA     C
 8271 AA          646        XRA     D
 8272 AB          647        XRA     E
 8273 AC          648        XRA     H
 8274 AD          649        XRA     L
 8275 AE          650        XRA     M
 8276 F5          651        PUSH    PSW
 8277 31A483      652        LXI     SP,DATA2
 827A C1          653        POP     B
 827B D1          654        POP     D
 827C E1          655        POP     H
 827D 97          656        SUB     A
 827E 90          657        SUB     B
 827F 91          658        SUB     C
 8280 92          659        SUB     D
 8281 93          660        SUB     E
 8282 94          661        SUB     H
 8283 95          662        SUB     L
 8284 96          663        SUB     M
 8285 F5          664        PUSH    PSW
 8286 9F          665        SBB     A
 8287 98          666        SBB     B
 8288 99          667        SBB     C
 8289 9A          668        SBB     D
 828A 9B          669        SBB     E
 828B 9C          670        SBB     H
 828C 9D          671        SBB     L
 828D 9E          672        SBB     M
 828E F5          673        PUSH    PSW
 828F AF          674        XRA     A
 8290 B0          675        ORA     B
 8291 B1          676        ORA     C
 8292 B2          677        ORA     D
 8293 B3          678        ORA     E
 8294 B4          679        ORA     H
 8295 B5          680        ORA     L
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

8296 B6          681        ORA     M
8297 B7          682        ORA     A
8298 F5          683        PUSH    PSW
                 684
8299 78          685        MOV     A,B
829A BF          686        CMP     A
829B F5          687        PUSH    PSW
829C 79          688        MOV     A,C
829D B8          689        CMP     B
829E F5          690        PUSH    PSW
829F 7A          691        MOV     A,D
82A0 B9          692        CMP     C
82A1 F5          693        PUSH    PSW
82A2 7B          694        MOV     A,E
82A3 BA          695        CMP     D
82A4 F5          696        PUSH    PSW
82A5 7C          697        MOV     A,H
82A6 BB          698        CMP     E
82A7 F5          699        PUSH    PSW
82A8 7D          700        MOV     A,L
82A9 BC          701        CMP     H
82AA F5          702        PUSH    PSW
82AB 7E          703        MOV     A,M
82AC BD          704        CMP     L
82AD F5          705        PUSH    PSW
82AE 7F          706        MOV     A,A
82AF BE          707        CMP     M
82B0 F5          708        PUSH    PSW
                 709 ;
                 710 ;   (B) - IMMEDIATE INSTRUCTIONS.
                 711 ;
                 712 ;   THE OPPORTUNITY IS TAKEN HERE TO USE DATA WHICH HAS OTHERWISE NOT
                 713 ; BEEN READ IN FROM THE DATA BUS (UNUSED OPCODES).
                 714 ;
82B1 3E08        715        MVI     A,08H
82B3 C610        716        ADI     10H
82B5 D628        717        SUI     28H
82B7 CE38        718        ACI     38H
82B9 F6CB        719        ORI     0CBH
82BB DE18        720        SBI     18H
82BD EED9        721        XRI     0D9H
82BF E6ED        722        ANI     0EDH
82C1 FEDD        723        CPI     0DDH
82C3 F5          724        PUSH    PSW
                 725 ;
                 726 ;    (C) - MOVE INSTRUCTIONS.
                 727 ;
82C4 3AAB83      728        LDA     DATAFD          ;THIS DATA IS NOT READ ELSEWHERE.
82C7 3D          729        DCR     A
82C8 47          730        MOV     B,A
82C9 3D          731        DCR     A
82CA 4F          732        MOV     C,A
82CB 3D          733        DCR     A
82CC 57          734        MOV     D,A
82CD 3D          735        DCR     A
```

```
LOC  OBJ        LINE     SOURCE STATEMENT

82CE 5F          736      MOV    E,A
82CF 3D          737      DCR    A
82D0 67          738      MOV    H,A
82D1 3D          739      DCR    A
82D2 6F          740      MOV    L,A
82D3 3D          741      DCR    A
82D4 77          742      MOV    M,A
82D5 F5          743      PUSH   PSW
82D6 C5          744      PUSH   B
82D7 D5          745      PUSH   D
82D8 E5          746      PUSH   H
                 747
82D9 21AA83      748      LXI    H,DATA76      ;THIS DATA IS NOT READ ELSEWHERE (HLT OPCODE)
82DC 34          749      INR    M             ;THESE 2 INSTRUCTIONS ARE NOT EXECUTED ELSEWH
82DD 35          750      DCR    M             ; AND ARE CONVENIENTLY TESTED HERE.
82DE 46          751      MOV    B,M
82DF 05          752      DCR    B
82E0 48          753      MOV    C,B
82E1 0D          754      DCR    C
82E2 51          755      MOV    D,C
82E3 15          756      DCR    D
82E4 5A          757      MOV    E,D
82E5 1D          758      DCR    E
82E6 63          759      MOV    H,E
82E7 25          760      DCR    H
82E8 6C          761      MOV    L,H
82E9 2D          762      DCR    L
82EA 75          763      MOV    M,L
82EB C5          764      PUSH   B
82EC D5          765      PUSH   D
82ED E5          766      PUSH   H
                 767
82EE 21AA83      768      LXI    H,DATA76
82F1 4E          769      MOV    C,M
82F2 0D          770      DCR    C
82F3 59          771      MOV    E,C
82F4 1D          772      DCR    E
82F5 6B          773      MOV    L,E
82F6 2D          774      DCR    L
82F7 45          775      MOV    B,L
82F8 05          776      DCR    B
82F9 50          777      MOV    D,B
82FA 15          778      DCR    D
82FB 62          779      MOV    H,D
82FC 25          780      DCR    H
82FD 74          781      MOV    M,H
82FE C5          782      PUSH   B
82FF D5          783      PUSH   D
8300 E5          784      PUSH   H
                 785
8301 21AA83      786      LXI    H,DATA76
8304 56          787      MOV    D,M
8305 15          788      DCR    D
8306 6A          789      MOV    L,D
8307 2D          790      DCR    L
```

```
 LOC  OBJ          LINE       SOURCE STATEMENT

 8308 4D           791        MOV     C,L
 8309 0D           792        DCR     C
 830A 61           793        MOV     H,C
 830B 25           794        DCR     H
 830C 44           795        MOV     B,H
 830D 05           796        DCR     B
 830E 58           797        MOV     E,B
 830F 1D           798        DCR     E
 8310 73           799        MOV     M,E
 8311 C5           800        PUSH    B
 8312 D5           801        PUSH    D
 8313 E5           802        PUSH    H
                   803
 8314 21AA83       804        LXI     H,DATA76
 8317 5E           805        MOV     E,M
 8318 1D           806        DCR     E
 8319 43           807        MOV     B,E
 831A 05           808        DCR     B
 831B 60           809        MOV     H,B
 831C 25           810        DCR     H
 831D 4C           811        MOV     C,H
 831E 0D           812        DCR     C
 831F 69           813        MOV     L,C
 8320 2D           814        DCR     L
 8321 55           815        MOV     D,L
 8322 15           816        DCR     D
 8323 72           817        MOV     M,D
 8324 C5           818        PUSH    B
 8325 D5           819        PUSH    D
 8326 E5           820        PUSH    H
                   821
 8327 21AA83       822        LXI     H,DATA76
 832A 66           823        MOV     H,M
 832B 25           824        DCR     H
 832C 54           825        MOV     D,H
 832D 15           826        DCR     D
 832E 42           827        MOV     B,D
 832F 05           828        DCR     B
 8330 68           829        MOV     L,B
 8331 2D           830        DCR     L
 8332 5D           831        MOV     E,L
 8333 1D           832        DCR     E
 8334 4B           833        MOV     C,E
 8335 0D           834        DCR     C
 8336 71           835        MOV     M,C
 8337 C5           836        PUSH    B
 8338 D5           837        PUSH    D
 8339 E5           838        PUSH    H
                   839
 833A 21AA83       840        LXI     H,DATA76
 833D 6E           841        MOV     L,M
 833E 2D           842        DCR     L
 833F 65           843        MOV     H,L
 8340 25           844        DCR     H
 8341 5C           845        MOV     E,H
```

```
 LOC  OBJ        LINE        SOURCE STATEMENT

 8342 1D          846         DCR     E
 8343 53          847         MOV     D,E
 8344 15          848         DCR     D
 8345 4A          849         MOV     C,D
 8346 0D          850         DCR     C
 8347 41          851         MOV     B,C
 8348 05          852         DCR     B
 8349 70          853         MOV     M,B
 834A C5          854         PUSH    B
 834B D5          855         PUSH    D
 834C E5          856         PUSH    H
                  857
 834D 40          858         MOV     B,B
 834E 49          859         MOV     C,C
 834F 52          860         MOV     D,D
 8350 5B          861         MOV     E,E
 8351 64          862         MOV     H,H
 8352 6D          863         MOV     L,L
 8353 C5          864         PUSH    B
 8354 D5          865         PUSH    D
 8355 E5          866         PUSH    H
                  867
                  868 ;
                  869 ;   (D) - MISCELLANEOUS.
                  870 ;
 8356 019C83      871         LXI     B,DATA1
 8359 0A          872         LDAX    B
 835A 0B          873         DCX     B
 835B 02          874         STAX    B
 835C 11A483      875         LXI     D,DATA2
 835F 1A          876         LDAX    D
 8360 13          877         INX     D
 8361 12          878         STAX    D
 8362 320000      879         STA     0000H
 8365 217600      880         LXI     H,0076H
 8368 EB          881         XCHG
 8369 29          882         DAD     H
 836A 09          883         DAD     B
 836B 29          884         DAD     H
 836C 19          885         DAD     D
 836D 318A83      886         LXI     SP,RETTAB
 8370 E3          887         XTHL
 8371 E5          888         PUSH    H
                  889
 8372 217783      890         LXI     H,L43
 8375 E9          891         PCHL
 8376 00          892         NOP
 8377 FF          893 L43:    RST     7               ;BACK TO START VIA RESTARTS.
                  894
                  895 $EJECT
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE  18
8085 Self-test Program V8

 LOC  OBJ        LINE       SOURCE STATEMENT

                  896 ;
                  897 ;            DATA TABLES
                  898 ;            -----------
                  899 ;
 8378 01          900 FLGTAB: DB     01H,00H        ;DATA FOR FLAG TEST: CY=1.
 8379 00
 837A 04          901         DB     04H,00H        ;P=1.
 837B 00
 837C 10          902         DB     10H,00H        ;AC=1.
 837D 00
 837E 40          903         DB     40H,00H        ;Z=1.
 837F 00
 8380 80          904         DB     80H,00H        ;S=1.
 8381 00
                  905 TABEND:
                  906
 8382 00          907 FLGTB2: DB     00H,00H        ;CY=0,AC=0.
 8383 00
 8384 01          908         DB     01H,00H        ;CY=1,AC=0.
 8385 00
 8386 10          909         DB     10H,00H        ;CY=0,AC=1.
 8387 00
 8388 11          910         DB     11H,00H        ;CY=1,AC=1.
 8389 00
                  911 TB2END:
                  912
                  913
 838A 0682        914 RETTAB: DW     L33            ;RETURN ADDRESSES FOR CONDITIONAL RETURN TEST
 838C 0982        915         DW     L34
 838E 0C82        916         DW     L35
 8390 0F82        917         DW     L36
 8392 1282        918         DW     L37
 8394 1582        919         DW     L38
 8396 1882        920         DW     L39
 8398 1B82        921         DW     L40
 839A 1E82        922         DW     L41
                  923
                  924
 839C 0000        925 DATA1:  DW     0000H          ;DATA FOR ARITHMETIC INSTRUCTIONS TEST - PSW.
 839E F3F7        926         DW     0F7F3H         ;BC.
 83A0 F0F1        927         DW     0F1F0H         ;DE.
 83A2 FF83        928         DW     83FFH          ;HL - HL IS SET TO AN ADDRESS IN THIS ROM SO
                  929                               ;     THAT A FIXED VALUE CAN BE LOADED FROM
                  930                               ;     THE 'M REGISTER'.
                  931
 83A4 2010        932 DATA2:  DW     1020H          ;BC.
 83A6 0804        933         DW     0408H          ;DE.
 83A8 0180        934         DW     8001H          ;HL - SEE NOTE ABOVE.
                  935
 83AA 76          936 DATA76: DB     76H            ;DATA NOT USED ELSEWHERE.
 83AB FD          937 DATAFD: DB     0FDH           ;
                  938
 83FF             939         ORG    83FFH
 83FF E0          940         DB     0E0H           ;DATA FOR ARITHMETIC INSTRUCTIONS
                  941                               ; TEST.
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   19
8085 Self-test Program V8

  LOC  OBJ         LINE        SOURCE STATEMENT

                    942
                    943
                    944          END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
CS6     A 0030    CS7     A 0038    DATA1   A 839C    DATA2   A 83A4    DATA76 A 83AA    DATAFD A 83AB
FLGTAB  A 8378    FLGTB2  A 8382    L1      A 800C    L10     A 806F    L10A   A 8070    L10B   A 8077
L10C    A 807B    L10D    A 807F    L10E    A 8083    L11     A 80DC    L12    A 8139    L12A   A 814E
L13     A 815D    L14     A 81B0    L15     A 81B9    L16     A 81BD    L17    A 81C1    L18    A 81C5
L19     A 81C9    L2      A 8020    L20     A 81CD    L21     A 81D1    L22    A 81D5    L23    A 81D9
L24     A 81E0    L25     A 81E4    L26     A 81E8    L27     A 81EC    L28    A 81F0    L29    A 81F4
L3      A 8028    L30     A 81F8    L31     A 81FC    L32     A 8200    L33    A 8206    L34    A 8209
L35     A 820C    L36     A 820F    L37     A 8212    L38     A 8215    L39    A 8218    L4     A 8030
L40     A 821B    L41     A 821E    L41A    A 8227    L42     A 8233    L43    A 8377    L5     A 8038
L6      A 8040    L7      A 8048    L8      A 8052    L9      A 805E    RETTAB A 838A    SLFTST A 8002
START   A 0600    TABEND  A 8382    TB2END  A 838A

ASSEMBLY COMPLETE,  NO ERRORS
```

APPENDIX H

SIGNATURE SET FOR THE 8085A FUNCTIONAL TEST

Signature Set for 8085A Functional Test Program (SLFTST.V8)

Stage IA

(1) Insert the GREEN plug into the address selction socket.

Apply power to the system and press RESET.

(2) Verify the following signature sets:

A: With CLOCK (↓) = RD/ (A11-32)

START (↓) = IOCS6 (Refer to Fig. 5.3)

STOP (↓) = IOCS7 (Refer to Fig. 5.3)

Pin	Signal Name	Signature
A11-40	Vcc	5933
A11-12	AD0	47CA
A11-13	AD1	803F
A11-14	AD2	3463
A11-15	AD3	1314
A11-16	AD4	39UC
A11-17	AD5	HPA4
A11-18	AD6	AA1C
A11-19	AD7	5644

B: With CLOCK (↓) = WR/ (A11-31)

START (↓) = IOCS6

STOP (↓) = IOCS7

Pin	Signal Name	Signature
A11-40	Vcc	71AC
A11-12	AD0	P82U
A11-13	AD1	A200
A11-14	AD2	U1UF
A11-15	AD3	HAF9
A11-16	AD4	6216
A11-17	AD5	U1CU
A11-18	AD6	5459
A11-19	AD7	752H

C: With CLOCK (↓) = ALE (A11-30)

START (↓) = IOCS6

STOP (↑) = IOCS7

Pin	Signal Name	Signature
A11-40	Vcc	151H
A11-12	AD0	8PU3
A11-13	AD1	P78F
A11-14	AD2	H8AC
A11-15	AD3	F7PC
A11-16	AD4	2FU4
A11-17	AD5	CC8C
A11-18	AD6	A390
A11-19	AD7	AOH7
A11-21	A8	A3FO
A11-22	A9	869A
A11-23	A10	25AF
A11-24	A11	F882
A11-25	A12	406C
A11-26	A13	FHAF
A11-27	A14	H320
A11-28	A15	5FHF

APPENDIX I

LISTING OF THE 8279 FUNCTIONAL TEST PROGRAM

(KDCTST.V9)

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    1


 LOC  OBJ        LINE       SOURCE STATEMENT

                   1 ;*******************************************************************************
                   2 ;
                   3 $      TITLE   ('8279 Functional Test Program  V9')
                   4 ;
                   5 ;*******************************************************************************
                   6 ;
                   7 ;
                   8 ;                    8279 TEST PROGRAM
                   9 ;                    =================
                  10 ;
                  11 ;                    COPYRIGHT (C) 1980
                  12 ;
                  13 ;                      M. J. LIEBELT
                  14 ;                 ELECTRICAL ENGINEERING DEPT.
                  15 ;                   UNIVERSITY OF ADELAIDE
                  16 ;                        30/6/80
                  17 ;
                  18 ;   THIS PROGRAM IS A FUNCTIONAL TEST FOR THE 8279 KEYBOARD-DISPLAY
                  19 ; CONTROLLER ON THE SDK-85 BOARD.  THE PROGRAM IS INTENDED TO RUN AT THE
                  20 ; BEGINNING OF STAGE III OF THE SDK-85 SIGNATURE ANALYSIS PROCEDURE.
                  21 ;
                  22 ;
                  23 ;*******************************************************************************
                  24 ;
                  25 ;          EXTERNAL SYMBOL DEFINITIONS (AS DEFINED IN SDK85S.V34)
                  26 ;
1800              27 KDCD    EQU     1800H          ;8279 DATA REGISTER.
1900              28 KDCC    EQU     1900H          ;8279 CONTROL REGISTER.
2002              29 RSTFLG  EQU     2002H          ;'RST' INTERRUPT FLAG ADDRESS.
0728              30 CHRTAB  EQU     0728H          ;DISPLAY CHARACTER TABLE ADDRESS.
0098              31 TEST2   EQU     0098H          ;RE-ENTRY POINT TO SDK85S.V34.
0640              32 DELAY   EQU     0640H          ;1 MILLISECOND DELAY ROUTINE.
064D              33 DLY500  EQU     064DH          ;500    "       "     "
06EF              34 RDKBD   EQU     06EFH          ;KEYBOARD FIFO READ ROUTINE.
06FF              35 CLRKBD  EQU     06FFH          ;FIFO CLEAR ROUTINE.
06D8              36 CLRDSP  EQU     06D8H          ;DISPLAY CLEAR ROUTINE.
06E2              37 CONVRT  EQU     06E2H          ;KEY NUMBER TO DISPLAY CODE CONVERSION ROUTIN
069F              38 ERRDSP  EQU     069FH          ;ERROR MESSAGE DISPLAY ROUTINE.
0051              39 NEXT    EQU     51H            ;'NEXT' KEY CODE (RETURNED BY RDKBD).
                  40
                  41
                  42
8000              43         ORG     8000H          ;EXTERNAL 1K ROM.
                  44
8000 F3           45 TEST0:  DI
8001 210019       46         LXI     H,KDCC         ;8279 CONTROL REG. ADRS
8004 0601         47         MVI     B,01H          ;INITIAL ERROR NUMBER.
                  48 ;
                  49 ;
                  50 ;   INITIALISE THE 8279.
                  51 ;
8006 363F         52         MVI     M,3FH          ;CLOCK PRESCALER = 31.
8008 3608         53         MVI     M,08H          ;16 CHAR. LEFT ENTRY, ENCODED SCAN,
                  54                                ; TWO KEY LOCKOUT.
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

800A 3640         55        MVI     M,40H           ;READ FIFO.
                  56 ;
                  57 ;              DISPLAY TEST
                  58 ;              ------------
                  59 ;
800C 36C1         60        MVI     M,0C1H          ;CLEAR FIFO AND DISPLAY RAM (TO 0'S).
800E CD3F83       61        CALL    CLRDLY          ;WAIT 160us.
8011 7E           62        MOV     A,M             ;READ STATUS AND CHECK THAT DISPLAY
8012 B7           63        ORA     A               ; UNAVAILABLE IS NOT SET.
8013 FA1783       64        JM      ERR0            ;ERROR #01 IF IT IS.
8016 04           65        INR     B               ;(B) = 2.
8017 E60F         66        ANI     0FH             ;CHECK THAT FIFO IS EMPTY.
8019 C21783       67        JNZ     ERR0            ;ERROR #02 IF NOT.
801C 04           68        INR     B               ;(B) = 3.
801D 1E00         69        MVI     E,00H           ;(E) = # OF CHARS. IN FIFO.
801F 3A0018       70        LDA     KDCD            ;ATTEMPT TO READ A CHAR. AND CHECK
8022 7E           71        MOV     A,M             ; FOR UNDERRUN.
8023 E610         72        ANI     10H             ;MASK UNDERRUN FLAG.
8025 CA1783       73        JZ      ERR0            ;ERROR #03 IF NOT SET.
8028 04           74        INR     B               ;(B) = 4.
8029 CD3283       75        CALL    WTFRCH          ;WAIT FOR KEY ENTRY TO CONFIRM THAT
                  76                                ; ALL DISPLAY SEGMENTS ARE ON.
                  77
802C 36D8         78        MVI     M,0D8H          ;CLEAR DISPLAY RAM (TO 20H).
802E CD3F83       79        CALL    CLRDLY          ;WAIT 160us FOR CLEAR.
8031 7E           80        MOV     A,M             ;READ STAUS AND CHECK D.U. AGAIN.
8032 B7           81        ORA     A               ;
8033 FA1983       82        JM      ERROR           ;ERROR #04 IF D.U. STILL SET.
8036 04           83        INR     B               ;(B) = 5.
8037 E60F         84        ANI     0FH             ;CHECK THAT NO. OF CHARS. IN THE FIFO
8039 BB           85        CMP     E               ; IS STILL 1 (I.E. FIFO WAS NOT AFFECTED).
803A C21783       86        JNZ     ERR0            ;ERROR #05 IF NOT.
803D 04           87        INR     B               ;(B) = 6.
803E CD3283       88        CALL    WTFRCH          ;WAIT FOR KEY ENTRY TO CONFIRM THAT THE
                  89                                ; DISPLAY IS CORRECT.
                  90
8041 36DC         91        MVI     M,0DCH          ;CLEAR DISPLAY TO FF'S.
8043 CD3F83       92        CALL    CLRDLY          ;WAIT 160us.
8046 7E           93        MOV     A,M             ;READ STATUS AND CHECK D.U.
8047 B7           94        ORA     A               ;
8048 FA1983       95        JM      ERROR           ;ERROR #06 IF STILL SET.
804B 04           96        INR     B               ;(B) = 7.
804C E60F         97        ANI     0FH             ;CHECK NO. OF CHARS. IN THE FIFO HAS
804E BB           98        CMP     E               ; NOT CHANGED.
804F C21783       99        JNZ     ERR0            ;ERROR #07 IF IT HAS.
8052 04          100        INR     B               ;(B) = 8.
8053 CD3283      101        CALL    WTFRCH          ;WAIT FOR KEY ENTRY TO CONFIRM THAT THE
                 102                                ; DISPLAY IS BLANK.
                 103
                 104 ;
                 105 ;                  DISPLAY RAM TEST
                 106 ;
                 107 ;   THE DISPLAY RAM IS TESTED BY FIRST WRITING UNIQUE DATA INTO EACH OF THE
                 108 ; 16 LOCATIONS AND READING THE DATA BACK (USING THE AUTO-INCREMENT ADDRESS
                 109 ; MECHANISM, AND THEN BY WRITING (AND READING BACK) OFFH TO 00H IN EACH
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    3
8279 Functional Test Program  V9

 LOC  OBJ        LINE       SOURCE STATEMENT

                  110 ; LOCATION.  AT THE END OF THE TEST ALL DISPLAY SEGMENTS SHOULD BE ON.
                  111 ;
 8056 3670        112        MVI     M,70H          ;READ DISPLAY RAM, LOCATION 0, AUTO-INC.
 8058 3690        113        MVI     M,90H          ;WRITE TO DISPLAY RAM LOC. 0, AUTO-INC.
 805A 25          114        DCR     H              ;(HL) = KDCD
 805B 0E10        115        MVI     C,10H          ;NO. OF RAM LOCATIONS.
 805D 3E01        116        MVI     A,01H          ;INITIAL DATA.
 805F 57          117        MOV     D,A            ;SAVE IT IN D.
 8060 77          118 DRT1:  MOV     M,A            ;WRITE DATA TO KDCD.
 8061 3C          119        INR     A              ;UPDATE THE DATA.
 8062 0D          120        DCR     C              ;TEST FOR LAST LOCATION.
 8063 C26080      121        JNZ     DRT1
 8066 0E10        122        MVI     C,10H          ;RESET THE COUNTER AND READ BACK THE DATA.
 8068 7A          123        MOV     A,D            ;RESTORE THE INITIAL DATA - NOTE THAT THE
                  124                               ; AUTO-INC. MECHANISM WILL WRAP AROUND SO
                  125                               ; THE FIRST READ WILL BE FROM LOC. 0.
 8069 56          126 DRT2:  MOV     D,M            ;READ FROM KDCD.
 806A BA          127        CMP     D              ;COMPARE WITH EXPECTED DATA (IN A).
 806B C21783      128        JNZ     ERR0           ;ERROR #08 IF NOT THE SAME.
 806E 3C          129        INR     A              ;UPDATE EXPECTED DATA.
 806F 0D          130        DCR     C              ;TEST FOR LAST LOCATION.
 8070 C26980      131        JNZ     DRT2           ;
 8073 04          132        INR     B              ;(B) = 9.
                  133 ;
 8074 24          134        INR     H              ;(HL) = KDCC.
 8075 0E8F        135        MVI     C,8FH          ;COMMAND TO WRITE DISP. RAM, LOCATION 15,
                  136                               ; NO INCREMENT.
 8077 71          137 DRT3:  MOV     M,C            ;WRITE THE COMMAND TO KDCC.
 8078 25          138        DCR     H              ;(HL) = KDCD.
 8079 16FF        139        MVI     D,0FFH         ;INITIAL DATA FOR EACH LOCATION.
 807B 72          140 DRT4:  MOV     M,D            ;WRITE THE DATA TO KDCD.
 807C 7E          141        MOV     A,M            ;READ IT BACK.
 807D BA          142        CMP     D              ;CHECK THAT IT IS WHAT WAS WRITTEN.
 807E C21783      143        JNZ     ERR0           ;ERROR #09 IF NOT.
 8081 15          144        DCR     D              ;UPDATE THE DATA.
 8082 7A          145        MOV     A,D            ;
 8083 FEFF        146        CPI     -1             ;DON'T GO PAST 00H IN EACH LOCATION.
 8085 C27B80      147        JNZ     DRT4           ;
 8088 24          148        INR     H              ;(HL) = KDCC.
 8089 0D          149        DCR     C              ;UPDATE THE WRITE COMMAND FOR THE NEXT
 808A 79          150        MOV     A,C            ; LOWER LOCATION, BUT DON'T GO BELOW
 808B FE7F        151        CPI     7FH            ; LOCATION 0.
 808D C27780      152        JNZ     DRT3           ;
 8090 04          153        INR     B              ;(B) = A.
                  154 ;
                  155 ;   THE OPERATION OF THE BLANKING AND WRITE INHIBIT CONTROLS IS NOW TESTED.
                  156 ; NOTE THAT THE DISPLAY RAM SHOULD BE FILLED WITH 00'S, SO ALL DISPLAY
                  157 ; SEGMENTS SHOULD NOW BE ON.
                  158 ;
 8091 36A1        159        MVI     M,0A1H         ;BLANK BITS B0-B3 (SEGMENTS e,f,g,d,p,).
 8093 CD3283      160        CALL    WTFRCH         ;WAIT FOR KEY ENTRY TO CONFIRM THAT THE
                  161                               ; DISPLAY IS CORRECT.
 8096 36A2        162        MVI     M,0A2H         ;NOW BLANK A0-A3 (SEGMENTS a-d).
 8098 CD3283      163        CALL    WTFRCH         ;WAIT FOR KEY ENTRY.
                  164
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE    4
8279 Functional Test Program  V9

 LOC  OBJ        LINE      SOURCE STATEMENT

 809B 36A8        165        MVI     M,0A8H          ;DISABLE BLANKING, BUT INHIBIT WRITE TO
                  166                                ; A0-A3.
 809D 3690        167        MVI     M,90H           ;WRITE DISPLAY RAM LOC. 0, AUTO-INC.
 809F 0EFF        168        MVI     C,0FFH          ;DATA TO BE WRITTEN TO EACH LOCATION.
 80A1 CD4883      169        CALL    WRDRAM          ;WRITE DATA TO EACH LOCATION, BUT ONLY
                  170                                ; THE BOTTOM 4 BITS SHOULD BE AFFECTED.
 80A4 0E10        171        MVI     C,10H           ;READ THE DISPLAY RAM CONTENTS TO CHECK (WE A
 80A6 3A0018      172 WIT1:  LDA     KDCD            ; STILL READING FROM DISPLAY RAM, AUTO-INC.)
 80A9 FE0F        173        CPI     0FH             ;CHECK A0-A3=0, B0-B3=1.
 80AB C21783      174        JNZ     ERR0            ;ERROR #0A IF NOT.
 80AE 0D          175        DCR     C               ;
 80AF C2A680      176        JNZ     WIT1            ;
 80B2 04          177        INR     B               ;(B) = B.
                  178
 80B3 36A0        179        MVI     M,0A0H          ;DISABLE WRITE INHIBITS AND REFILL THE RAM
 80B5 0E00        180        MVI     C,00H           ; WITH 0'S TO TURN ON ALL SEGMENTS.
 80B7 CD4883      181        CALL    WRDRAM          ;
 80BA 36A4        182        MVI     M,0A4H          ;INHIBIT WRITE TO B0-B3, SO ONLY THE TOP 4 BI
 80BC 0EFF        183        MVI     C,0FFH          ; SHOULD BE SET TO 1.
 80BE CD4883      184        CALL    WRDRAM          ;
 80C1 0E10        185        MVI     C,10H           ;READ THE DISPLAY RAM CONTENTS TO CHECK.
 80C3 3A0018      186 WIT2:  LDA     KDCD            ;
 80C6 FEF0        187        CPI     0F0H            ;CHECK A0-A3=1, B0-B3=0.
 80C8 C21783      188        JNZ     ERR0            ;ERROR #0B IF NOT.
 80CB 0D          189        DCR     C               ;
 80CC C2C380      190        JNZ     WIT2            ;
 80CF 04          191        INR     B               ;(B) = C.
                  192
                  193 ;
                  194 ;   THE KEYBORD FIFO SHOULD NOW CONTAIN 5 CHARACTERS.  THE FIFO CLEAR
                  195 ; COMMAND IS TESTED TO SEE THAT IT:
                  196 ;           -CLEARS THE FIFO CHARACTER COUNTER, O/R AND U/R FLAGS;
                  197 ;           -DOES NOT AFFECT THE DISPLAY RAM.
                  198 ;
 80D0 36C2        199        MVI     M,0C2H          ;CLEAR FIFO COMMAND.
 80D2 7E          200        MOV     A,M             ;READ STATUS AND CHECK THAT ALL IS
 80D3 E63F        201        ANI     3FH             ; CLEAR.
 80D5 C21783      202        JNZ     ERR0            ;ERROR #0C IF NOT.
 80D8 04          203        INR     B               ;(B) = D.
 80D9 1E00        204        MVI     E,00H           ;NO. OF CHARS. IN FIFO.
                  205
 80DB 0E10        206        MVI     C,10H           ;READ DISPLAY RAM TO SEE THAT IT IS UNCHANGED
 80DD 3A0018      207 KCT1:  LDA     KDCD            ;
 80E0 FEF0        208        CPI     0F0H            ;
 80E2 C21783      209        JNZ     ERR0            ;ERROR #0D IF ANY CHANGE.
 80E5 0D          210        DCR     C               ;
 80E6 C2DD80      211        JNZ     KCT1            ;
 80E9 04          212        INR     B               ;(B) = E.
                  213 ;
                  214 ;   THE DISPLAY BUFFER OUTPUT LINES ARE NOW TESTED BY WALKING A 0 ACROSS
                  215 ; THE 8 SEGMENT OUTPUT LINES.  THIS IS DONE BY SHIFTING A 0 ACROSS EACH
                  216 ; DISPLAY LOCATION.  AT THE END OF THE TEST ONLY SEGMENT d OF EACH DIGIT
                  217 ; WILL BE ON.
                  218 ;
 80EA 36A0        219        MVI     M,0A0H          ;TURN OFF WRITE INHIBIT.
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

80EC CDD806      220        CALL    CLRDSP          ;CLEAR THE DISPLAY.
80EF 3EFE        221        MVI     A,0FEH          ;INITIAL DATA.
80F1 0E10        222 DOBT1: MVI     C,10H           ;LOCATION COUNTER.
80F3 320018      223 DOBT2: STA     KDCD            ;WRITE DATA TO KDCD (NOTE THAT WE'RE STILL
                 224                                ; WRITING TO DISPLAY RAM, AUTO-INCREMENT).
80F6 0D          225        DCR     C               ;ALL 16 LOCATIONS.
80F7 C2F380      226        JNZ     DOBT2           ;
80FA F5          227        PUSH    PSW             ;SAVE THE DATA IN A.
80FB CD4D06      228        CALL    DLY500          ;WAIT 1/2 SECOND.
80FE F1          229        POP     PSW             ;RETRIEVE THE DATA.
80FF 07          230        RLC                     ;SHIFT THE 0.
8100 DAF180      231        JC      DOBT1           ;UPDATE DISPLAY RAM IF NOT DONE 8 TIMES.
8103 CD3283      232        CALL    WTFRCH          ;WAIT FOR APPROVAL.
                 233 ;
                 234 ;   THE DIFFERENT DISPLAY MODES OF THE CONTROLLER ARE NOW TESTED IN THE
                 235 ; FOLLOWING ORDER:
                 236 ;      16 CHARACTER LEFT ENTRY (THE CURRENT MODE);
                 237 ;       8 CHARACTER LEFT ENTRY;
                 238 ;       8 CHARACTER RIGHT ENTRY;
                 239 ;      16 CHARACTER RIGHT ENTRY.
                 240 ;
                 241 ;   NOTE THAT ONLY 6 DIGITS ARE AVAILABLE AND THAT THE HARDWARE IS SO
                 242 ; ARRANGED THAT IN 16 CHARACTER MODES, DIGITS 0-7 AND 8-15 WILL BE
                 243 ; SUPERIMPOSED.
                 244 ;
8106 CDD806      245        CALL    CLRDSP          ;CLEAR THE DISPLAY.
8109 D5          246        PUSH    D               ;SAVE E (FIFO CHAR. COUNT).
810A 112307      247        LXI     D,CHRTAB        ;ADDRESS OF DISPLAY CHAR. TABLE
810D 0E10        248        MVI     C,10H           ;NO. OF CHARS. TO BE DISPLAYED.
810F 1A          249 DMT1:  LDAX    D               ;GET CHAR. FROM TABLE.
8110 320018      250        STA     KDCD            ;WRITE OUT TO KDCD.
8113 CD4D06      251        CALL    DLY500          ;WAIT 1/2 SEC. BEFORE DISPLAYING NEXT CHAR.
8116 13          252        INX     D               ;NEXT CHAR. IN TABLE.
8117 0D          253        DCR     C               ;TEST FOR 16 CHARS. WRITTEN.
8118 C20F81      254        JNZ     DMT1            ;FINAL DISPLAY: 0123 45 SUPERIMPOSED ON
                 255                                ;               89Ab Cd
811B D1          256        POP     D               ;RESTORE FIFO CHAR. COUNT TO E.
811C CD3283      257        CALL    WTFRCH          ;WAIT FOR KEY ENTRY TO CONFIRM THAT THE
                 258                                ; DISPLAY IS CORRECT.
                 259
811F CD2583      260        CALL    CLRALL          ;CLEAR DISPLAY AND FIFO.
8122 3600        261        MVI     M,00H           ;CHANGE DISPLAY MODE TO 8 CHAR. LEFT ENTRY
                 262                                ; (THIS CLEARS THE FIFO).
8124 112807      263        LXI     D,CHRTAB        ;ADDRESS OF DISPLAY CHAR. TABLE.
8127 0E10        264        MVI     C,10H           ;NO. OF CHARS. TO BE WRITTEN TO DISPLAY RAM.
8129 1A          265 DMT2:  LDAX    D               ;GET CHAR. FROM TABLE.
812A 320018      266        STA     KDCD            ;WRITE OUT TO KDCD.
812D CD4D06      267        CALL    DLY500          ;WAIT 1/2 SECOND BEFORE DISPLAYING NEXT CHAR.
8130 13          268        INX     D               ;POINT TO NEXT CHAR. IN TABLE.
8131 0D          269        DCR     C               ;TEST FOR 16 CHARS. DONE.
8132 C22981      270        JNZ     DMT2            ;
8135 1E00        271        MVI     E,00H           ;SET FIFO CHAR. COUNT TO 0.
8137 CD3283      272        CALL    WTFRCH          ;WAIT FOR CONFIRMATION OF CORRECT DISPLAY.
                 273
813A CD2583      274        CALL    CLRALL          ;CLEAR DISPLAY AND FIFO.
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

813D 3610        275        MVI     M,10H          ;8 CHAR.,. RIGHT ENTRY.
813F 3693        276        MVI     M,93H          ;WRITE TO DISPLAY RAM FROM LOC. 3
8141 0E10        277        MVI     C,10H          ;NO. OF CHARS. WRITTEN TO RAM (ONLY 8 ARE
                 278                               ; DISPLAYED).
8143 112807      279        LXI     D,CHRTAB       ;DISPLAY TABLE ADDRESS.
8146 1A          280 DMT3:  LDAX    D              ;
8147 320018      281        STA     KDCD           ;
814A CD4D06      282        CALL    DLY500         ;
814D 13          283        INX     D              ;
814E 0D          284        DCR     C              ;
814F C24681      285        JNZ     DMT3           ;FINAL DISPLAY SHOULD BE: dEF8 9A
8152 1E00        286        MVI     E,00H          ;SET FIFO CHAR. COUNT TO 0.
8154 CD3283      287        CALL    WTFRCH         ;WAIT FOR CONFIRMATION OF CORRECT DISPLAY.
                 288
8157 CD2583      289        CALL    CLRALL         ;CLEAR DISPLAY AND FIFO.
815A 3618        290        MVI     M,18H          ;16 CHAR. RIGHT ENTRY.
815C 3696        291        MVI     M,96H          ;START FROM RIGHTMOST DIGIT (LOC. 6).
815E 0E10        292        MVI     C,10H          ;NOTE THAT AFTER '7' APPEARS, ALL FOLLOWING
8160 112807      293        LXI     D,CHRTAB       ; CHARACTERS WILL BE SUPERIMPOSED ON 0-7.
8163 1A          294 DMT4:  LDAX    D              ;
8164 320018      295        STA     KDCD           ;
8167 CD4D06      296        CALL    DLY500         ;
816A 13          297        INX     D              ;
816B 0D          298        DCR     C              ;
816C C26381      299        JNZ     DMT4           ;FINAL DISPLAY: AbCd EF SUPERIMPOSED ON
                 300                               ;               2345 67
816F 1E00        301        MVI     E,00H          ;SET FIFO CHAR. COUNT TO 0.
8171 CD3283      302        CALL    WTFRCH         ;WAIT FOR CONFIMATION OF CORRECT DISPLAY.
                 303 ;
                 304 ;   THE DECODED SCAN MODE OF OPERATION IS NOW TESTED.  THE DISPLAY SHOULD SHO
                 305 ;                          ___C _b        WHERE _ REPRESENTS A BLANK
                 306 ;
                 307 ;   IT IS NOT POSSIBLE TO TEST THE KEYBOARD OPERATION IN DECODED SCAN MODE
                 308 ; BECAUSE THE HARDWARE ARRANGEMENT IS SUCH THAT NONE OF THE 3 ROWS OF KEYS
                 309 ; ON THE SDK-85 WOULD BE SELECTED BY ANY OF THE 4 SCAN LINES GOING LOW.
                 310 ;
8174 3619        311        MVI     M,19H          ;SELECT DECODED SCAN MODE.
8176 160A        312        MVI     D,10D          ;WAIT FOR 5 SECONDS.
8178 CD4D06      313 DST1:  CALL    DLY500         ;
817B 15          314        DCR     D              ;
817C C27881      315        JNZ     DST1           ;
                 316
                 317
                 318
                 319 ;
                 320 ;                KEYBOARD CONTROLLER TEST
                 321 ;                ------------------------
                 322 ;
817F 3600        323        MVI     M,00H          ;BACK TO 8 CHAR. LEFT ENTRY
8181 3640        324        MVI     M,40H          ;SELECT READ FIFO.
                 325 ;
                 326 ;   THE KEYBOARD TEST STARTS WITH A CLEAR AND A CHECK THAT (a) THE FIFO
                 327 ; IS EMPTY AND (b) THERE IS NO INTERRUPT PENDING FROM THE 8279.
                 328 ;
8183 010101      329        LXI     B,0101H        ;INITIAL ERROR CODE (11).
```

```
 LOC  OBJ        LINE       SOURCE STATEMENT

 8186 CD2583      330        CALL    CLRALL          ;CLEAR FIFO (AND DISPLAY).
 8189 7E          331        MOV     A,M             ;READ STATUS.
 818A E60F        332        ANI     0FH             ;CHECK FIFO IS EMPTY.
 818C C21983      333        JNZ     ERROR           ;ERROR #10 IF NOT
 818F 04          334        INR     B               ;(B) = 1.
 8190 20          335        RIM                     ;READ INTERRUPT STATUS.
 8191 E610        336        ANI     10H             ;CHECK THAT RST 5.5 IS NOT SET.
 8193 C21983      337        JNZ     ERROR           ;ERROR #11 IF IT IS.
 8196 04          338        INR     B               ;(B) = 2.
                  339 ;
                  340 ;   NOW WAIT UNTIL A CHARACTER IS ENTERED.  A CHECK IS MADE TO SEE THAT ONLY
                  341 ; ONE CHARACTER IS IN THE FIFO AND THAT AN INTERRUPT IS NOW PENDING.  THE
                  342 ; INTERRUPT IS ALLOWED AND A CHECK IS MADE TO SEE THAT IT DOES, IN FACT, HAVE
                  343 ; THE EXPECTED RESULT (THIS ALSO CHECKS THE 8085 RST 5.5 MECHANISM).  THE FIF
                  344 ; IS READ AND THE RST 5.5 INTERRUPT INPUT AND THE FIFO ARE CHECKED TO SEE
                  345 ; THAT THEY ARE BOTH CLEAR.
                  346 ;
 8197 7E          347 KIT1:  MOV     A,M             ;READ STATUS.
 8198 E60F        348        ANI     0FH             ;WAIT UNTIL A CHAR. IS ENTERED.
 819A CA9781      349        JZ      KIT1            ;
 819D FE01        350        CPI     01H             ;CHECK THAT ONLY 1 CHAR. WAS ENTERED.
 819F C21983      351        JNZ     ERROR           ;ERROR #12 IF NOT.
 81A2 04          352        INR     B               ;(B) = 3.
 81A3 20          353        RIM                     ;RST 5.5 SHOULD NOW BE SET.
 81A4 E610        354        ANI     10H             ;
 81A6 CA1983      355        JZ      ERROR           ;ERROR #13 IF NOT.
 81A9 AF          356        XRA     A               ;CLEAR THE RST INTERRUPT FLAG.
 81AA 320220      357        STA     RSTFLG          ;
 81AD 3E0E        358        MVI     A,0EH           ;CLEAR THE RST 5.5 MASK.
 81AF 30          359        SIM                     ;
 81B0 FB          360        EI                      ;ALLOW THE INTERRUPT TO OCCUR.
 81B1 00          361        NOP                     ;
 81B2 00          362        NOP                     ;
 81B3 F3          363        DI                      ;
 81B4 3E0F        364        MVI     A,0FH           ;SET ALL MASKS AGAIN.
 81B6 30          365        SIM                     ;
 81B7 3A0220      366        LDA     RSTFLG          ;SEE IF THE INTERRUPT WAS SUCCESSFUL.
 81BA B7          367        ORA     A               ;
 81BB CA1983      368        JZ      ERROR           ;ERROR #14 IF FLAG NOT SET.
 81BE 04          369        INR     B               ;(B) = 5.
 81BF 3A0018      370        LDA     KDCD            ;READ THE FIFO.
 81C2 20          371        RIM                     ;THIS SHOULD HAVE CLEARED THE INTERRUPT
 81C3 E610        372        ANI     10H             ; (ONLY 1 CHAR WAS PRESENT).
 81C5 C21983      373        JNZ     ERROR           ;ERROR #15 IF NOT CLEARED.
 81C8 04          374        INR     B               ;(B) = 6.
 81C9 7E          375        MOV     A,M             ;READ FIFO STATUS AND CHECK THAT IT
 81CA E60F        376        ANI     0FH             ; IS EMPTY.
 81CC C21983      377        JNZ     ERROR           ;ERROR #16 IF NOT.
 81CF 04          378        INR     B               ;(B) = 7.
                  379 ;
                  380 ;   THE FIFO OVERRUN FLAG IS TESTED BY ALLOWING THE FIFO TO FILL UP AND
                  381 ; VERIFYING THAT THE FLAG IS SET WHEN A NINTH CHARACTER IS ENTERED.
                  382 ;
 81D0 7E          383        MOV     A,M             ;READ STATUS AND CHECK THAT O/R IS CLEAR.
 81D1 E620        384        ANI     20H             ;
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

81D3 C21983      385        JNZ     ERROR          ;ERROR #17 IF NOT.
81D6 04          386        INR     B              ;(B) = 8.
81D7 1E00        387        MVI     E,00H          ;NO. OF CHARS. IN FIFO.
81D9 3E08        388        MVI     A,08H          ;NO. OF CHARS. WANTED.
81DB CD3283      389 ORT1:  CALL    WTFRCH         ;WAIT FOR ENTRY.
81DE BB          390        CMP     E              ;8 CHARS. YET ?
81DF C2DB81      391        JNZ     ORT1           ;GO BACK IF NOT.
81E2 7E          392 ORT2:  MOV     A,M            ;NOW WAIT UNTIL OVERRUN IS SET (AFTER NEXT
81E3 E620        393        ANI     20H            ; KEY IS PRESSED).
81E5 C2E281      394        JNZ     ORT2           ;       .
                 395 ;
                 396 ;   THE KEYBOARD IS TESTED WITH THE 8279 IN ENCODED SCAN - KEYBOARD, 2 KEY
                 397 ; LOCKOUT MODE.  A CHAR. IS DISPLAYED IN THE RIGHT-MOST DIGIT OF THE DISPLAY
                 398 ; DISPLAY TO PROMPT THE ENTRY OF EACH CHARACTER FROM THE KEYBOARD.  ALL 22 KE
                 399 ; ARE TESTED.
                 400 ;
81E8 CD2583      401        CALL    CLRALL         ;CLEAR THE FIFO.
81EB 3685        402        MVI     M,85H          ;WRITE TO DISPLAY LOCATION 5.
81ED 3E00        403        MVI     A,00H          ;FIRST KEY NUMBER.
81EF 57          404        MOV     D,A            ;SAVE IT IN D.
81F0 CDE206      405 KBDT1: CALL    CONVRT         ;CONVERT TO DISPLAY CODE.
81F3 320018      406        STA     KDCD           ;DISPLAY THE CHARACTER.
81F6 CDEF06      407 KBDT2: CALL    RDKBD          ;WAIT FOR INPUT FROM KEYBOARD.
81F9 B7          408        ORA     A              ;
81FA CAF681      409        JZ      KBDT2          ;
81FD E63F        410        ANI     3FH            ;MASK OUT 'CHAR. PRESENT' FLAG.
81FF BA          411        CMP     D              ;IS IT THE RIGHT KEY ?
8200 C2F681      412        JNZ     KBDT2          ;IGNORE IT IF NOT.
8203 14          413        INR     D              ;NEXT CHARACTER CODE.
8204 7A          414        MOV     A,D            ;PUT IT IN A.
8205 FE16        415        CPI     22D            ;22 DONE ?
8207 C2F081      416        JNZ     KBDT1          ;REPEAT IF NOT.
                 417 ;
                 418 ;   SENSOR MATRIX MODE IS NOW TESTED.  AFTER PROGRAMMING THE 8279 TO
                 419 ; SENSOR MATRIX MODE 'END INTERRUPT' COMMANDS ARE SENT UNTIL NONE ARE
                 420 ; PENDING (THE 8279 MAY GENERATE MULTIPLE INTERRUPTS ON RESET).  A
                 421 ; CHARACTER IS THEN READ IN AND INTERRUPT OPERATION TESTED WITH THE
                 422 ; 8279 PROGRAMMED TO READ THE SENSOR RAM, AUTO-INCREMENT.  THE MODE IS
                 423 ; THEN CHANGED TO READ SENSOR RAM, NO INCREMENT, AND ANOTHER KEY IS READ
                 424 ; IN AND INTERRUPT TESTED.  FINALLY, THE MODE IS CHANGED BACK TO AUTO-
                 425 ; INCREMENT AND THE REMAINING 20 KEYS ARE READ IN, AFTER A PROMPT ON THE
                 426 ; DISPLAY.
                 427 ;
820A 3604        428        MVI     M,04H          ;SELECT SENSOR MATRIX MODE.
820C CD2583      429        CALL    CLRALL         ;CLEAR DISPLAY (AND FIFO).
820F 3650        430        MVI     M,50H          ;READ SENSOR RAM AUTO-INCREMENT.
8211 3685        431        MVI     M,85H          ;WRITE DISPLAY RAM LOC. 5, NO AUTO-INC.
8213 CD4D06      432        CALL    DLY500         ;WAIT 1/2 SEC. - TO ENSURE THAT THE LAST KEY
                 433                               ; IS RELEASED AND TO ALLOW INITIALISATION OF
                 434                               ; THE SENSOR RAM.
8216 1640        435        MVI     D,64D          ;MAX. NO. OF INITIAL INTERRUPTS ALLOWED.
8218 36E0        436 CLRINT: MVI    M,0E0H         ;SEND A 'CLEAR INTERRUPT COMMAND'.
821A 3E14        437        MVI     A,20D          ;WAIT 20ms TO ALLOW RESCAN OF THE KEYBOARD.
821C CD4006      438        CALL    DELAY          ;
821F 20          439        RIM                    ;SEE IF INTERRUPT PENDING.
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

8220 E610        440        ANI     10H            ;
8222 CA2C82      441        JZ      SMT1           ;PROCEDE WITH TEST IF NONE.
8225 15          442        DCR     D              ;DECREMENT COUNTER.
8226 C21882      443        JNZ     CLRINT         ;TRY AGAIN IF NOT DONE 64 TIMES.
8229 C31983      444        JMP     ERROR          ;ERROR #18 IF DONE 64 TIMES.
                 445
822C 04          446 SMT1:  INR     B              ;(B) = 9.
822D 1608        447        MVI     D,08H          ;NOW CHECK THAT ALL SENSOR RAM LOCATIONS
822F 3A0018      448 SMT2:  LDA     KDCD           ; ARE SET TO 00H (NO KEYS PRESSED).
8232 FE00        449        CPI     00H            ;
8234 C21983      450        JNZ     ERROR          ;ERROR #19 IF NOT.
8237 15          451        DCR     D              ;
8238 C22F82      452        JNZ     SMT2           ;
823B 04          453        INR     B              ;(B) = A.
                 454
823C 7E          455        MOV     A,M            ;READ STATUS AND TEST THAT THE 'SENSOR
823D E640        456        ANI     40H            ; CLOSURE' FLAG IS NOT SET.
823F C21983      457        JNZ     ERROR          ;ERROR #1A IF IT IS.
8242 04          458        INR     B              ;(B) = B.
                 459
8243 3640        460        MVI     M,40H          ;READ SENSOR RAM LOC. 0, NO AUTO-INC.
8245 3E0C        461        MVI     A,0CH          ;DISPLAY '0' TO PROMPT ENTRY OF KEY '0'.
8247 320018      462        STA     KDCD           ;
824A CD5783      463 SMT3:  CALL    SCSET          ;WAIT FOR A KEY CLOSURE.
824D 20          464        RIM                    ;TEST INTERRUPT STATUS.
824E E610        465        ANI     10H            ;
8250 CA1983      466        JZ      ERROR          ;ERROR #1B IF INTERRUPT NOT PENDING.
8253 04          467        INR     B              ;(B) = C.
8254 36E0        468        MVI     M,0E0H         ;END INTERRUPT COMMAND.
8256 20          469        RIM                    ;TEST INT. STATUS AGAIN.
8257 E610        470        ANI     10H
8259 C21983      471        JNZ     ERROR          ;ERROR #1C IF IT'S STILL THERE.
825C 04          472        INR     B              ;(B) = D.
825D 3A0018      473        LDA     KDCD           ;READ ROW 0 OF SENSOR RAM.
8260 57          474        MOV     D,A            ;SAVE THE DATA.
8261 CD6183      475        CALL    SCCLR          ;WAIT UNTIL KEY IS RELEASED.
8264 36E0        476        MVI     M,0E0H         ;CLEAR THE 'KEY RELEASE' INTERRUPT.
8266 7A          477        MOV     A,D            ;RETRIEVE ROW 0 DATA.
8267 FE01        478        CPI     01H            ;TEST FOR KEY 0.
8269 C24A82      479        JNZ     SMT3           ;GO BACK FOR ANOTHER KEY ENTRY IF NOT.
                 480 ;
                 481 ; THE TEST IS NOW REPEATED, WITH NON AUTO-INCREMENT ON READING THE SENSOR
                 482 ; RAM.  INTERRUPTS ARE NOW CLEARED BY READING THE RAM, NOT BY 'END INTERRUPT'
                 483 ; COMMAND.
826C 3E9F        484        MVI     A,9FH          ;DISPLAY '1'.
826E 320018      485        STA     KDCD           ;
8271 CD5783      486 SMT5:  CALL    SCSET          ;WAIT FOR KEY CLOSURE.
8274 20          487        RIM                    ;TEST INTERRUPT STATUS.
8275 E610        488        ANI     10H            ;
8277 CA1983      489        JZ      ERROR          ;ERROR #1D IF NONE PENDING.
827A 04          490        INR     B              ;(B) = E.
827B 3A0018      491        LDA     KDCD           ;READ FIRST ROW (THIS SHOULD CLEAR INT.)
827E 57          492        MOV     D,A            ;SAVE THE DATA IN D.
827F 20          493        RIM                    ;CHECK INTERRUPT IS NOW CLEAR.
8280 E610        494        ANI     10H            ;
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   10
8279 Functional Test Program  V9

 LOC  OBJ        LINE       SOURCE STATEMENT

 8282 C21983      495        JNZ     ERROR          ;ERROR #1E IF STILL SET.
 8285 04          496        INR     B              ;(B) = F.
 8286 CD6183      497        CALL    SCCLR          ;WAIT UNTIL KEY RELEASED.
 8289 3A0018      498        LDA     KDCD           ;CLEAR THE 'KEY RELEASE' INTERRUPT.
 828C 7A          499        MOV     A,D            ;RETRIEVE THE ROW 0 DATA.
 828D FE02        500        CPI     02H            ;TEST FOR KEY '1'.
 828F C27182      501        JNZ     SMT5           ;GO BACK FOR ANOTHER KEY IF NOT
                  502 ;
                  503 ; NOW RETURN TO AUTO-INCREMENT MODE AND PROMPT AND READ IN THE 20
                  504 ; REMAINING KEYS.
 8292 3E02        505        MVI     A,02H          ;NEXT CHAR. TO BE DISPLAYED.
 8294 57          506 SMT7:  MOV     D,A            ;SAVE CHARACTER IN D.
 8295 CDE206      507        CALL    CONVRT         ;CONVERT TO DISPLAY CODE AND
 8298 320018      508        STA     KDCD           ; DISPLAY IT.
 829B 3650        509 SMT8:  MVI     M,50H          ;READ SENSOR RAM, LOC. 0, AUTO-INC.
 829D CD5783      510        CALL    SCSET          ;WAIT FOR KEY ENTRY.
 82A0 36E0        511        MVI     M,0E0H         ;CLEAR THE INTERRUPT.
 82A2 1E08        512        MVI     E,08H          ;LOAD ROW COUNT INTO E.
 82A4 3A0018      513 SMT9:  LDA     KDCD           ;READ NEXT ROW OF SENSOR RAM.
 82A7 FE00        514        CPI     00H            ;TEST FOR A CLOSURE IN THIS ROW.
 82A9 C2B582      515        JNZ     FOUND          ;JUMP OUT IF SO.
 82AC 1D          516        DCR     E              ;DECREMENT ROW COUNTER.
 82AD C2A482      517        JNZ     SMT9           ;LOOK AT NEXT ROW IF NOT 0 YET.
 82B0 0E01        518        MVI     C,01H          ;PUT TEST NO. IN C.
 82B2 C31983      519        JMP     ERROR          ;ERROR #1F IF NO CLOSURE IN ANY ROW.
                  520
 82B5 0E07        521 FOUND: MVI     C,07H          ;SHIFT COUNTER.
 82B7 07          522 SMT10: RLC                    ;SHIFT NEXT BIT TO CY.
 82B8 DABF82      523        JC      SMT11          ;JUMP OUT IF FOUND.
 82BB 0D          524        DCR     C              ;DECREMENT COLUMN COUNTER.
 82BC C3B782      525        JMP     SMT10          ;LOOK AT NEXT BIT.
 82BF 3E08        526 SMT11: MVI     A,08H          ;CALCULATE ROW NO. FROM E
 82C1 93          527        SUB     E              ;
 82C2 87          528        ADD     A              ;MULTIPLY IT BY 8.
 82C3 87          529        ADD     A              ;
 82C4 87          530        ADD     A              ;
 82C5 81          531        ADD     C              ;ADD THE COLUMN NUMBER TO GIVE THE CHAR. NO.
 82C6 4F          532        MOV     C,A            ;SAVE IT IN C.
 82C7 CD6183      533 SMT12: CALL    SCCLR          ;WAIT UNTIL KEY RELEASED.
 82CA 36E0        534        MVI     M,0E0H         ;CLEAR THE 'KEY RELEASE' INTERRUPT.
 82CC 7A          535        MOV     A,D            ;RETRIEVE NO. OF THE DISPLAYED CHAR.
 82CD B9          536        CMP     C              ;COMPARE WITH THE INPUT CHAR.
 82CE C29B82      537        JNZ     SMT8           ;GO BACK FOR ANOTHER CHAR. IF NOT THE SAME.
 82D1 3C          538        INR     A              ;NEXT CHARACTER.
 82D2 FE16        539        CPI     22D            ;SEE IF ALL 22 DONE.
 82D4 C29482      540        JNZ     SMT7           ;REPEAT FOR NEXT CHAR. IF NOT.
                  541 ;
                  542 ;   FINALLY THE N KEY ROLLOVER AND 2-KEY LOCKOUT MODES ARE TESTED.  THE
                  543 ; 8279 IS FIRST PROGRAMMED FOR N KEY ROLLOVER MODE, AND THEN ANY CHARACTER
                  544 ; ENTERED INTO THE FIFO IS DISPLAYED.  THE ROLLOVER CAN THUS BE VERIFIED
                  545 ; MANUALLY.  WHEN 'NEXT' IS ENTERED THE 8279 IS SET TO 2 KEY LOCKOUT MODE
                  546 ; AND ANY CHARACTERS ENTERED INTO THE FIFO ARE AGAIN DISPLAYED.  WHEN
                  547 ; 'NEXT' IS ENTERED THE TEST IS FINISHED AND TEST 2 (OF SDK85S.V34) IS
                  548 ; STARTED.
                  549 ;
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   11
8279 Functional Test Program  V9

 LOC  OBJ        LINE        SOURCE STATEMENT

 82D7 3602        550         MVI     M,02H          ;ENCODED SCAN, N KEY ROLLOVER, 8 CHAR.
                  551                                ; LEFT ENTRY.
 82D9 CD2583      552         CALL    CLRALL         ;CLEAR DISPLAY AND FIFO.
 82DC 3640        553         MVI     M,40H          ;READ FIFO.
 82DE 3685        554         MVI     M,85H          ;WRITE TO DISPLAY RAM LOC. 5.
 82E0 CDEF06      555 NKRT1:  CALL    RDKBD          ;WAIT FOR KEY ENTRY.
 82E3 B7          556         ORA     A              ;
 82E4 CAE082      557         JZ      NKRT1          ;
 82E7 FE51        558         CPI     NEXT           ;TEST FOR 'NEXT'.
 82E9 CAF782      559         JZ      TKLT1          ;GO TO NEXT STAGE IF SO.
 82EC E63F        560         ANI     3FH            ;MASK OUT 'CHAR. PRESENT' FLAG.
 82EE CDE206      561         CALL    CONVRT         ;CONVERT TO DISPLAY CODE.
 82F1 320018      562         STA     KDCD           ;...AND DISPLAY IT.
 82F4 C3E082      563         JMP     NKRT1          ;GO BACK FOR NEXT CHAR.
                  564
 82F7 3600        565 TKLT1:  MVI     M,00H          ;SELECT TWO KEY LOCKOUT.
 82F9 CD2583      566         CALL    CLRALL         ;CLEAR DISPLAY AND FIFO.
 82FC 3640        567         MVI     M,40H          ;READ FIFO.
 82FE 3685        568         MVI     M,85H          ;WRITE TO DISPLAY RAM LOC. 5.
 8300 CDEF06      569 TKLT2:  CALL    RDKBD          ;WAIT FOR KEY ENTRY.
 8303 B7          570         ORA     A              ;
 8304 CA0083      571         JZ      TKLT2          ;
 8307 FE51        572         CPI     NEXT           ;SEE IF 'NEXT'.
 8309 CA9800      573         JZ      TEST2          ;GO TO NEXT TEST IF SO.
 830C E63F        574         ANI     3FH            ;ELSE MASK OFF 'CHAR PRESENT' FLAG.
 830E CDE206      575         CALL    CONVRT         ;CONVERT TO DISPLAY CODE.
 8311 320018      576         STA     KDCD           ;DISPLAY IT.
 8314 C30083      577         JMP     TKLT2          ;GO BACK FOR ANOTHER CHAR.
                  578
                  579
                  580
                  581
 8317 0E00        582 ERR0:   MVI     C,00H          ;SET TEST NO. FOR TEST 0.
 8319 CD9F06      583 ERROR:  CALL    ERRDSP         ;DISPLAY THE ERROR CODE IN BC.
 831C 76          584         HLT                    ;STOP: 8279 ERRORS ARE FATAL.
                  585
                  586 $EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                 587 ;*********************************************************************************
                 588 ;
                 589 ;
                 590 ;              SUBROUTINES FOR TESTS 0 & 1
                 591 ;              ---------------------------
                 592 ;
                 593 ;
                 594 ;              100 MILLISECOND DELAY ROUTINE
                 595 ;
                 596 ;   DLY100 CREATES A DELAY OF 100ms, USING THE 1ms DELAY ROUTINE, TO ALLOW
                 597 ; TIME FOR KEY DEBOUNCE WHEN THE 8279 IS OPERATING IN SENSOR MATRIX MODE.
                 598 ;
8319 F5          599 DLY100: PUSH    PSW            ;SAVE A.
831E 3E64        600         MVI     A,100D         ;
8320 CD4006      601         CALL    DELAY          ;1ms DELAY.
8323 F1          602         POP     PSW            ;
8324 C9          603         RET                    ;
                 604 ;
                 605 ;
                 606 ;                DISPLAY AND FIFO/SENSOR RAM CLEAR ROUTINE
                 607 ;
                 608 ;   CLRALL SENDS A 'CLEAR ALL' COMMAND TO THE 8279, THEN WAITS 1ms FOR
                 609 ; THE DISPLAY TO BE CLEARED.
                 610 ;
8325 F5          611 CLRALL: PUSH    PSW            ;SAVE A.
8326 3ECD        612         MVI     A,0CDH         ;CLEAR ALL COMMAND.
8328 320019      613         STA     KDCC           ;SEND TO 8279.
832B 3E01        614         MVI     A,01H          ;WAIT 1ms BEFORE RETURNING.
832D CD4006      615         CALL    DELAY          ;
8330 F1          616         POP     PSW            ;
8331 C9          617         RET                    ;
                 618 ;
                 619 ;
                 620 ;              WAIT FOR FIFO ENTRY ROUTINE
                 621 ;
                 622 ;   WTFRCH MONITORS THE FIFO STATUS UNTIL ANOTHER CHARACTER IS ENTERED.
                 623 ; E IS ASSUMED TO CONTAIN THE NUMBER OF CHARACTERS IN THE FIFO ON ENTRY.
                 624 ; ON EXIT IT CONTAINS THE NEW NO. OF CHARS. IN THE FIFO (1 MORE).
                 625 ;
                 626
8332 F5          627 WTFRCH: PUSH    PSW            ;
8333 1C          628         INR     E              ;INCREMENT CHARACTER COUNTER.
8334 3A0019      629 WFC1:   LDA     KDCC           ;READ FIFO STATUS.
8337 E60F        630         ANI     0FH            ;MASK NO. OF CHARS. IN FIFO.
8339 BB          631         CMP     E              ;EQUAL TO (E) YET ?
833A C23483      632         JNZ     WFC1           ;GO BACK AND WAIT IF NOT.
833D F1          633         POP     PSW            ;
833E C9          634         RET                    ;
                 635 ;
                 636 ;
                 637 ;       DISPLAY CLEAR DELAY ROUTINE
                 638 ;
                 639 ;   CLRDLY IS A 176us (160us + 10%) DELAY ROUTINE USED TO WAIT
                 640 ; UNTIL A DISPLAY CLEAR OPERATION IS COMPLETED.
                 641 ;
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

833F F5          642 CLRDLY: PUSH    PSW             ;
8340 3E26        643         MVI     A,38D           ;
8342 3D          644 CD1:    DCR     A               ;
8343 C24283      645         JNZ     CD1             ;
8346 F1          646         POP     PSW             ;
8347 C9          647         RET                     ;
                 648 ;
                 649 ;
                 650 ;        DISPLAY RAM WRITE ROUTINE
                 651 ;
                 652 ;   WRDRAM WRITES (C) TO THE DISPLAY RAM 16 TIMES.  IT ASSUMES THAT
                 653 ; THE DISPLAY RAM IS CURRENTLY SELECTED FOR WRITING, AUTO-INCREMENT
                 654 ; SO THAT ON EXIT ALL 16 DISPLAY RAM LOCATIONS WILL BE SET TO (C),
                 655 ; UNLESS ANY WRITE INHIBIT BITS ARE CURRENTLY SET.
                 656 ;
8348 E5          657 WRDRAM: PUSH    H               ;SAVE HL.
8349 D5          658         PUSH    D               ;SAVE D.
834A 210018      659         LXI     H,KDCD          ;
834D 1610        660         MVI     D,10H           ;LOCATION COUNTER.
834F 71          661 WDR1:   MOV     M,C             ;WRITE DATA TO DISPLAY RAM.
8350 15          662         DCR     D               ;DECREMENT COUNTER.
8351 C24F83      663         JNZ     WDR1            ;REPEAT 16 TIMES.
8354 D1          664         POP     D               ;
8355 E1          665         POP     H               ;
8356 C9          666         RET                     ;
                 667 ;
                 668 ;
                 669 ;        SENSOR CLOSURE SET ROUTINE
                 670 ;
                 671 ;   SCSET MONITORS THE 'SENSOR CLOSURE' FLAG IN THE STATUS WORD, WAITING
                 672 ; FOR A KEY CLOSURE.  WHEN SC GOES HIGH A 100ms DELAY IS CALLED (FOR
                 673 ; DEBOUNCE).
                 674 ;
8357 7E          675 SCSET:  MOV     A,M             ;READ STATUS.
8358 E640        676         ANI     40H             ;MASK SC.
835A CA5783      677         JZ      SCSET           ;WAIT UNTIL SET.
835D CD1D83      678         CALL    DLY100          ;DEBOUNCE BEFORE RETURNING TO CLEAR
8360 C9          679         RET                     ; THE INTERRUPT.
                 680 ;
                 681 ;
                 682 ;        SENSOR CLOSURE CLEAR ROUTINE
                 683 ;
                 684 ;   SCCLR MONITORS THE 'SENSOR CLOSURE' FLAG IN THE STATUS WORD, WAITING
                 685 ; UNTIL A KEY IS RELEASED.  WHEN SC GOES LOW A 100ms DELAY IS CALLED (FOR
                 686 ; DEBOUNCE).
                 687 ;
8361 7E          688 SCCLR:  MOV     A,M             ;READ STATUS.
8362 E640        689         ANI     40H             ;MASK SC.
8364 C26183      690         JNZ     SCCLR           ;WAIT UNTIL CLEAR.
8367 CD1D83      691         CALL    DLY100          ;DEBOUNCE BEFORE RETURNING TO CLEAR
836A C9          692         RET                     ; THE INTERRUPT.
                 693
                 694
                 695
                 696
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0        MODULE   PAGE   14
8279 Functional Test Program  V9

  LOC  OBJ        LINE        SOURCE STATEMENT

                   697         END


PUBLIC SYMBOLS


EXTERNAL SYMBOLS


USER SYMBOLS
CD1    A 8342   CHRTAB A 0728   CLRALL A 8325   CLRDLY A 833F   CLRDSP A 06D8   CLRINT A 8218
CLRKBD A 06FF   CONVRT A 06E2   DELAY  A 0640   DLY100 A 831D   DLY500 A 064D   DMT1   A 810F
DMT2   A 8129   DMT3   A 8146   DMT4   A 8163   DOBT1  A 80F1   DOBT2  A 80F3   DRT1   A 8060
DRT2   A 8069   DRT3   A 8077   DRT4   A 807B   DST1   A 8178   ERR0   A 8317   ERRDSP A 069F
ERROR  A 8319   FOUND  A 82B5   KBDT1  A 81F0   KBDT2  A 81F6   KCT1   A 80DD   KDCC   A 1900
KDCD   A 1800   KIT1   A 8197   NEXT   A 0051   NKRT1  A 82E0   ORT1   A 81DB   ORT2   A 81E2
RDKBD  A 06EF   RSTFLG A 2002   SCCLR  A 8361   SCSET  A 8357   SMT1   A 822C   SMT10  A 82B7
SMT11  A 82BF   SMT12  A 82C7   SMT2   A 822F   SMT3   A 824A   SMT5   A 8271   SMT7   A 8294
SMT8   A 829B   SMT9   A 82A4   TEST0  A 8000   TEST2  A 0098   TKLT1  A 82F7   TKLT2  A 8300
WDR1   A 834F   WFC1   A 8334   WIT1   A 80A6   WIT2   A 80C3   WRDRAM A 8348   WTFRCH A 8332

ASSEMBLY COMPLETE,  NO ERRORS
```

APPENDIX J

8279 FUNCTIONAL TEST ROUTINE OPERATING INSTRUCTIONS

8279 FUNCTIONAL TEST ROUTINE

The 8279 self (or "functional") test routine is intended to be run as part of Stage III of the SDK-85 signature analysis procedure.

To run the test, the 8279 self test program (KDCTST.V9, listed in Appendix I) must be stored in the external 1K ROM, located at address 8000H. The Stage III test program (Appendix E) must be patched to jump to 8000H immediately after initialisation. The 8279 test will then replace tests 0 and 1 of Stage III, and the following operating instructions will replace steps (2) to (4) inclusive of the Stage III operating instructions (Appendix F).

(2) A. Apply power to the system and press RESET.

Display	Procedure
8.8.8.8. 8.8.	Go to B.
Err 01 Err 02 Err 03 Any other display	Replace A13

B. Press any key (except RESET or VECT INTR)

Display	Procedure
6.6.6.6. 6.6.	Go to C.
Err 04 Err 05 Any other display	Replace A13

C. Press any key.

Display	Procedure
Blank	Go to D.
Err 06 Err 07 Any other display	Replace A13

D. Press any key.

This should produce some brief activity on the displays, with all digits finally showing ⌋

If display shows Err 08, 09 or any other display

then replace A13.

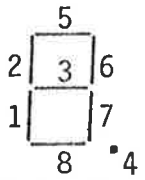
E. Press any key.

All digits should show ⊢.

If any other display results then replace A13.

F. Press any key.

The segments of each display digit will be lit for ½ second in the sequence shown:



and the bottom segment will be left on.

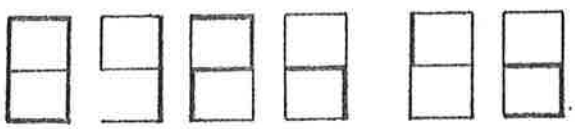
If display shows Err 0A, Err 0b, Err 0C, Err 0d, any other display or if the display sequence is not correct

then replace A13.

G. Press any key.

The characters 0, 1, 2, 3, 4 and 5 appear from left to right across the display, and are then superimposed on characters 8, 9, A, b, C and d.

The display should finally be:



If the display sequence is not correct then replace A13.

H. Press any key.

The characters 0, 1, 2, 3, 4 and 5 appear from left to right across the display, and are then replaced by characters 8, 9, A, b, C, and d.

If the display sequence is incorrect then replace A13.

I. Press any key.

The display will go blank and the characters will be shifted from right to left across the display, starting at the third digit from the left as follows:

```
* * 0 * * *
* 0 1 * * *
0 1 2 * * *
1 2 3 * * *
2 3 4 * * *
3 4 5 * * 0
4 5 6 * 0 1
5 6 7 0 1 2
6 7 8 1 2 3
      .
      .
      .
d E F 8 9 A     (final display)
```

where * represents a blank digit.

If the display sequence is incorrect then replace A13.

J. Press any key.

The display will go blank, and the characters 0 ... 7 will be shifted from right to left (starting at the rightmost digit) across the display. Then a series of superimposed characters will be shifted across the display (it is not important to verify these characters).

If the display sequence is incorrect then replace A13.

K. Press any key.

The display will show

* * * C * b

(where * represents a blank digit)

for five seconds.

If the display is incorrect then replace A13.

L. The display should go blank.

Display	Procedure
Blank	Go to M.
Err 10	Replace A13
Err 11	Examine pin A13-4 with a logic probe. If A13-4 is high then replace A13* else examine pin A11-9 with a logic probe. If A11-9 is low then replace A11 else the connection from A13-4 to A11-9 is faulty.

M. Press any key.

Display	Procedure
Blank	Go to N.
Err 12 Err 15 Err 16 Err 17	Replace A13.
Err 13	Examine pin A13-4 with a logic probe. If A13-4 is low then replace A13* else examine pin A11-9 with a logic probe. If A11-9 is high then replace A11 else the connection from A13-4 to A11-9 is faulty.
Err 14	Replace A11.

N. Press any key eight times.

After the eighth key entry, the character '0' should be displayed in the rightmost display digit.

If the display is incorrect then replace A13.

O. Press the '0' key and the display should be updated to show '1'.

A sequence of 22 characters will be displayed in this manner. As each character is displayed, the corresponding key should be pressed, and the display will be updated to the next character. An out-of-sequence key entry will be ignored.

Character	0	1	2	3	4	5	6	7	8	9	A	b	C	d	E	F
Key	0	1	2	3	4	5	6	7	8	9	A	b	C	d	E	F
Character		H			C			P		U				r		t
Key	EXEC			NEXT			GO		SUBST		MEM	EXAM		REG		SINGLE STEP

When the last key is pressed, the display will go blank briefly.

If any error is observed then replace A13.

P. The display will show one of the following:

Display	Procedure
0	Go to Q
Err 18 Err 19 Err 1A Any other display	Replace A13

Q. Press the '0' key.

Display	Procedure
1	Go to R
Err 1b Err 1C Any other display	Replace A13

R. Press the '1' key.

Display	Procedure
2	Go to S
Err 1d Err 1E Any other display	Replace A13

S. Proceed as in step 0., pressing the key corresponding to each character displayed. After the last key is pressed, the display will go blank.

If any error is observed, or the display shows 'Err 1F'

then replace A13.

T. Press any key except 'NEXT' and its corresponding character will be displayed in the rightmost display digit. Verify that, if any key (except 'NEXT') is pressed and held down, the display is updated when a second key is pressed.

If this does not happen then replace A13.

U. Press 'NEXT' and the display should go blank.

Verify that if any key (except 'NEXT') is pressed, its corresponding code is displayed, but if a second key is pressed while the first is held down, the display is not updated until the first key is released.

If this does not happen then replace A13.

V. Go to step (5) of Stage III (Appendix F).

REFERENCES

1. R.G. Bennetts, "The Philosophy of Testing Digital Systems - A Pragmatic Approach", *Electronics and Power*, Vol. 27, No. 2, Feb. 1981, pp 162-165.

2. P.G. Nutburn, "Economic Considerations of Testing Integrated Circuits", *Electronic Engineering*, Vol. 51, No. 631, Nov. 1979, pp 143-144.

3. J. Hotchkiss, "The Roles of In-Circuit and Functional Board Test", *Electronic Engineering*, Vol. 51, No. 625, July 1979, pp 63-71.

4. D. Izumi, "The Challenge of Microprocessor Chip Testing", *1975 WESCON Technical Papers*, Session 27.

5. D. Moralee, "Economics of Using A.T.E.", *Electronics and Power*, Vol. 26, No. 2, Feb. 1980, pp 176-182.

6. G. Moore, "VLSI: Some Fundamental Challenges", *IEEE Spectrum*, Vol. 16, No. 4, Apr. 1979, pp 30-37.

7. M.A. Breuer and A.D. Friedman, *Diagnosis and Reliable Design of Digital Systems*, Pitman, London, 1977.

8. S.B. Akers, "Test Generation Techniques", *Computer*, Vol. 13, No. 3, Mar. 1980, pp 9-15.

9. E.I. Muehldorf and A.D. Savkar, "LSI Logic Testing - An Overview", *IEEE Trans. Comput.*, Vol. C-30, No. 1, Jan. 1981, pp 1-17.

10. T.W. Williams and K.P. Parker, "Testing Logic Networks and Designing for Testability", *Computer*, Vol. 12, No. 10, Oct. 1979, pp 9-21.

11. R.G. Bennetts and R.V. Scott, "Recent Developments in the Theory and Practice of Testable Logic Design", *Computer*, Vol. 9, No. 6, June 1976, pp 47-63.

12. A. Bluestone, "Logical Environment Comparison Testing Handles Complex LSI Devices", *Computer Design*, Vol. 18, No. 4, Apr. 1979, pp 95-102.

13. D. Tose, "Digital Logic Board Design with Test Needs in Mind", *Electronic Engineering*, Vol. 49, No. 587, Jan., 1977, pp 46,49.

14. M. Thurman, "Cutting Production Costs with In-Circuit Test Systems", *Solid State Technology*, Vol. 21, No. 10, Oct. 1978, pp 77-79.

15. F.R. Boswell, "Designing Testability into Complex Logic Boards", *Electronics*, Vol. 45, No. 17, August 14, 1972, pp 116-119.

16. D. Tose, "Digital Logic Board Design with Test Needs in Mind", *Electronic Engineering*, Vol. 48, No. 586, Dec. 1976, pp 73-75.

17. D. Tose, "Design Circuits for Testability to Save Time and Cut Bottlenecks", *EDN*, Vol. 22, No. 10, May 20, 1977, pp 95-98.

18. D. Schneider, "Designing Logic Boards for Automatic Testing", *Electronics*, Vol. 47, No. 15, July 25, 1974, pp 100-104.

19. J.H. Jhu, "Design for Fault Isolation", *Electronic Design*, Vol. 23, No. 23, Nov. 8, 1975, pp 86-90.

20. C. Gaskell, "Designing in Testability", *Electronics Industry*, Vol. 6, No. 8, Aug. 1980, pp 21-23.

21. M.E. Granieri and W.J. Schmitt, "An Overview of Contemporary 'Portable' Digital A.T.E. System Architectures", *The Radio and Electronic Engineer*, Vol. 50, No. 9, Sept. 1980, pp 459-466.

22. S.P. Morse, B.W. Ravenel, S. Mazor and W.B. Pohlman, "Intel Microprocessors - 8008 to 8086", *Computer*, Vol. 13, No. 10, Oct. 1980, pp 42-60.

23. P.M. Russo, "VLSI Impact on Microprocessor Evolution, Usage, and System Design", *IEEE J. Solid-State Circuits*, Vol. SC-15, No. 4, Aug. 1980, pp 397-406.

24. D. Moralee, "Microprocessor Architectures: Ten Years of Development", *Electronics and Power*, Vol. 27, No. 3, Mar. 1981, pp 214-221.

25. J.C. Leininger, "On-Chip Testing Enhancement of a Single-Chip Microprocessor", *IBM Tech. Disc. Bull.*, Vol. 21, No. 1, June 1978, pp 5-6.

26. N. Purkis, "Testing Intelligent L.S.I.", *New Electronics*, Vol. 11, No. 22, Nov. 14, 1978, pp 134-138.

27. L.H. Goldstein, "Controllability/Observability Analysis of Digital Circuits", *IEEE Trans. Circuits and Systems*, Vol. CAS-26, No. 9, Sept. 1979, pp 685-693.

28. E.R. Hnatek, "Microprocessor Device Reliability", *Microelectronics and Reliability*, Vol. 17, No. 3, 1978, pp 379-385.

29. D. Hackmeister and A.C.L. Chiang, "Microprocessor Test Technique Reveals Instruction Pattern Sensitivity", *Computer Design*, Vol. 14, No. 12, Dec. 1975, pp 81-85.

30. K.P. Tashioglou, "Current Aspects of LSI Board-Level Testing", *Electronic Engineering*, Vol. 51, No. 620, Apr. 1979, pp 109-119.

31. J.P. Hayes and E.J. McCluskey, "Testability Considerations in Microprocessor-Based Design", *Computer*, Vol. 13, No. 3, Mar. 1980, pp 17-26.

32. J. Lyman, "LSI Boards Give Testers Fits", *Electronics*, Vol. 51, No. 24, Nov. 23, 1978, pp 91-92.

33. M.A. Breuer and A.D. Friedman, "Functional Level Primitives in Test Generation", *IEEE Trans. Comput.*, Vol. C-29, No. 3, Mar. 1980, pp 223-235.

34. A.R. Wilkinson, "Designing a Test Programme", *Electron*, No. 175, May 1979, p 13.

35. J.M. Bilton, "A Survey of Self-Test and BITE Program Generation", *Euromicro J.*, Vol. 6, No. 3, May 1980, pp 168-174.

36. S.M. Thatte and J.A. Abraham, "Test Generation for Microprocessors", *IEEE Trans. Comput.*, Vol. C-29, No. 6, June 1980, pp 429-441.

37. R.G. Bennetts, "Testing Digital Circuits: Guidelines for Research", *IEE J. Computers and Digital Techniques*, Vol. 2, No. 5, Oct. 1979, pp 185-186.

38. K. Jefferies, "Microprocessor Testing Techniques", *Electronic Engineering*, Vol. 50, No. 601, Jan. 1978, pp 61-62.

39. J.R. Armstrong and G.W. Woodruff, "Chip-Level Simulation of Microprocessors", *Computer*, Vol. 13, No. 1, Jan. 1980, pp 94-100.

40. A. Santoni, "Automatic Testers Can Characterize as well as Inspect", *Electronic Design*, Vol. 26, No. 24, Nov. 22, 1978, pp 84-88.

41. K. Ripley, "Testing Microprocessor-Based Circuits", *Electronic Engineering*, Vol. 49, No. 597, Oct. 1977, pp 65-68.

42. A.C.L. Chiang and R. McCaskill, "Two New Approaches Simplify Testing of Microprocessors", *Electronics*, Vol. 49, No. 2, Jan. 22, 1976, pp 100-105.

43. T.S. Bush, "Unique Problems Encountered in Testing Microprocessor Controlled PCBs", *WESCON-77 Conference Record*, Session 9.

44. S.E. Scrupski, "Why and How Users Test Microprocessors", *Electronics*, Vol. 51, No. 5, March 2, 1978, pp 97-104.

45. T.C. Chen, H.J. Gray, J.D. Wellin and B.Y. Woo, "A General Procedure for Designing Tests for LSI Digital Circuits", *Microelectronics J.*, Vol. 9, No. 4, March/April 1979, pp 4-12.

46. R.C. Goldblatt, "How Computers Can Test Their Own Memories", *Computer Design*, Vol. 15, No. 7, July 1976, pp 69-73.

47. W. Barraclough, A.C.L. Chiang and W. Sohl, "Techniques for Testing the Microcomputer Family", *Proc. IEEE*, Vol. 64, No. 6, June 1976, pp 943-950.

48. J. Knaizuk and C.R.P. Hartmann, "An Algorithm for Testing Random Access Memories", *IEEE Trans. Comput.*, Vol. C-26, No. 4, Apr. 1977, pp 414-416.

49. A.J. Borer, "Total Memory Test", *Microprocessors and Microsystems*, Vol. 4, No. 4, May 1980, pp 141-144.

50. K. Jessen, "In-Circuit Tester Answers μP Board Challenge", *Electronic Design*, Vol. 28, No. 23, Nov. 8, 1980, pp 97-101.

51. S.R. Purks, "Experiences with ATE Providing Testability of Microprocessor Boards", *IEEE Trans. Instrumentation and Measurement*, Vol. IM-27, No. 2, June 1978, pp 178-181.

52. S. Runyon, "Testing LSI-Based Boards: Many Issues, Many Answers", *Electronic Design*, Vol. 27, No. 6, March 15, 1979, pp 58-66.

53. M. Neil and R. Goodner, "Designing a Serviceman's Needs into Microprocessor-Based Systems", *Electronics*, Vol. 52, No. 5, March 1, 1979, pp 122-128.

54. G. Gordon and H. Nadig, "Hexadecimal Signatures Identify Troublespots in Microprocessor Systems", *Electronics*, Vol. 50, No. 5, March 3, 1977, pp 89-96.

55. H. Davis, "Testing μP-Boards Can be Easier - If you Design Your Own Tester", *Electronic Design*, Vol. 27, No. 5, March 1, 1979, pp 196-198.

56. E.B. Foley and A.H. Firman, "Testing Microcomputer Boards Automatically", *Computer Design*, Vol. 15, No. 12, Dec. 1976, pp 92-94.

57. E. Steinberg and R. Lecoq, "A Blackbox Approach to Testing and Fault Isolating an 8080 Chip Set on Existing A.T.E.", *WESCON-77 Conference Record*, Session 9.

58. J. McLeod, "High-Level Language Models ATE LSI", *Electronic Design*, Vol. 28, No. 22, Oct. 25, 1980, pp 32-33.

59. K.P. Wacks, P. de Bruyn Kops, F.J. Hill and M. Masud, "Model LSI Devices from Manufacturer's Data", *Electronic Design*, Vol. 28, No. 23, Nov. 8, 1980, pp 103-108.

60. J. Galiay, Y. Crouzet and M. Vergniault, "Physical Versus Logical Fault Models MOS LSI Circuits: Impact on Their Testability", *IEEE Trans. Comput.*, Vol. C-29, No. 6, June 1980, pp 527-531.

61. C. Robach, G. Saucier and C. Aléonard, "Microprocessor Systems Testing - A Review and Future Prospects", *Euromicro J.*, Vol. 5, No. 1, Jan 1979, pp 31-37.

62. R.L. Wadsack, "Fault Coverage in Digital Integrated Circuits", *Bell Sys. Tech. J.*, Vol. 57, No. 5, May-June 1978, pp 1475-1488.

63. J. McLeod, "Boards Become Self-Testing: Fast Test Systems Gain on Complex LSI", *Electronic Design*, Vol. 27, No. 24, Nov. 22, 1979, pp 28,30.

64. G. Crichton, "Testing Microprocessors", *IEEE J. Solid-State Circuits*, Vol. SC-14, No. 3, June 1979, pp 609-613.

65. S. Bisset, "LSI Tester Gets Microprocessors to Generate Their Own Test Patterns", *Electronics*, Vol. 51, No. 11, May 25, 1978, pp 141-145.

66. J. McLeod, "1981 Technology Forecast: Instruments", *Electronic Design*, Vol. 29, No. 1, Jan. 8, 1981, pp 210-232.

67. R. Shinn, "Automatic Test Systems Grow With Components and Boards", *Electronic Design*, Vol. 28, No. 21, Oct. 11, 1980, pp 125-133.

68. J. Shifman, "Programmable Sequencer Tests LSI In-Circuit", *Electronic Design*, Vol. 28, No. 23, Nov. 8, 1980, pp 115-118.

69. C. Pynn, "In-Circuit Tester Using Signature Analysis Adds Digital LSI to its Range", *Electronics*, Vol. 52, No. 11, May 24, 1979, pp 153-157.

70. J.D. Hutcheson, "Semiconductor Testing Requirements in the 1980's", *Solid State Technology*, Vol. 23, No. 8, Aug. 1980, pp 133-137.

71. G. Heftman, "IC Testers Turn Complex Semis to Good Account", *Electronic Design*, Vol. 28, No. 21, Oct. 11, 1980, pp 95-104.

72. E.J. Lerner, "Instrumentation: Laboratory/Bench Units", *IEEE Spectrum*, Vol. 18, No. 1, Jan. 1981, pp 67-68.

73. A.F. Shackil, "Tesging Debate Enters New Phase", *Electronics*, Vol. 52, No. 4, Feb. 15, 1979, pp 88-90.

74. R.W. Comerford, "Smaller Boards Mean Bigger Problems", *Electronics*, Vol. 52, No. 20, Sept. 27, 1979, pp 89-90.

75. D. Siewiorek and D. Rennels, "Workshop Report : Fault-Tolerant VLSI Design", *Computer*, Vol. 13, No. 12, Dec. 1980, pp 51-53.

76. T.J. Frechette and F. Tanner, "Support Processor Analyzes Errors Caught by Latches", *Electronics*, Vol. 52, No. 23, Nov. 8, 1979, pp 116-118.

77. N.C. Berglund, "Level-Sensitive Scan Design Tests Chips, Boards, System", *Electronics*, Vol. 52, No. 6, Mar. 15, 1979, pp 108-110.

78. B. Könemann, J. Mucha and G. Zwiehoff, "Built-In Test for Complex Digital Integrated Circuits", *IEEE J. Solid-State Circuits*, Vol. SC-15, No. 3, June 1980, pp 315-318.

79. B. Nicholson, "Troubleshooting μP-Based Equipment", *Electronics Industry*, Vol. 6, No. 7, July 1980, pp 15-19.

80. B. Nicholson, "Developments in Logic Analysers - Essential Digital Debugging Tools", *Electronics Industry*, Vol. 6, No. 9, Sept. 1980, pp 38-49.

81. J. Marshall, "Now Choosing the Right Logic Analyzer Requires a Logical Approach", *Electronic Design*, Vol. 26, No. 24, Nov. 22, 1978, pp 108-113.

82. E.J. Lerner, "Instrumentation: Field Service", *IEEE Spectrum*, Vol. 18, No. 1, Jan. 1981, p 66.

83. I.H. Spector, "Logic-State and Signature Analysis Combine for Fast, Easy Testing", *Electronics*, Vol. 51, No. 12, June 8, 1978, pp 141-145.

84. W.W. Moyer, "Designing a μC Test Unit", *Digital Design*, Vol. 8, No. 5, May 1978, pp 112-120.

85. "Field Servicing Microprocessor-Based Systems", *Digital Design*, Vol. 7, No. 10, Sept. 1977, pp 78-82.

86. *μScope 820 Operator's Handbook*, Intel Corporation, Santa Clara, California, 1977.

87. A. Santoni, "Self-Testing Terminals Ease System Troubleshooting", *Electronic Design*, Vol. 26, No. 24, Nov. 22, 1978, p 40.

88. M.D. Lippman and E.S. Donn, "Design Forethought Promotes Easier Testing of Microcomputer Boards", *Electronics*, Vol. 52, No. 2, Jan. 18, 1979, pp 113-119.

89. L. Lowe, "Designing for Testability", *Microprocessors and Microsystems*, Vol. 3, No. 1, Jan.-Feb. 1979, pp 3-6.

90. L. Yencharis, "μP-Based Instruments Last Longer When Test Help Comes From Within", *Electronic Design*, Vol. 27, No. 8, April 12, 1979, pp60-61.

91. V.P. Srini, "Fault Diagnosis of Microprocessor Systems", *Computer*, Vol. 10, No. 1, Jan. 1977, pp 60-65.

92. D. Sharrit, "Team Up a μP With Signature Analysis and Ease Trouble-shooting in the Field", *Electronic Design,* Vol. 27, No. 1, Jan 4, 1979, pp 138-143.

93. D.R. Ballard, "Designing Fail-Safe Microprocessor Systems", *Electronics,* Vol. 52, No. 1, Jan. 4, 1979, pp 139-143.

94. H.J. Nadig, "Field Testing Microprocessor Products With Signature Analysis", *1978 WESCON Technical Papers,* Session 29.

95. *Application Note 222: A Designer's Guide to Signature Analysis,* Hewlett-Packard Company, Palo Alto, California, Apr. 1977.

96. R.A. Frohwerk, "Signature Analysis: A New Digital Field Service Method", *Hewlett Packard J.,* Vol. 28, No. 9, May 1977, pp 2-8.

97. *Application Note 222-4: Guidelines for Signature Analysis, Understanding the Signature Measurement,* Hewlett-Packard Company, Palo Alto, California, Jan. 1981.

98. L.C. Badagliacca, "A Test and Service Approach as Unique as the Microprocessor Itself", *1978 WESCON Technical Papers,* Session 29.

99. *Application Note 222-3: A Manager's Guide to Signature Analysis,* Hewlett-Packard Company, Palo Alto, California, Oct. 1980.

100. J. Stephen, "Signature Analysis - A Reasonable Alternative", *Electronic Design,* Vol. 27, No. 5, Mar. 1, 1979, p 15.

101. A. Stefanski, "Free-Running Signature Analysis Simplifies Trouble-shooting", *EDN*, Vol. 24, No. 3, Feb. 5, 1979, pp 103-105.

102. M. Marshall, "Conferees Look at Logic Analysis Techniques", *Electronics*, Vol. 53, No. 3, Jan. 31, 1980, p 39.

103. M. Marshall, "Signature Analysis Wins New Acclaim", *Electronics*, Vol. 53, No. 4, Feb. 14, 1980, pp 102, 104.

104. *Application Note 222-2: Application Articles on Signature Analysis*, Hewlett-Packard Company, Palo Alto, California, May 1979.

105. *Application Note 222-1: Implementing Signature Analysis for Production Testing*, Hewlett-Packard Company, Palo Alto, California.

106. A. Natrasevschi, "SA Attacks Board Faults Without Extra Hardware", *Electronic Design*, Vol. 28, No. 21, Oct. 11, 1980, pp 191-195.

107. M. Riezenman, "European Instruments Tailor LSI to the Task", *Electronic Design*, Vol. 28, No. 21, Oct. 11, 1980. pp 81-90.

108. H. Bodt, "Instrumentation Industry Steers New Course", *Electronic Design*, Vol. 27, No. 17, Aug. 16, 1979, pp 149-151.

109. J. McLeod, "Field-Service Testers Sharpen On-Site Skills", *Electronic Design*, Vol. 28, No. 21, Oct. 11, 1980, pp 109-118.

110. D.G. West and H. Gillette, "Analyzer + Emulation = Faster Board Testing", *Electronic Design*, Vol. 28, No. 21, Oct. 11, 1980, pp 177-182.

111. M. Marshall, "Stimulus Unit Simplifies Failure Analysis", *Electronics*, Vol. 53, No. 14, June 19, 1980, pp 171-172.

112. R. Rhodes-Burke, "Applying Signature Analysis to Existing Processor-Based Products", *Electronics*, Vol. 54, No. 4, Feb. 24, 1981, pp 127-133.

113. M. Marshall, "Signature Analysis Tackles Mixed Logic", *Electronics*, Vol. 53, No. 25, Nov. 20, 1980, pp 44-46.

114. *Application Note 222-10: A Signature Analysis Case Study of a Z80-Based Personal Computer*, Hewlett-Packard Company, Palo Alto, California, Oct. 1980.

115. *Application Note 222-11: A Signature Analysis Case Study of a 6800-Based Display Terminal*, Hewlett-Packard Company, Palo Alto, California, Apr. 1981.

116. *Intel Component Data Catalog 1980*, Intel Corporation, Santa Clara, California, 1980.

117. *SDK-85 System Design Kit User's Manual*, Intel Corporation, Santa Clara, California, 1978.

118. D. Bursky, "Support Circuits - The 'Power' Behind Powerful Processors", *Electronic Design*, Vol. 28, No. 24, Nov. 22, 1980, pp 123-140.

119. J.G. Posa, "Peripheral Chips Shift Microprocessor Systems Into High Gear", *Electronics*, Vol. 52, No. 17, Aug. 16, 1979, pp 93-106.

120. *MCS-85 User's Manual*, Intel Corporation, Santa Clara, California, 1977.

121. M. Karpovsky and S.Y.H. Su, "Detection and Location of Input and Feedback Bridging Faults Among Input and Output Lines", *IEEE Trans. Comput.*, Vol. C-29, No. 6, June 1980, pp 523-527.

122. *Model 1640A Serial Data Analyzer Data Sheet*, Hewlett-Packard Company, Palo Alto, California, 1978.

123. J. Rattner and W.W. Lattin, "Ada Determines Architecture of 32-Bit Microprocessor", *Electronics*, Vol. 54, No. 4, Feb. 24, 1981, pp 119-126.

124. D.H. Smith, "Exercising the Functional Structure Gives Microprocessors a Real Workout", *Electronics*, Vol. 50, No. 4, Feb. 17, 1977, pp 109-112.

125. C. Robach and J.M. Gobbi, "Microprocessor Systems Testing", in *Euromicro Symposium, 4, Large Scale Integration*, H.W. Lawson, H. Berndt and G. Hermanson, Eds., North-Holland, Amsterdam, 1979, pp 66-73.

126. C. Gobach and G. Saucier, "Dynamic Testing of Control Units", *IEEE Trans. Comput.*, Vol. C-27, No. 7, July 1978, pp 617-623.

127. *Intel 8080 Microcomputer Systems User's Manual*, Intel Corporation, Santa Clara, California, 1975.

128. B. Nicholson, "LSI Design Support Essential", *Electronics Industry*, Vol. 6, No. 12, Dec. 1980, p 3.

129. D.A. Patterson and C.H. Séquin, "Design Considerations for Single-Chip Computers of the Future", *IEEE Trans. Comput.*, Vol. C-29, No. 2, Feb. 1980, pp 108-116.

130. J. Heering, "The Intel 8086, the Zilog Z8000, and the Motorola MC68000 Microprocessors", *Euromicro J.*, Vol. 6, No. 3, May 1980, pp 135-143.

131. *MC68000 Microprocessor User's Manual, Second Edition*, Motorola Inc., Austin, Texas, 1980.

132. *MCS-86 User's Manual*, Intel Corporation, Santa Clara, California, 1979.

133. *AmZ8000 Family Data Book*, Advanced Micro Devices Inc., Sunnyvale, California, 1980.

134. J. Boney and E. Rupp, "Let Your Next Microprocessor Check Itself and Cut Down Your Testing Overhead", *Electronic Design*, Vol. 27, No. 18, Sept. 1, 1979, pp 100-105.

LIST OF ABBREVIATIONS

AC	8085 Auxiliary Carry flag.
ACC	Accumulator (8085 functional unit).
ACT	Accumulator latch (8085 functional unit).
ALE	Address Latch Enable (8085 strobe output).
ALU	Arithmetic Logic Unit (8085 functional unit).
APG	Algorithmic Pattern Generation. .
ATE	Automatic Test Equipment.
C	A measure of instruction complexity.
CLK	Clock (8085 output).
CPU	Central Processing Unit; microprocessor.
CRO	Cathode Ray Oscilloscope.
CS0/ - CS7/	Chip Select outputs of the 8205 address decoder.
CY	8085 Carry flag.
Du	Display unavailable (8279 status flag).
DUT	Device Under Test.
EPROM	Erasable Programmable Read Only Memory.
F	A measure of instruction complexity.
FIFO	First-In First-Out buffer (8279 functional unit).
FU	Functional Unit.
GPIB	General Purpose Interface Bus.
H	Suffix to denote hexadecimal numbers.
HLDA	Hold Acknowledge (8085 status output).
ICE	In Circuit Emulation/Emulator.
INTA/	Interrupt Acknowledge (8085 output).
INTR	Interrupt (8085 input).
I/O	Input/Output.
IO/$\overline{M}$	Input-Output/Memory (8085 status output).
IRQ	Interrupt Request (8279 output).
K	Multiplier of 1024.

LSI	Large Scale Integration.
LSSD	Level Sensitive Scan Design.
MSI	Medium Scale Integration.
MUX	Register multiplexer (8085 functional unit).
NOP	No-operation instruction mnemonic.
PROM	Programmable Read Only Memory.
RAM	Random Access (read/write) Memory.
RD/	Read (8085 strobe output).
RL0-RL7	Return Line inputs of the 8279.
ROM	Read Only Memory.
RTL	Register-Transfer Level.
S0	8085 status output.
S1	8085 Status output.
s-a	Stuck-at.
SA	Signature Analysis.
SC	Sensor Closure (8279 status flag).
SID	Serial Input Data (8085 input).
SL0-SL3	Scan Line outputs of the 8279.
SOD	Serial Output Data (8085 output).
SSI	Small Scale Integration.
TMP	Temporary register (8085 functional unit).
TTL	Transistor-Transistor Logic.
UUT	Unit Under Test.
VLSI	Very Large Scale Integration.
WR/	Write (8085 strobe output).