

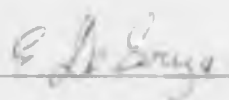
EDULAN : A LOCAL AREA NETWORK FOR
AN
EDUCATIONAL ENVIRONMENT

Eduardo Albino De Souza

A dissertation submitted to the Faculty of Engineering, University of the
Witwatersrand, Johannesburg, in fulfilment of the requirements for the
degree of Master of Science in Engineering.

DECLARATION

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.



E. A. DE SOUZA

On this the 16th day of March 1972

ABSTRACT

The needs and requirements of a typical Personal Computer (P.C.) network in an educational environment are discussed. A proposed system with a 'star-bus' topology with a file-server at the hub is described. Each ray of the star is a RS 422 twisted pair bus onto which a number of personal computers are attached. Access to the bus is by means of the CSMA CA (Carrier Sense Multiple Access with Collision Avoidance) access method. The buses operate at bit rates of between 1 and 1.5 Mbits per second. The interface to the Personal Computer is by means of a floppy disk drive emulator. The principles of operation of the floppy disk drive system in a Personal Computer are described as well as the design and implementation and testing of the disk drive emulator. Development of a device driver to create a number of logical drives on each Personal Computer with associated drives on the file-server is described.

ACKNOWLEDGEMENTS

I wish to express my thanks to:

- Mr. G. T. Gray who supervised my project, for his support and help, for the discussions we have had, and for the great autonomy I have had in my work.
- Professor H. E. Hanrahan for his assistance and the discussions we had.
- The Council for Scientific and Industrial Research for their financial assistance by way of a bursary during my period of research.
- Mr. S. Marantos whose assistance in a number of aspects is greatly appreciated.
- Mr. R. A. Schutz for his research into automated marking techniques for novice Pascal programmers which is to be implemented on the file-server.
- Mr. M. W. Hildyard For his research into the high level software for the file-server.
- I am also grateful to those laboratory technicians, who by their valuable advice, have helped me in this project.

TABLE OF CONTENTS

1.0 INTRODUCTION	1
2.0 NETWORKS IN AN EDUCATIONAL ENVIRONMENT: ARE THEY NECESSARY?	2
3.0 NETWORKS FOR EDUCATIONAL ENVIRONMENTS: WHY ARE THEY DIFFERENT?	4
4.0 NETWORK REQUIREMENTS	6
5.0 SYSTEM OVERVIEW	7
5.1 Use of one P.C. with the floppy disk drive emulator connected to the file-server. Only one floppy disk is emulated.	10
5.2 Emulation of 25 floppy disk drives using the hardware config- uration in (5.1).	12
5.3 The operation of the menu program and software protection on the system described in (5.2) above.	13
5.4 A number of P.C.'s operating as in (5.3) above.	15
6.0 THE FLOPPY DISK DRIVE EMULATOR	16
6.1 The Floppy Disk Drive Emulator (Hardware)	19
6.2 The Floppy Disk Drive Emulator (Software)	23
7.0 REQUIREMENTS OF THE FILE-SERVER	28
8.0 CONCLUSION	30
APPENDIX A THE STRUCTURE OF FLOPPY DISKS	33

APPENDIX B THE FLOPPY DISK DRIVE INTERFACE	38
APPENDIX C THE FLOPPY DISK CONTROLLER	41
APPENDIX D DATA RECORDING TECHNIQUES	43
APPENDIX E THE DECODER CIRCUITRY	47
APPENDIX F THE ENCODER CIRCUITRY	60
APPENDIX G THE INTERFACE BOARD	68
APPENDIX H THE MICROPROCESSOR CONTROLLER UNIT	70
APPENDIX I THE TRACK CACHE	81
APPENDIX J WHY 'STAR-BUS' NOT ETHERNET	89
APPENDIX K THE CSMA CA ACCESS PROTOCOL	91
APPENDIX L MESSAGE PASSING ON TRACK 40	96
APPENDIX M THE MS-DOS DEVICE DRIVER	99
APPENDIX N THE ENVISAGED FILE SERVER HARDWARE	119
APPENDIX O EMULATION SOFTWARE	121
APPENDIX P THE TRACK EMULATION SOFTWARE	143
APPENDIX Q TESTING THE FLOPPY DISK DRIVE EMULATOR	157

LIST OF REFERENCES	166
BIBLIOGRAPHY	168

LIST OF FIGURES

Figure	Page
5.1 Schematic Diagram Of The 'Star-bus' Network Topology	7
5.2 Schematic Diagram Of The Configuration For The Emulation Of a Single Floppy Disk	10
5.3 Schematic Diagram Of The Configuration For The Emulation Of 25 Floppy Disks	12
5.4 Logical Association And Physical Mapping Of The Disk Drives On The P.C. And The File-server	12
6.1 Components Comprising The Floppy Disk System In The P.C.	17
6.2 Flow Of Data In And Out Of The Emulator	21
6.3 Schematic Diagram Of The Emulator Track Counter ..	22
6.4 Flow-chart Of The Emulator Software	25
A.1 Subdivision Of The Disk Surface Into A Number Of Tracks	33
A.2 Diagram Of The IBM System 34 Double Density Track Format	35
A.3 Composition Of A Sector	37
D.1 Diagram Illustrating FM And MFM Encoding	44
D.2 Encoding Of The Address And I D Address Markers .	45
E.1 The Decoder Circuit Diagram	48
E.2 Timing Diagram Of The Counter Reset Circuit	50
E.3 Timing Diagram Of Write Data At The Start Of A Write Operation	51

E.4 Timing Diagram Of Clock Pulses Showing Expanded Pulses And Added Clock Pulses	54
F.1 Bit Cell Structure - Pulses May Occur During The First And Third Microsecond	61
F.2 Encoding Of The Address And I.D. Address Markers .	62
F.3 Counter Output Waveforms ..	64
F.4 The Encoder Circuit Diagram	67
G.1 The Interface Circuit Diagram	69
H.1 Block Diagram Of The Floppy Disk Drive Emulator ..	71
H.2 The 80188 CPU Circuit Diagram	72
H.3 The Memory Circuit Diagram	73
H.4 The Memory Map Of The 80188	76
H.5 Timing Diagram Of The RAS, CAS And WE signals	77
H.6 The I/O Circuit Diagram	79
H.7 The I/O Map	80
I.1 Cache Implementation With Each Data Segment Preceded By A Header	81
I.2 Separate Index And Data Areas For The Cache	82
I.3 Diagrammatic Representation Of The Track Cache Data Structure	82
K.1 Timing Of A CSMA CA Network Dialogue	93
K.2 CSMA CA Frame Format	95
P.1 Diagram Of The IBM System 34 Double Density Track Format	144
P.2 Bit Functions Of Write Register 5 Of The Z8530 ...	149

1.0 INTRODUCTION

I.B.M.'s launch of its Personal Computer (P.C.) in 1981 and the subsequent flood of P.C. clones on the market has accelerated the use of computers in a number of fields and education is no exception. A number of networks have been designed for these P.C.'s for commercial use but the design of networks for use in educational environments presents certain design challenges that are rarely encountered in other environments.

In this dissertation the necessity of a network system in an educational environment is discussed as well as the aspects that make these networks different to others. A 'star-bus' topology is proposed with a file-server at the hub. Access to each of the rays of the star which consist of an RS422 twisted pair operating at speeds between 1 and 1.5 Mbits per second is by means of the CSMA CA access method. A novel approach has been adopted whereby the interface between the P.C. and the network is by means of the floppy disk drive system of the machine. This hardware approach along with a small amount of software makes the network appear to the P.C. as a number of disk drives. A degree of software protection is included to reduce the possibility of software on the file-server being copied.

The design, implementation and testing of the P.C. interface comprised the major portion of the research and hence forms the major part of the dissertation. Aspects such as network media access, requirements of the file-server and software protection are also discussed.

Due to the extremely technical nature of most of the descriptions, a large number of these descriptions have been placed in the appendices to make the main body of the report easier to read. The P.C.'s referred to in a number of places in the report refer to the I.B.M. and compatible Personal Computers. These machines operate primarily on the PC-DOS and MS-DOS operating systems.

2.0 NETWORKS IN AN EDUCATIONAL ENVIRONMENT: ARE THEY NECESSARY?

To answer the question 'Are networks necessary in an educational environment' consider a set of computers used in an educational laboratory, and the advantages which will be provided by adding a network.

A Personal Computer (P.C.) laboratory is equipped with between 25 and 40 computers. Each of these computers has two floppy-disks and no hard-disk. This laboratory may be required to serve more than 100 students in total. During this computer course the students are required to do certain laboratory exercises, prepare assignments and write tests. One may initially think that it is a simple task to provide each of the computers with a 'Master Disk' which has the required Disk Operating System (DOS) along with the necessary laboratory exercises or assignment. If we stop to consider the time required to prepare these master disks we soon realise that the preparation of 40 such disks is a tedious task. We must be realistic since as experience has shown, it is rare that these exercises are fault free. As the exercise is carried out, invariably errors are found that have to be corrected. We are now confronted again with the master disk preparation task.

It would be far more efficient for one master disk to be corrected and be available immediately to the students. This would certainly be possible if these masters were kept on a central file server to which all the students had access via each P.C. The network would thus be required to provide the interconnection between the file server and each individual P.C..

Another aspect mentioned above viz. the writing of tests would benefit by the use of a network. Considering the number of students mentioned above we must examine the effort required by the examiners in such an

exercise. In most cases each student requires two floppy disks for the test in addition to the disk which contains the test itself. If a network was available there would only have to be one test master disk. The disks that the students 'write' their tests on have to be collected from the students, distributed to the markers, re-collected, then distributed to other markers for checking and again re-collected. Would it not be easier for all the tests to be stored on one device with possibly another copy on a floppy-disk in case of a catastrophe on the network or mass storage device. This would reduce the number of floppy-disks required to half. The marking of the tests would be simplified since the markers would be able to access the tests from this central point as well. If an artificial intelligence system was adopted to perform the task of marking the tests the centralised data storage would also be of great benefit.

The questions of software piracy and software security have been extremely contentious issues of late. In an educational environment in particular we may have a large amount of expensive software available which should not be copied. The software available to the user on the network can be broadly categorised into two well defined categories viz. That which can be copied and that which can't. In the first category we may have software such as master disks for laboratory exercises and assignments. This software along with each users' own work area can be copied to allow the students to be able to work elsewhere if they wished. In the latter category we have all the commercial software which is subject to strict copy-write protection.

It would be an advantage if the network could thus provide as large a degree as possible of software protection i.e. preventing students from copying the commercial software.

From the above discussion it is immediately evident that a network can provide extensive advantages in this type of environment.

3.0 NETWORKS FOR EDUCATIONAL ENVIRONMENTS: WHY ARE THEY DIFFERENT?

Networks for use in an educational environment have some peculiarities that make them unique in some respects from those used in commercial environments.

A number of factors emerge when we examine the personal computer laboratory more closely. Firstly when we consider the type of work performed in such a laboratory we find that most of it is file orientated. A student will enter a Turbo Pascal environment and then edit, save and run this program. When this program is executed it may read a data-file, write a data-file or both. The essential point to note is that all the units being processed are files. A second aspect is that although all the students may be working on the same exercise, there is no communication required between the computers. We should now contrast this with typical commercially available network systems. Two aspects that are considered to be of prime importance in these networks are message passing between P.C.'s and the facility for a number of users to be running the same program and editing a common data-base for example. These features are not required in the system envisaged and in fact message passing between P.C.'s is quite undesirable during the writing of a test. A number of users may run the same program but the running of each persons program is completely independent to that of other users e.g. there would be no sharing of data bases.

Network loading is also rather unusual. The network will probably experience very heavy loading at the start of the laboratory session for example when the students access the files for that session. The loading will then dwindle to a low level while the actual exercises are being performed. The loading will reach extreme proportions during the writing of a test initially when the test is accessed and likewise at the end of

the test when everyone is saving their attempts. During a test it would also be undesirable for some students to receive faster service than others. Loading such as described is rarely encountered in such proportions in other environments.

Another important aspect to be considered here is that of operating systems. In this type of environment it is necessary that more than one type of operating system can be used on the P.C.'s concurrently or at different times. In this respect a number of commercial networks fail. The number of networks that support more than one operating system are by far in the minority[1]. I quote from Olivetti's "10-NET" product overview

"10-NET" runs under MS-DOS 2.0. DOS function calls are intercepted by "10-NET" and re-routed across the network when appropriate.

This is one of many that share the same problem.

The aspects mentioned above illustrate the differing requirements between networks for use in educational environments as opposed to those used in business type environments.

4.0 NETWORK REQUIREMENTS

The requirements of a network for a particular P.C. laboratory that was investigated will be laid down in broad terms.

- o The network should be able to cater for up to 40 P.C.'s.
- o Each user on the network should have a response time comparable to or better than that obtained from a floppy-disk drive.
- o Each user should be able to access a pool of common software as well as his own user area.
- o As much copy protection as possible must be provided for software in the 'strictly copyright' area.
- o Since the response time should be comparable to a floppy-disk the cost of each P.C.'s interface should be comparable to that of a floppy-disk drive unit.
- o The network should be able to operate under more than one operating system.
- o There should be no restriction on the type of computer that can be connected on the network

5.0 SYSTEM OVERVIEW

The system consists of a 'star-bus' topology as shown below in Figure 5.1.

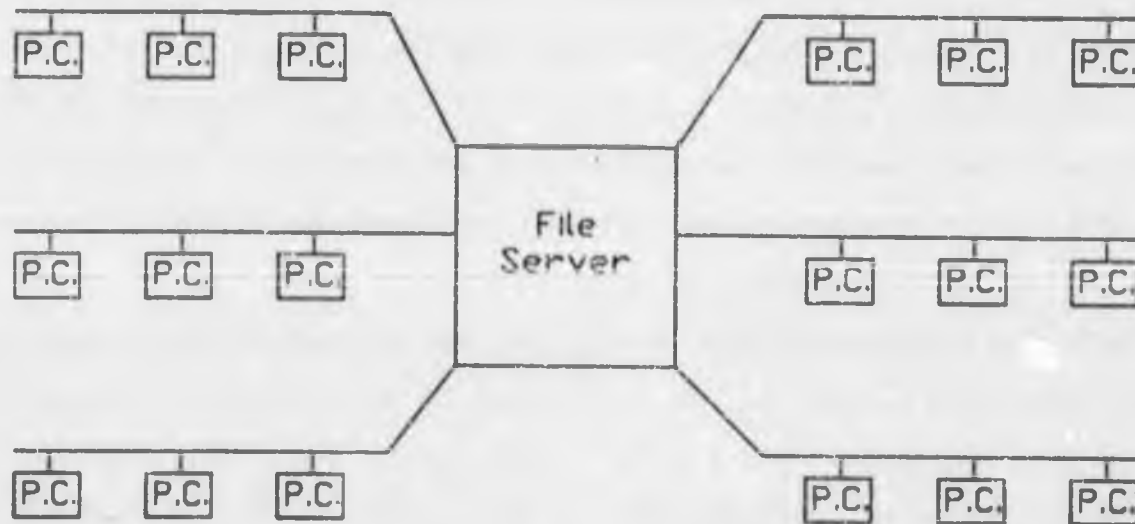


Figure 5.1 Schematic Diagram Of The 'Star-bus' Network Topology.

The reason for the term 'star-bus' is evident from the diagram. We have a star since the file server forms the hub of a star topology while each ray of the star is itself a bus. Hence the name star bus. Each of the busses uses the CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) access method. The bus itself operates at between 1 and 1.5 Mbits per second and consists of a RS422 twisted pair. The interface to each of the P.C.'s consists of a floppy disk drive emulator. This emulator is connected in the same manner that a normal floppy disk drive would be connected. The reasons for choosing this type of topology is described in detail in Appendix J. The data rate between the disk drive emulator and the P.C. is 250 Kbits per second. To maintain this data rate on a network of 25 P.C.'s for example requires data to be supplied at a bit rate of about 10 Mbits per second. the use of a single bus capable of this data rate would make the cost of each emulator much higher as well as adding to the complexity of the emulator circuitry. This single bus has rather been divided into a number of busses operating in parallel at a

lower data rate. A substantial reduction in the cost per emulator can be achieved but the complexity of the file-server hardware is increased. This compromise was accepted however.

The network appears to the P.C. on the hardware level as a floppy disk drive unit and to the software as a number of logical floppy disk drives. These additional logical disk drives are used by the file-server to provide access to the software pool as well as individual user areas. The data is transferred from the file server to the emulator in the form of tracks of data rather than individual files. This requirement is a direct result of the nature of the floppy disk emulator and will be explained later.

The P.C. is provided with a type of menu program, the selections being the software available for that particular operating system. One of these menu programs would be required for each of the operating systems that may be used on the P.C.. When the P.C. 'boots-up' initially off the network, the user would have to provide a user I.D. and password which is used in the network management. The menu program loaded would be that for MS-DOS since this is the most frequently used operating system at this stage. One of the options in this menu would be to change operating systems. The menu program performs a fundamental role in the software protection system since it is through this program that the user obtains access to the protected software on the file-server. The menu program 'maps-in' the requested software and hands over control to this software package. When the package terminates, control is returned to the menu program to 'map-out' the software package. By doing this the user, even if he leaves the menu program to the operating system, is unable to copy any of the protected software since it will be mapped out of the system. Problems do arise however in software packages that allow the user to copy files from within the package. Although elimination of the floppy disk drives altogether from the P.C. would ensure that no copying took place,

it was considered necessary that at least one drive should be retained. The reasoning behind this argument is that the students would be unable to take any of their work home with them if they wished to do so.

An explanation of the operation of the entire system is rather complicated and for this reason the explanation will start off with the simplest configuration. This will be expanded upon until the full system is reached.

The system developments that will be described are the following:-

1. One P.C. with the floppy disk drive emulator connected to the file-server. The emulation of a single floppy disk will be described.
2. The same hardware configuration as in (1) but 25 floppy disk drives will be emulated.
3. The operation of the menu program and software protection using the same configuration as in (2).
4. The combination of a number of P.C.'s operating as in (3).

5.1 USE OF ONE P.C. WITH THE FLOPPY DISK DRIVE EMULATOR CONNECTED TO THE FILE-SERVER. ONLY ONE FLOPPY DISK IS EMULATED.

This configuration is shown in the diagram below. See Figure 5.2.

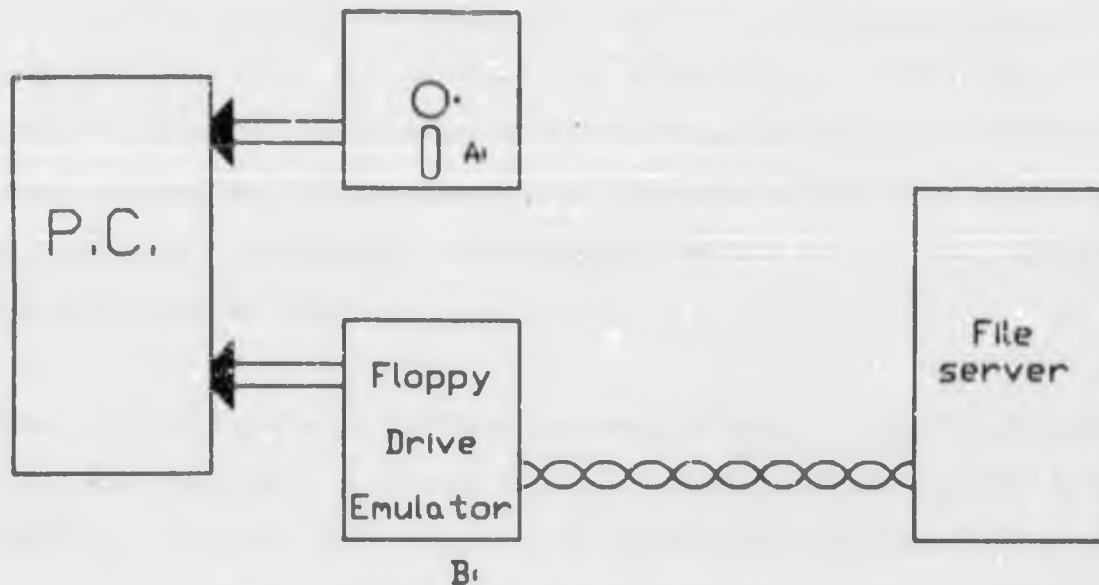


Figure 5.2 Schematic Diagram Of The Configuration For The Emulation Of A Single Floppy Disk

In the diagram above it can be seen that drive B on the P.C. is now actually the floppy disk drive emulator. This is done by simply plugging in the emulator instead of physical disk drive B.

Since we are only emulating one floppy disk, the file server only needs to be able to store 40 tracks of data for each side of the floppy disk. As will be described, the floppy disk drive emulator emulates whole tracks of data.

When a read access is made to drive B, the emulator will interpret the Floppy Disk Controller signals to determine which track and which side of the disk is required. Included as part of the emulator, as will be described, is a cache which stores tracks of data. At this point two situations may arise viz. the track cache on the emulator may be full,

or the cache may be empty or partly filled. In the latter case which will occur most often when the emulator has just been powered up, the track cache index will be searched to see if the required data is in the cache. Obviously immediately after power-up the cache will be empty. In general if the required data is found, the emulation process will commence immediately thus allowing the Floppy Disk Controller to begin reading in the required data. When the required data is not found in the cache, a 'request' is sent to the file-server for the required data. The file-server then accesses the required data from its cache or disk and sends it to the emulator. The emulator then updates its track cache and begins the process described again.

When the track cache is full and the required data is not in the cache, the cache management algorithm as will be described is implemented. This algorithm decides which data in the cache should be over-written. The process is then the same as described above viz. a request is sent to the file-server and the data returned is then placed in the area that has just been freed.

A write access to drive B is very similar to that described above for a read access. The entire track of data to which the write is taking place still has to be emulated and thus the same procedure as above is followed. When a track has been written to, this fact is flagged and the track cannot be replaced in the cache until the updated version of the track is written to the file server. This write takes place as soon as the emulator is able to access the file server for this write operation.

The fact then that the data is not coming from a physical floppy disk drive is totally invisible to the P.C.. The only modification that has been made to the P.C. is the replacement of the physical disk drive by the emulator.

5.2 EMULATION OF 25 FLOPPY DISK DRIVES USING THE HARDWARE CONFIGURATION IN (5.1).

To emulate 25 logical floppy disk drives requires the device driver that will be described to be loaded when the P.C. is 'booted up'. The configuration is now as shown below. See Figures 5.3 and 5.4.

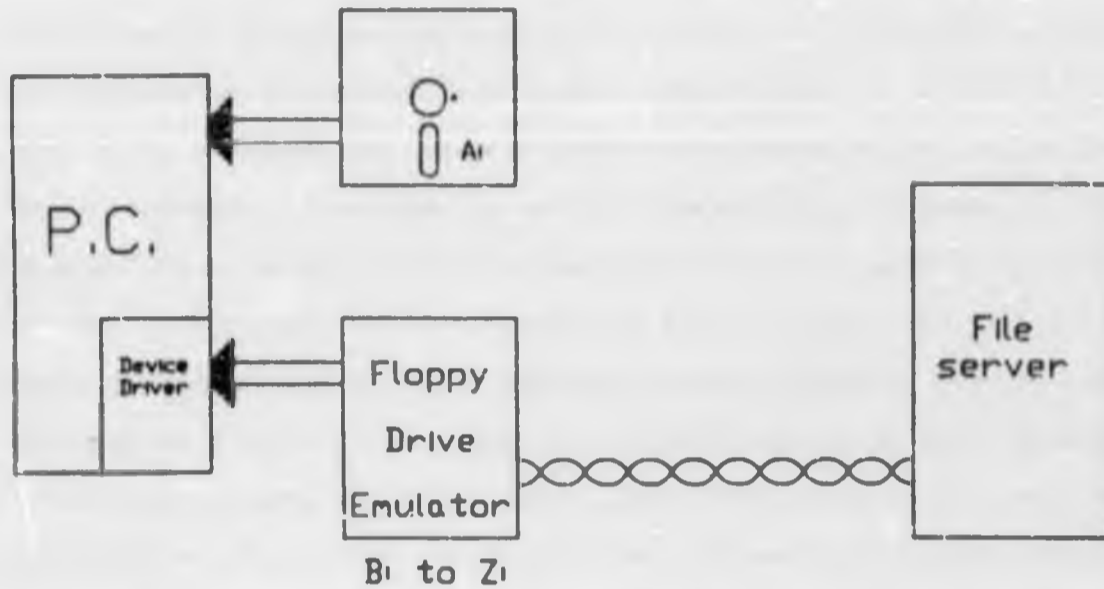


Figure 5.3 Schematic Diagram Of The Configuration For The Emulation Of

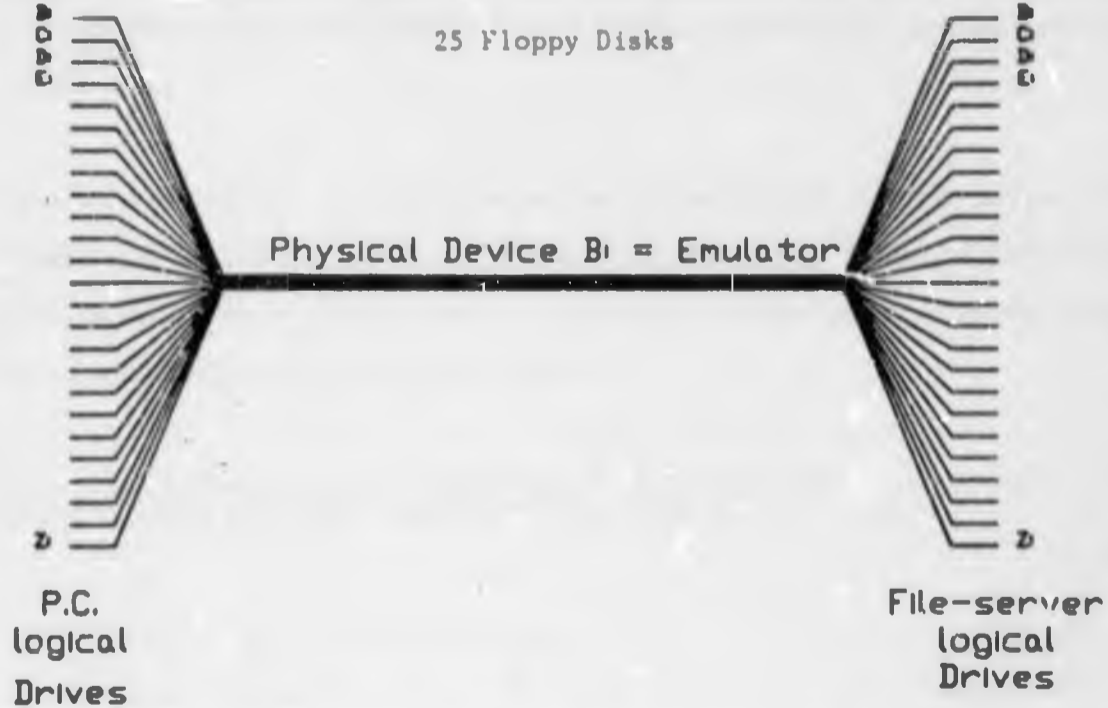


Figure 5.4 Logical Association And Physical Mapping Of The Disk Drives On The P.C. And The File-server

As described in detail in Appendix M, the device driver performs some 'pre-processing' of the DOS calls for the floppy disks. This pre-processing consists of mapping all the logical disk drives from J: to Z: to one physical device viz. drive B. (Note MS-DOS assigns the logical drives the letters of the alphabet from B to Z). This alone would be rather pointless since we would still be accessing the same data no matter which logical drive was selected. To overcome this a record is kept both in the device driver and the floppy disk emulator as to which logical device was accessed last. Whenever there is a change in the logical device being accessed, a 'message' is sent to the emulator informing it of the change. This message is sent to the emulator by writing a sector to track 40 which is beyond the normal range of track numbers used viz. 0 to 39. Sending this message is vital since we may have track 1, say, of each of the logical devices in the cache on the emulator and we must be able to distinguish between which one is required. This write is performed using one of the BIOS routines. The track emulator interprets this message and all subsequent searches of the track cache index include this drive identifier as one of the search criteria. The reason that track 40 is used for message passing and the mechanism used in message passing is described in Appendix L.

The user would then be able to access in effect 26 floppy disks. This includes his physical drive A along with 25 floppy disks of data on the file-server. Any of this data is accessible in the same way as data would be accessed from any other disk drive.

5.3 THE OPERATION OF THE MENU PROGRAM AND SOFTWARE PROTECTION ON THE SYSTEM DESCRIBED IN (5.2) ABOVE.

The operation of the menu program is vital in the attempt to prevent unauthorised software copying. The idea itself is rather simple. When the user 'boots-up' the computer the first menu program will automatically be executed. This menu will be that of the programs available that use

the MS-DOS operating system. This program then prompts the user for his choice. When a choice is made the file server is notified of the requested software. The file-server then associates virtual floppy disks in its mass storage with as many disks as the package requires. The current drive is changed to whichever drive the package is on. The program is then loaded and executed using the normal DOS call. This commences with a directory search for the file-name. The directory was empty previously but now it contains the required file-name.

An example may be useful to further illustrate the system. Consider a user that boots up the computer and is provided with the menu choice. He then branches from the menu to the operating system level. If the user now asks for a directory of any of the logical drives on the file server, the response will indicate that there are no files on any of these drives. He is unable then to copy any software.

Assuming the user returns to the menu program and requests a program called 'Gamma-graphix' for example. This package when normally run on floppy disks occupies three disks. The menu program then maps this package into drives B, C and D say. At this stage if the user was able to exit to the operating system and ask for a directory of drives B, C or D he would find all the files on these disks that would normally be on the individual floppy disks. (He would then be able to copy these files). The menu program would then commence execution of Gamma-graphix. When the user exits from Gamma-graphix whether normally or abnormally, the menu program would map Gamma-graphix out of drives B, C and D. If the user now gets a directory of these drives they would indicate that there are no files present.

When a software package is requested in the menu program, a message is passed to the disk emulator in the same way as in (2) above. The information contained in such a message is different however. The message content in this case would include the number of floppy disks the package

requires as well as an 'application number'. These messages are forwarded directly to the file-server. Each application package has an application number that has an important role in the overall system. For example when the file-server maps out a software package a number of tracks of that package will probably be present in the disk drive emulator track cache. We could remove these tracks from the cache though it would be more efficient to leave them there so that if the user left package A and then decided soon after that he was going to use package A again, the tracks of package A still in the cache would not have to be re-transferred across the network.

The application number becomes an added search criterion when the track cache index is being scanned to find a particular entry. The present application number being used is stored in the floppy disk drive emulator and is updated each time the menu program writes a message to the emulator.

The menu program would then provide a degree of software protection by not allowing the user to have direct access to the software files.

5.4 A NUMBER OF P.C.'S OPERATING AS IN (5.3) ABOVE.

Extending the above system to a number of P.C.'s does not require much effort at all. The only requirements now are that the user is prompted for a user I.D. and password when the P.C. is booted up initially. Each P.C. has associated with it a particular node number. The user I.D. provides a logical link to each user for the file-server while the node number provides a means for the network software to know where exactly to send the data. The network software also now has to contend for the network since the emulator file-server link is now shared by more than one user.

6.0 THE FLOPPY DISK DRIVE EMULATOR

Up until now reference has been made to the floppy disk drive emulator that provides the interface between the P.C. and the network. Naturally a few questions arise viz. What is this floppy disk drive emulator, how does it work and why emulate a floppy disk drive at all. These questions will now be answered.

Firstly, what is a floppy disk drive emulator? The disk drive emulator is a piece of microprocessor based circuitry that is able to emulate the characteristics of a floppy disk drive unit. Operation of the emulator will be discussed in detail later. The most interesting aspect is why emulate a floppy disk drive. If we look at microprocessor systems over recent years we soon see that the floppy disk has been the fundamental means of transferring data between these machines since they are in effect removable mass storage media. To execute a number of software packages on a P.C. for example we would load these packages from a number of floppy disks. The driving force behind the floppy disk drive emulator idea as the connection to the network is that instead of physically loading separate floppy disks, the network could provide the same data, that would be stored on a central file-server, to a floppy disk drive emulator. The P.C. is then totally unaware that the data is coming from a network and not from separate floppy disks. To appreciate why the floppy disk drive itself should be emulated we must examine the whole disk system of a P.C..

Components Comprising the Floppy Disk System in a P.C. The primary components of the floppy disk system in a P.C. are identified and their functions and structure described. The components are shown in Figure 6.1.

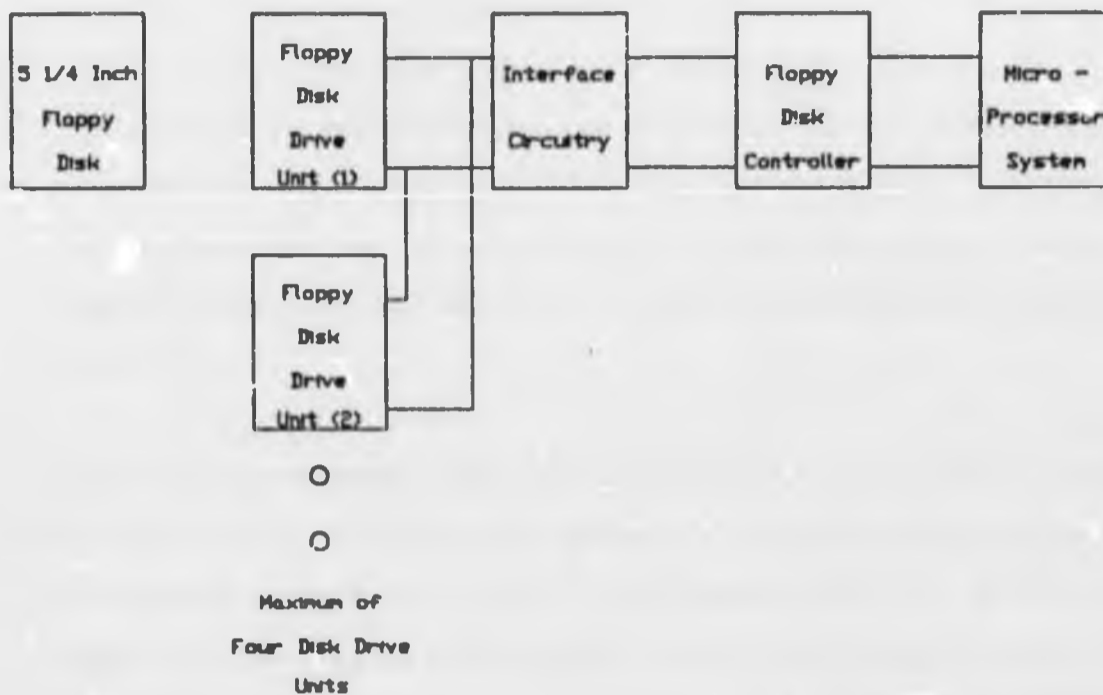


Figure 6.1 Components Comprising The Floppy Disk System In The P.C.

Functions:

Floppy Disk: A removable magnetic data storage medium which is double sided, divided into 40 tracks each subdivided into 9 sectors.

Floppy Disk Drive Unit: This provides a 'recording and play back' facility for the floppy disk.

Interface Circuitry: This provides the necessary line driver receiver circuits which are required between the disk drive units and the floppy disk controller as well as data separation circuitry.

Floppy Disk Controller: A dedicated processor provides and interprets the signals associated with the disk system. 'High level' commands are provided to the controller by the microprocessor system. The controller processes these commands thus alleviating the microprocessor of this task.

Microprocessor System: This consists of the hardware making up the remainder of the P.C. This provides decoding, DMA and interrupt facilities.

The BIOS (Basic Input/Output Subroutines) provides low level software for the floppy disk system (among other things).

From the above description we can see that only two components may be emulated viz. the floppy disk drive unit and the floppy disk controller. When we consider the inputs and outputs and complexity of emulation of each of these devices, emulation of the floppy disk drive unit is more favourable.

From the diagram we can see that the floppy disk controller is interfaced directly to the microprocessor system. This requires that the controller be emulated on both the hardware and software levels. As described in Appendix C the floppy disk controller is a very complex device. From a purely practical point of view emulation of the controller would require in a number of situations that the component be unsoldered from a circuit and be replaced with a plug in header, say, that is connected to the emulation circuitry. From an installation point of view the user would have a daunting task on his hands.

Emulation of the floppy disk drive unit is restricted to hardware emulation. The number of inputs and outputs to the emulator is much smaller. A disadvantage of this system, however, is that it is not possible to identify which sector is required by the floppy disk controller. It is only possible to isolate which track and which side of the disk is required. See Appendices B and C.

An alternative to emulating both the floppy disk controller as well as the floppy disk drive is the design and construction of a plug-in board for the P.C.. This is the form that most of the commercially available network boards take. These boards can interfere with the DMA and interrupt system on the P.C. (e.g. the same interrupt used by more than one board, the same I/O addresses may be used etc.). Extensive software is often also

required to drive them. This software can often cause problems when different operating systems are used with the same network card and may even be problematic with everyday software packages. In this respect emulation of the floppy disk drive unit does not interfere with DMA, interrupts or the software since we can virtually say that the software is un-modified. The MS-DOS device driver for the extra logical drives is virtually identical to that for the normal disk drive system. The only exception is the message passing to track 40 but even this is totally transparent to MS-DOS.

System busses into which plug-in boards are inserted and floppy disk controllers are very specific devices. Emulation of the disk drive unit is less specific since a number of disk drives have the same interface. The small discrepancies that may occur can be overcome by adapting the emulation principles developed. With speculation that future P.C.'s will be equipped with 3 $\frac{1}{2}$ inch drives makes emulation of the disk drive a logical step. Current specifications for 3 $\frac{1}{2}$ inch devices indicate that the emulator can be used, with possibly some small software modification on these systems [2].

By keeping the P.C. interface as we would by emulating the disk drive it would be possible to include P.C.'s of various makes on the same network. If these P.C.'s share the same operating system as other makes of machine on the same network, it would be possible for files to be shared by these machines. This would be more difficult if a plug in card were used, say, since these cards would have to be tailored to the individual systems.

6.1 THE FLOPPY DISK DRIVE EMULATOR (HARDWARE)

The floppy disk drive emulator can be divided into six sub-sections. These sections are:-

1. A decoder to process data being written to the disk drive.

2. An encoder to reproduce data being read from the disk drive.
3. A track counter to monitor the required head movement.
4. The CPU, RAM and EPROM to control the emulator and provide the track cache.
5. The network interface.
6. The interface to the floppy disk drive interface.

As described in Appendix D the data recorded on the floppy disk is MFM encoded. If the fact that the data is MFM encoded is disregarded for the moment and the data format on the floppy disk as described in Appendix A is considered, the following characteristics become apparent:

1. The data stream is very similar to that on a synchronous communications link.
2. The data is divided into blocks on which Cyclic Redundancy Checks (CRC) are calculated.

Considering these two observations a synchronous communications device was chosen to perform these tasks. Using NRZ (Non Return-to-Zero) encoding on this communications chip the data entering and leaving the emulator would just require MFM decoding and encoding respectively. The data flow is shown in Figure 6.2.

WD and RD are the data input and output to the floppy disk drive units. The lines shown here are the same as those on the 34 way interface cable (See Appendix B) except they have already been buffered.

This is conceptually how the data flow is managed. Unfortunately the implementation is not quite so simple. The full details of these subsections are given in Appendices E and F.

It is necessary to monitor the head position so that the correct track can be emulated when required. The required head movement is indicated to the floppy disk drive by two lines. These lines, DIRection and STEP, convey this information. The state of the DIR line determines whether the head must move inwards or outwards. Each pulse on the step line represents the movement to move from one track to the next.

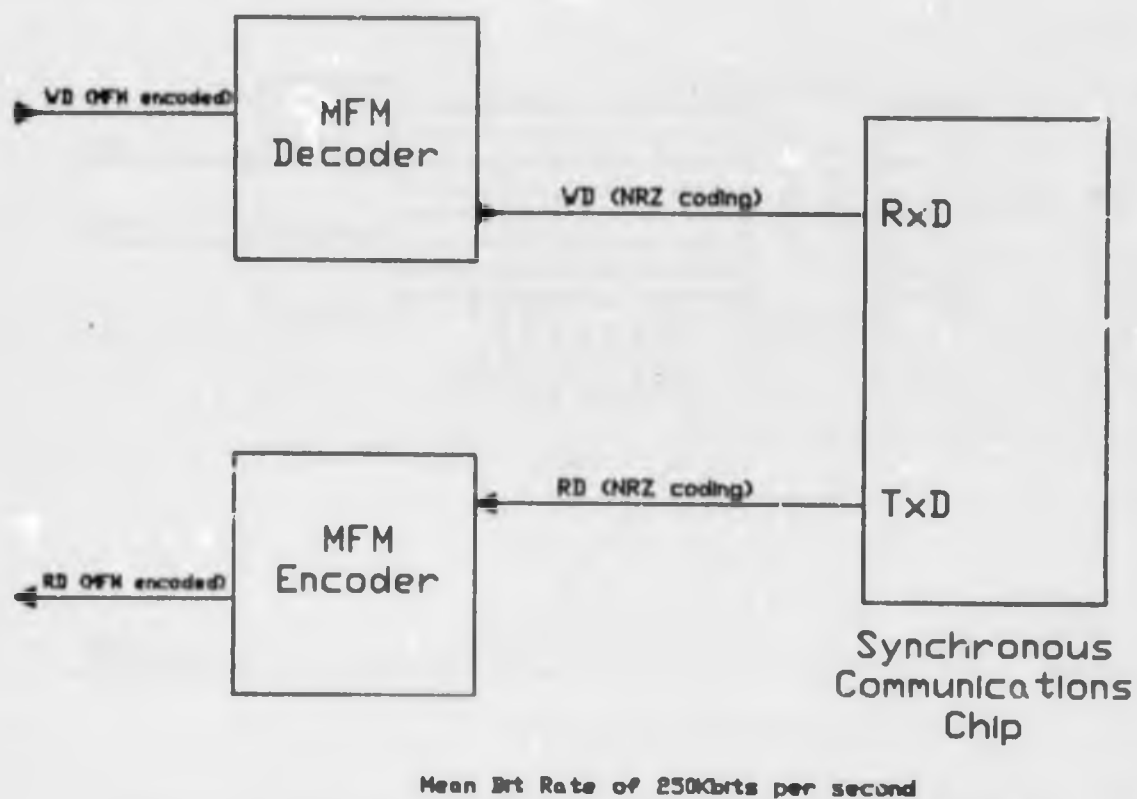


Figure 6.2 Flow Of Data In And Out Of The Emulator

For the floppy disk controller to know where the head is immediately after power-up, for example, a feedback signal is provided from the floppy disk drive unit to the disk controller. This signal, TRACK0, indicates that the head is positioned over track 0 (the outermost track of the disk). The floppy disk controller, when issued with a Recalibrate command [3], steps the head outwards until the TRACK0 signal becomes active. The floppy disk drive controller then sets its internal counter to zero.

The principle of the floppy disk controller's track control circuitry is extended to the emulator. An up-down counter whose direction is controlled by DIR and clocked by STEP performs this function. The outputs are gated to provide the TRACK0 signal when the counter's value is zero. The outputs are also made available to an input port on the CPU controller so that the counter can be monitored.

The track counter is shown below in Figure 6.3.

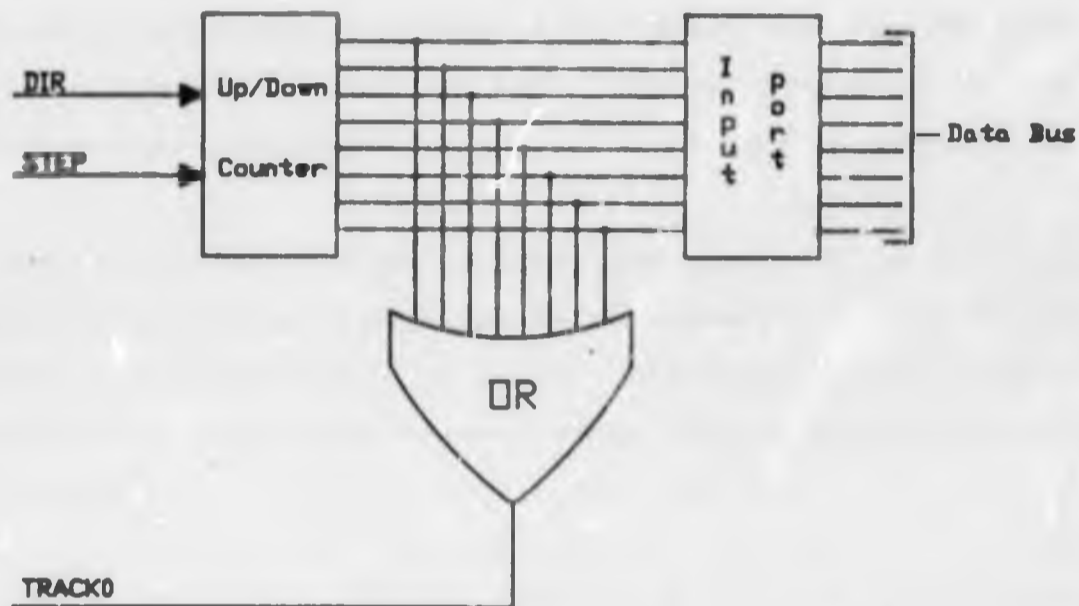


Figure 6.3 Schematic Diagram Of The Emulator Track Counter

The entire functioning of the emulator is co-ordinated using a micro-processor based system. the central processing unit used is the INTEL 80188. This is supported by 256Kbytes of RAM, (expandable to approximately 1Mbyte), 8Kbytes of EPROM, 2 Z8530 Synchronous Communications Controllers, an 8 bit input and an 8 bit output port.

The 80188 CPU is ideal in this application since it combines a number of peripheral devices within one physically small component. A dual channel DMA controller, 3 16 bit timers, an interrupt controller, bus controller,

chip select logic, ready generation logic as well as a clock generator are included in the package. 1 Mbyte of memory is directly addressable which is extremely useful for cache management. The timer, clock generator and DMA unit are used to overcome the problems commonly associated with dynamic RAM memory. See Appendix II. In the present implementation of the emulator only 256Kbytes of RAM are used. The emulation software and general 'house-keeping' software resides in the EPROM. The two Synchronous Communications Controllers (SCC) are used for the emulator and network. As mentioned above the input port is required for the track counter. An output port is provided to drive L.E.D.'s to indicate various states on the emulator and for general diagnostic purposes. The SCC's provide various other control inputs and outputs that are required.

The network interface consists of direct transmission of the data clock along with the data on a three line system between the nodes. The only additional circuitry required on the SCC's are the RS422 line drivers and receivers. The line drivers outputs are controlled by the RTS line of the network SCC.

Circuitry is required to interface the emulator to the floppy disk drive interface circuitry. This circuitry is based on an open collector bus type of system. All the outputs of the emulator have to be equipped with open collector drivers while the inputs must only accept information on the bus when the relevant physical drive is selected. See Appendix G. The circuitry is simple, however, and merely consists of latches and pull up resistors on the inputs and open collector drivers on the outputs.

6.2 THE FLOPPY DISK DRIVE EMULATOR (SOFTWARE)

The first aspect of the emulation software that will be discussed is that of the track cache. An area of RAM memory (191K) in this case has been set aside for creating such a cache.

An index of the cache contents is kept separate from the data area of the cache. This index contains entries identifying the track number, the head number (i.e. side of the disk), the application number, counters for the implementation of the caching algorithm and a pointer to the actual data area. The function of the track, head and application number has been explained in the system overview. When the cache is full, a caching algorithm is implemented to determine which track entry should be replaced. The algorithm is based on a method where both the frequency of access as well as when last an entry was accessed are used in the decision algorithm. This decision algorithm is particularly well suited to small sized caches. A detailed explanation of both the cache and the replacement algorithm is given in Appendix I.

Important aspects of the flow-chart will be mentioned here. See Figure 6.4.

The hardware of the emulator needs to be initialised and the memory tested before the emulation process can be started. The CPU itself needs to have certain parameters set up before the rest of the software can be executed. Memory and I/O mapping as well as 'wait-state' generation are of primary importance. One of the three timers as well as one of the DMA channels need to be configured to commence the refreshing of the emulator's dynamic RAM. DMA cycles are performed at fixed time intervals to carry out this function. Faulty memory locations are located by writing known bit patterns to an area of memory and then reading the contents of this area and checking for correspondence. If all the memory functions as required the track cache is initialised. The current disk and application numbers are given values that would not exist under normal circumstances and this ensures that when the cache index is searched, the search fails and the data is requested from the file server.

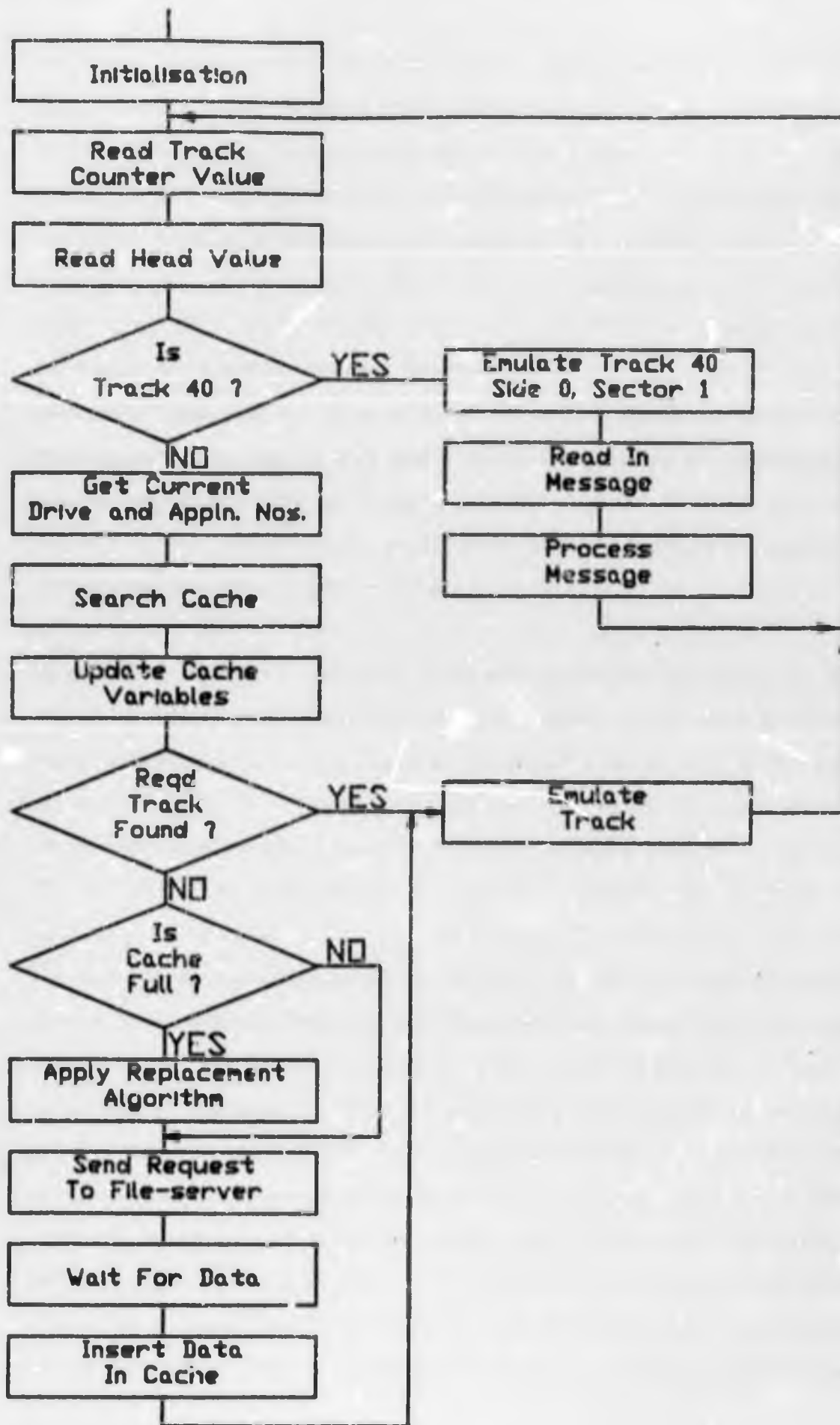


Figure 6.4 Flow-chart Of The Emulator Software

The track counter value is read from the input port. One of the control inputs on the SCC is used to check which head is required. This is possible since the head can be only one of two values viz 0 or 1. If the track counter value is 40, it is immediately identified with the exceptional case of a message being passed to the emulator and an abnormal track emulation takes place. The track size is reduced to 1 sector and only track 40, side 0, sector 1 is emulated. This is done primarily to save time as emulation of any further sectors is a waste of time. The message is then read in. This is done in the same way as any normal sector write except that the data is not written into the cache. Processing the message may simply involve updating variables on the emulator such as the current drive number and/or application number, sending an application request to the file server or both.

If the track counter value read is between 0 and 39 then using the stored drive and application numbers, the track index is searched to find an entry corresponding to the required track and side as well as the correct drive and application number. If the required entry is found, a pointer to the data area is returned and the track emulation process is initiated. The variables for cache management are also updated. In the case of an entry not being found the cache replacement algorithm comes into effect. The entry used least frequently is replaced. If the situation arises where two or more entries have the same frequency of use the one that hasn't been used for the longest period of time. This algorithm is described in detail in Appendix I. Once the area of the cache that is to be overwritten has been identified a message which includes information which identifies the requesting emulator (source address), the type of message (request for data), the required drive, application, track and side numbers, is sent to the file server. The actual access method on the network (CSMA-CA) is described in Appendix K. This method was chosen for its ease of implementation both in terms of hardware and software. With the small

number of users on each of the buses as explained in Appendix J (+- 5 users per bus), each user should receive sufficient use of the network.

When the data is returned from the file-server it is inserted into the track cache. The main emulation software is then called. As is explained in Appendix P, the track emulation begins with the generation of the index pulse followed by the gap preceding the first sector. Before each sector is emulated the track number as well as the side number is re-read from the respective input ports so that as soon as the floppy disk controller has performed the required function on the track initially chosen, the emulator can respond to any change in the requested track or side. Continued emulation of the originally chosen track is a waste. Emulation of a particular track continues therefore until there is a change in the emulator track or side inputs. The formatting information between sectors has been reduced to the minimum possible so that as much emulation time as possible is devoted to emulating data rather than formatting information that is useless to the user.

7.0 REQUIREMENTS OF THE FILE-SERVER

The requirements of the file-server both in terms of hardware and software are discussed.

From a high level point of view the file-server has to manage the access of a number of users both to an area of common software as well as the individual user areas. The file-server software is closely associated with the menu program on the P.C. and the finer details of both their implementations are beyond the scope of this report. In general though the menu program instructs the file-server to associate a certain number of floppy disks with logical disks on the P.C. Upon a request from the P.C. the file-server must be capable of providing any track required from one of these logical disks. The operating system on the P.C. is always in control of the logical disks themselves thus allowing the operating system on the file-server to be different from that on the P.C. It would be an advantage if the file-server used a multi-tasking operating system so that disk accesses for example can be taking place at the same time as other processing tasks. The file-server requires extensive mass storage facilities, the access time of which is of extreme importance. In this regard the implementation of a track cache on the file-server would reduce the access times particularly for software that is access by a number of users.

The lower levels of software are dependant on the type of hardware interface that exists between the file-server and the network. The two situations that can be defined here are the cases where the processor controlling the file-server controls the network, and then the case where the network is controlled by a separate processor. In the latter case communication with the network is by means of messages passed between the processors. In the first case the hardware of the file-server needs a synchronous communications port capable of handling SDLC communications

at speeds between 1 and 1.5Mbits per second. Software also has to be implemented to control the media access. This makes the use of a multi-tasking operating system virtually obligatory.

The second approach involves the inclusion in a multiprocessor environment of a processor to handle the network management and media access. Communication between the network processor and file-server processor is by means of two queues. Requests from the network are entered into the input queue while data for the network is extracted from an output queue. These queues would be maintained in a memory area that can be accessed by both processors. From the file-server's point of view communication with the network would consist of fetching requests from the input queue, processing them and providing the results in the output queue. This relieves the file-server of a large amount of processing.

The proposed system does not therefore pose any major constraints on the capacity, make of, or operating system used, on the file-server.

8.0 CONCLUSION

A set of network requirements for an educational environment were derived after analysing the requirements and peculiarities of such a system.

The proposed system, based on a 'star-bus' topology, provides a low cost, flexible system. The file-server which forms the hub of the star topology is subject to no major constraints and can be of any make and use any operating system but should have mass storage facilities with access times that are as short as possible. Using an RS422 twisted pair bus provides a medium speed link at between 1 and 1.5 Mbits per second as well as being cheaper, in terms of both material and installation costs, when compared with a coaxial cable based system. The CSMA CA access method provides a simple yet effective means for controlling each of the busses especially since the number of users per bus is small. No sophisticated hardware or software is required to implement it either. The use of a radio based system on one or more rays of the star for connecting computers that are some distance from the file-server should be investigated since the CSMA CA access method does lend itself to this type of medium as well.

The use of a floppy disk drive emulator as the interface between the P.C. and the network is certainly a new approach. This method eliminates most of the hardware incompatibilities that are often associated with plug-in network boards as well as dispensing with the need for operating system modifications. The design, implementation and testing of such a unit has been described. Certain constraints laid down in the network requirements have been satisfied by this interface. The cost of the interface is approximately the same as that of a typical floppy disk drive unit. No components have been used that are rare or expensive. Most of the components can be obtained from more than one manufacturer. The performance of the disk drive emulator unit when tested

as a stand alone unit proved to be excellent. Response time was within the requirements and no data errors were encountered during the testing.

Two disadvantages of the system must be mentioned. Due to the structure of the network and the P.C. interface, the nett data rate between the emulator and the P.C. is 250Kbits per second. When a single user, therefore, is using the system he will not receive the full benefit from the network operating at 1.5Mbits per second, say, due to the lower transfer rate between the emulator and the P.C.. The second matter is not really a disadvantage but should be considered for optimal performance. Since the data is transferred between the file-server and the emulator in the form of tracks, the files should be kept in a form that all the sectors on the tracks transferred actually carry useful data. This can be ensured by keeping the files on the file-server in a contiguous form.

A limiting factor of the system under heavy loading could be the data transfer rate between the file-server hardware and the file-server mass storage system. This would limit the amount of data available to the network. A cache in the memory of the file-server could reduce this limitation in situations where the loading is heavy but a number of users are requesting the same software.

These last two limitations should not make the overall system less attractive. To ensure that the requirements laid down are met further research and testing is required on the file-server and the network itself. The performance of the file-server is very important since all the data being transferred on the network passes into or out of it. Connection of a single floppy disk drive emulator to the file server will give a good idea of the best performance possible. Extending the testing to a full complement of P.C.'s on a ray of the star will give a good in-

dication of the overall performance of the system since any additional rays added will be operating in effect in parallel. The file-servers capabilities will be the limiting factor. Implementation of a printer and/or a plotter server using the message passing mechanism on track 40, as described, should be investigated. These servers will have to use other tracks though.

The research described in this dissertation has described the network system in general terms and the P.C. interface in detail. It has also provided a number of aspects that should be researched to optimise the performance of the network as a whole.

APPENDIX A THE STRUCTURE OF FLOPPY DISKS

The surface is divided into a number of concentric tracks or cylinders as shown in Figure A1. On the 5 $\frac{1}{4}$ inch disks there are typically 40 tracks. The beginning of a track is located by means of a small hole which is optically sensed on each revolution of the disk.

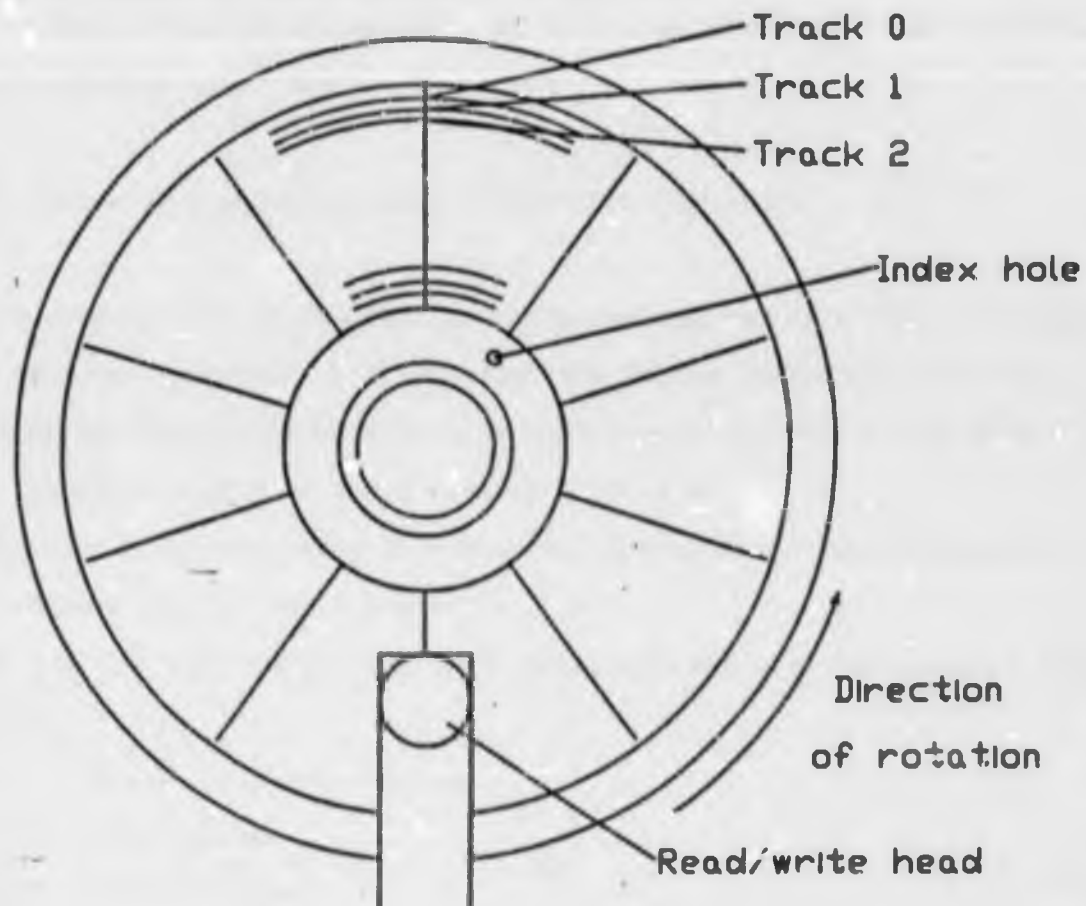


Figure A1 Subdivision Of The Disk Surface Into A Number Of Tracks.

Each track is further subdivided into sectors. See Figure A2. These sectors may be 128, 256, 512 or 1024 bytes long. The method of indicating the start of sectors is called soft sectoring which is the IBM standard method. This method allows software selection of sector sizes. Each data sector is preceded by a unique sector identifier that is read/written by the disk controller. The sector size is specified when the disk is formatted. The most popular disk format used in the I.B.M. type P.C.'s is

the 360K format. These disks are double sided which means there is a read/write head on either side of the disk. The disks rotate at 300 r.p.m which gives a rotation time of 200ms.

The 360 Kbyte capacity is achieved as follows:-

The disk is subdivided into 40 tracks which are then further subdivided into 9 sectors each with 512 bytes. (1/2Kbyte). (Note this applies to MS-DOS version 2 and subsequent versions). This gives a total of $\frac{1}{2} \times 40 \times 9 = 180K$. This is the capacity of each side of the disk and since the disk is double sided the total is 360K.

Each sector is composed of four fields. See Figure A3.

1) The Sector I.D. Field. This is ten bytes long and is written only when the disk is formatted. This provides the sector identification that is used by the controller when data is read from or written to the disk.

The ten bytes each have the following functions:

Bytes 1 - 3 are A1H bytes with missing clock pulses. These form part of the address marker. See Appendix D.

Byte 4 - I.D. Address marker. This specifies the beginning of the I.D. field.

Byte 5 - Track (cylinder) address

Byte 6 - Head address

Byte 7 - Sector address

Byte 8 - Sector length code

Bytes 9 and 10 - This is a 16 bit CRC for the I.D. field calculated on the first eight bytes.

The controller supplies the address mark during formatting and the three preceding A1H bytes. The rest is supplied by the processor software.

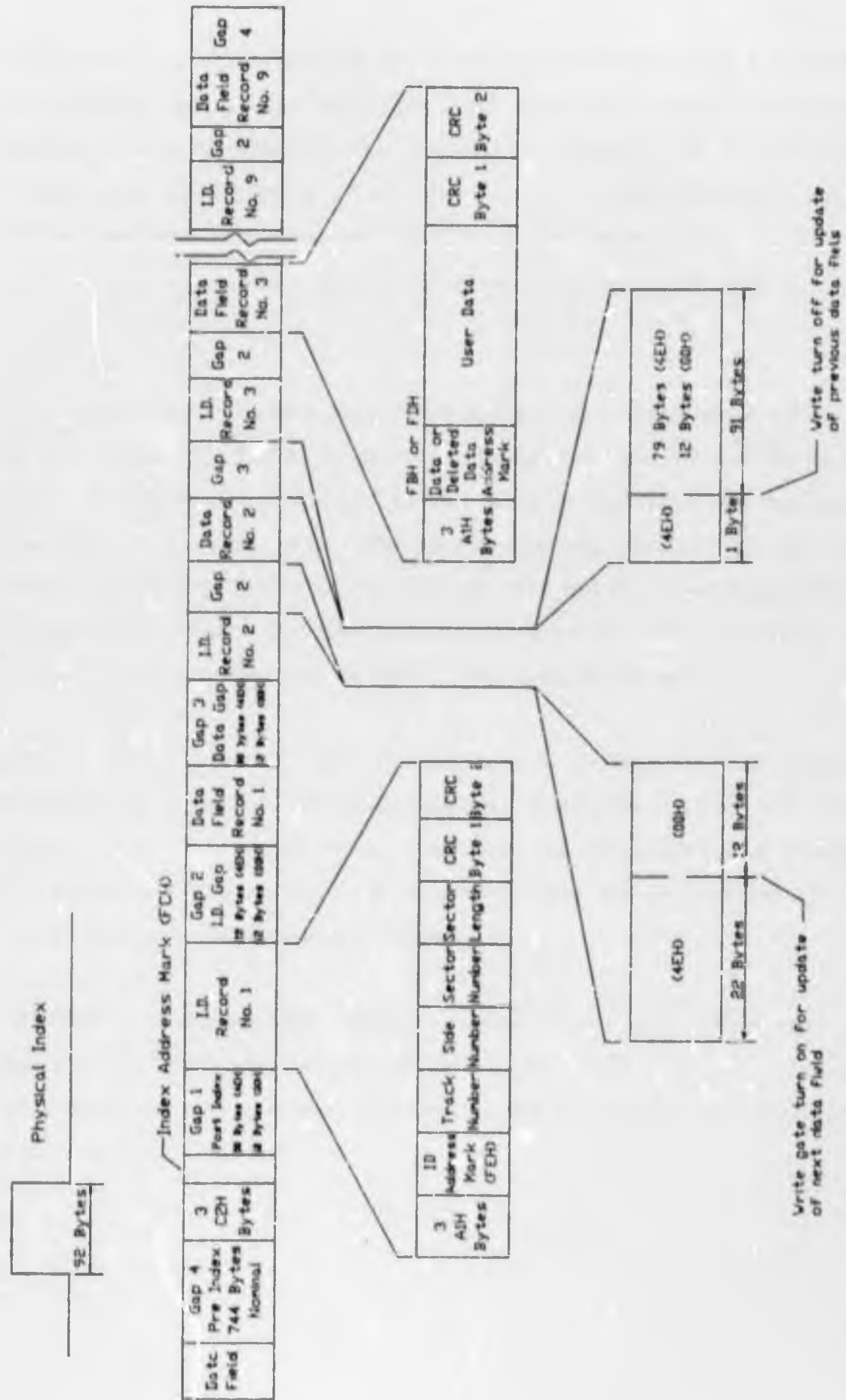


Figure A.2 Diagram Of The IBM System 34 Double Density Track Format

2) The Post I.D. Field Gap (Gap 2). This gap is written during formatting. During write operations, the device write circuitry is enabled within the gap and are re-written each time a sector is updated. During read operations the trailing bytes of the gap i.e. the 12 00H bytes are used to synchronise the data separator logic with the upcoming data field. The remainder of the gap consists of 4EH bytes as shown in the Figure.

3) The Data Field

This field also commences with 3 Alh bytes with missing clock pulses as in the sector I.D. field. In this case though they are followed by an FBH byte. If the data area is found to be faulty during the format operation the FBH byte is replaced by a FDH byte indicating a bad sector. The rest of the field is filled with the required data and has a two byte CRC for the whole data field appended at the end (Note The CRC calculation includes everything from and including the three AlH bytes).

4) Post Data Field Gap (Gap 3). This gap is written when the track is formatted and separates the proceeding data field from the next I.D. field on the track. This gap is not written after the last sector on the track. The number of bytes in this gap is programmable and is 22 bytes of 4EH and 12 bytes of 00H as shown in Figure A2.

The CRC's calculated above are calculated on the most significant bit first of each byte. The polynomial used is $x^{16} + x^{12} + x^5 + 1$. The CRC generators and receivers must be preset to all 1's before the calculations are started.

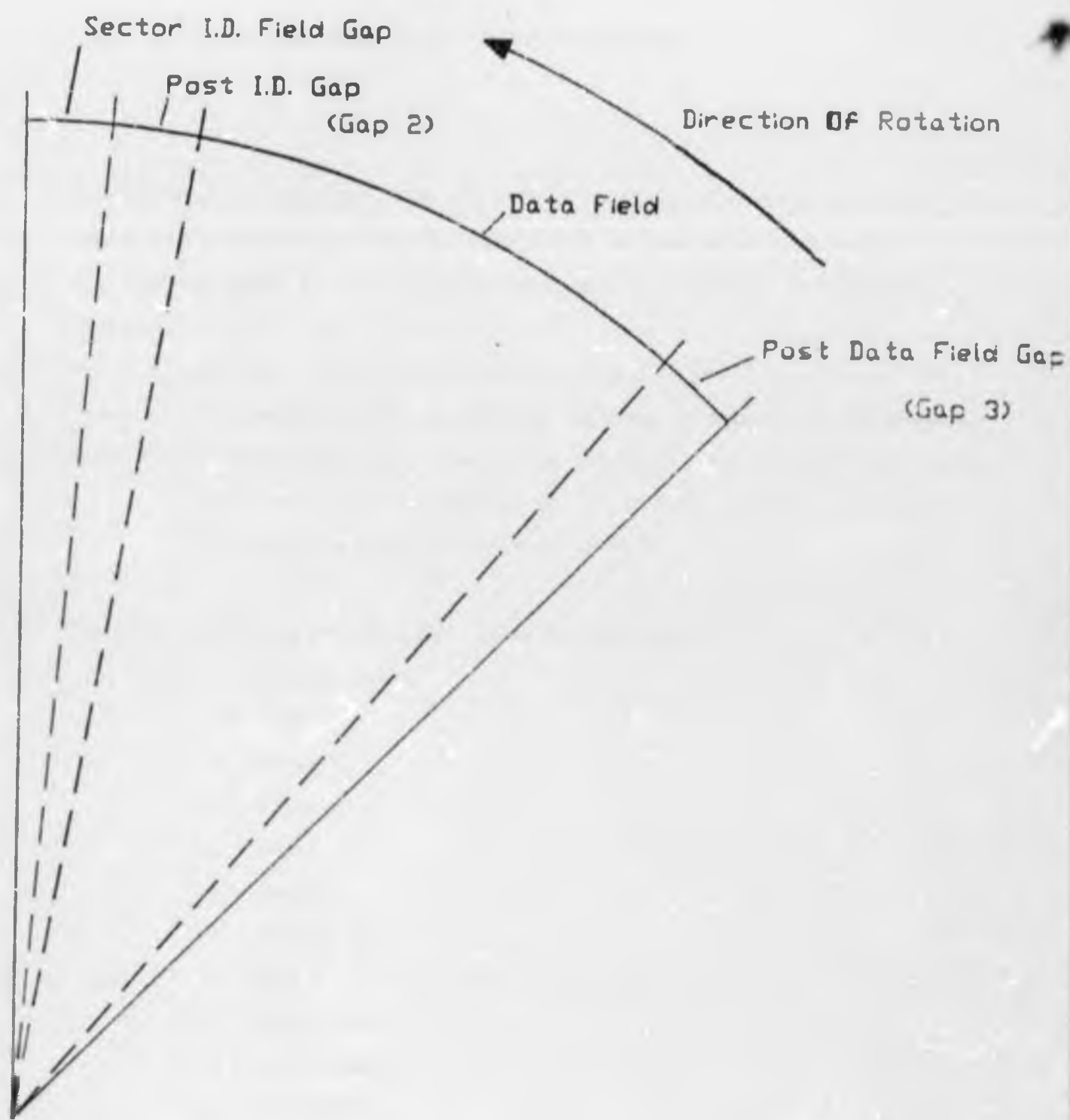


Figure A.9 Composition Of A Sector

APPENDIX B THE FLOPPY DISK DRIVE INTERFACE

The interface to the floppy disk drive unit consists of a 34 way connector cable. The odd numbered pins of this cable are grounded (i.e. pins 1 - 33). The remaining lines carry input and output signals at standard TTL levels.

Output lines: These lines are driven by standard open collector gates. The diskette drives provide the pull-up resistors for these lines.

Input lines: Each input line presents one low power Schottky (LS) load to the diskette drives connected to the interface and is terminated with a 150 Ohm load resistor to +5 V.

The even numbered lines have the following functions:

- 2,4,6,34 : No connection
- 8 : Index
- 10 : Motor 0
- 12 : Select 1
- 14 : Select 0
- 16 : Motor 1
- 18 : Direction
- 20 : Step
- 22 : Write Data
- 24 : Write Enable
- 26 : TR0 (Track 0)
- 28 : Write Protect
- 30 : Read Data
- 32 : Hdsel (Head select)

In the following table 0 denotes an output while 1 denotes an input from the P.C.'s point of view.

Signal -----	Direction -----	Description -----
Select 0, select 1	0	Drive Select - used to enable the driver outputs and Rx inputs of the diskette drive - active low. Select 0 for drive A, select 1 for drive B.
Motor 0, motor 1	0	Motor Enable - controls spindle motor low - motor on high - motor off Select 0 for drive A, select 1 for drive B.
Step	0	Step - used to move the R/W head in or out one cylinder on the selected drive for each pulse present on the line. The direction of motion is determined by the DIR line.
DIR	0	Direction - Moves head in if low, out if high, used in association with Step.
Hdsel	0	Head select - The side 1 (upper head) is selected when this line is <i>low</i>
WE	0	Write Enable - enables write current to the R/W head when active (low).
WD	0	Write Data - data to be written on disk
INDEX	1	Index - one pulse appears on this line each time the index hole is detected (i.e. once per revolution)
WP	1	Write protect - low when disk write

Signal -----	Direction -----	Description -----
Select 0, select 1	0	Drive Select - used to enable the driver outputs and Rx inputs of the diskette drive - active low. Select 0 for drive A, select 1 for drive B.
Motor 0, motor 1	0	Motor Enable - controls spindle motor low - motor on high - motor off Select 0 for drive A, select 1 for drive B.
Step	0	Step - used to move the R/W head in or out one cylinder on the selected drive for each pulse present on the line. The direction of motion is determined by the DIR line.
DIR	0	Direction - Moves head in if low, out if high, used in association with Step.
Hdsel	0	Head select - The side 1 (upper head) is selected when this line is low
WE	0	Write Enable - enables write current to the R/W head when active (low).
WD	0	Write Data - data to be written on disk
INDEX	I	Index - one pulse appears on this line each time the index hole is detected (i.e. once per revolution)
WP	I	Write protect - low when disk write

protected

TR0	I	Track 00 - low when head over track 0
Read Data	I	Data read from the disk (data with imbedded clock information)

APPENDIX C THE FLOPPY DISK CONTROLLER

The Floppy Disk Controller (F.D.C.) used in the I.B.M. type P.C.'s is the Intel 8272A and the NEC 765. These LSI devices are very complex and will only be described very briefly here. The F.D.C. interfaces to the central processing unit like most peripheral devices and makes full use of the interrupt and DMA resources available. The microprocessor is able to issue 15 'high level' commands to the F.D.C. These include read data, write data, format a track, read a track, seek, recalibrate, sense drive status etc. Typical parameters that are included with these commands are the cylinder number (track number), head number, sector number, number of data bytes in a sector, the last sector number on a track, the length of gap 3 etc.

Explanation of two of these commands will illustrate the functioning of the F.D.C.

Firstly the seek command. This command is issued before data is read from or written to a particular track. Although the cylinder number is one of the parameters included in the read or write command, the floppy disk controller does not automatically position the head over the required track. The seek command is therefore necessary to perform this function. When the seek command is issued the requested cylinder (track) number is compared to the value stored in the controller. Step pulses on the STEP line and the required level on the DIR line are asserted to position the head over the required track.

When a write command is issued, the F.D.C. begins to read the data passing the read/write head. This continues until the required sector I.D. field i.e. the sector I.D. field containing the required sector number is found. As is shown in Figure A2 in Appendix A the synchronisation 00's are re-written followed by the data for the sector as well as the first byte of

the next gap. These synchronisation bytes are rewritten each time a sector is rewritten so that when the sector is read, the floppy disk controller is properly synchronised with the incoming data field by the time the data field is encountered.

APPENDIX D DATA RECORDING TECHNIQUES

The recording technique used on a floppy - disk has to be such that it is capable of conveying data as well as clock information. The method used in double density recording as used on the P.C.'s under consideration is called MFM (Modified FM). FM is the technique used in single density recording systems. The FM technique will be described first and then the modifications will be discussed.

Each data bit can be considered to have a 'bit cell' associated with it. In the FM technique, a pulse is **always** present at the beginning of each bit cell (except for markers See later). This conveys the clock which is used by the floppy - disk controller circuitry to synchronise itself so that the data can be read correctly. If the data bit associated with a bit cell is a 1, another pulse is present in the middle of the bit cell i.e. there are two pulses in the bit cell. A 0 is therefore identified by the absence of a pulse in the middle of the bit cell.

In the MFM technique, the pulse at the beginning of the bit cell **NEED NOT** be present. The data bits are, as above, indicated by the presence or absence at the middle of the bit cell for a 1 or 0 bit respectively. When a 1 is followed by two or more 0 bits, the clock pulses at the start of the bit cell are re - inserted. This is necessary to maintain synchronisation.

If we consider a bit cell size of $4\mu\text{s}$ as encountered on the standard $5\frac{1}{4}$ inch floppy - disk interface, we can see the waveform structure after encoding in Figure D1. As can be seen, 3 fundamental time spacings between pulses become ev. ant. $4\mu\text{s}$ gaps are present between consecutive 1 bits and between 0 bits after the second 0 bit $8\mu\text{s}$ gaps occur when there are 01 or 10 bit combinations except for the case of 101 where the gap is $8\mu\text{s}$.

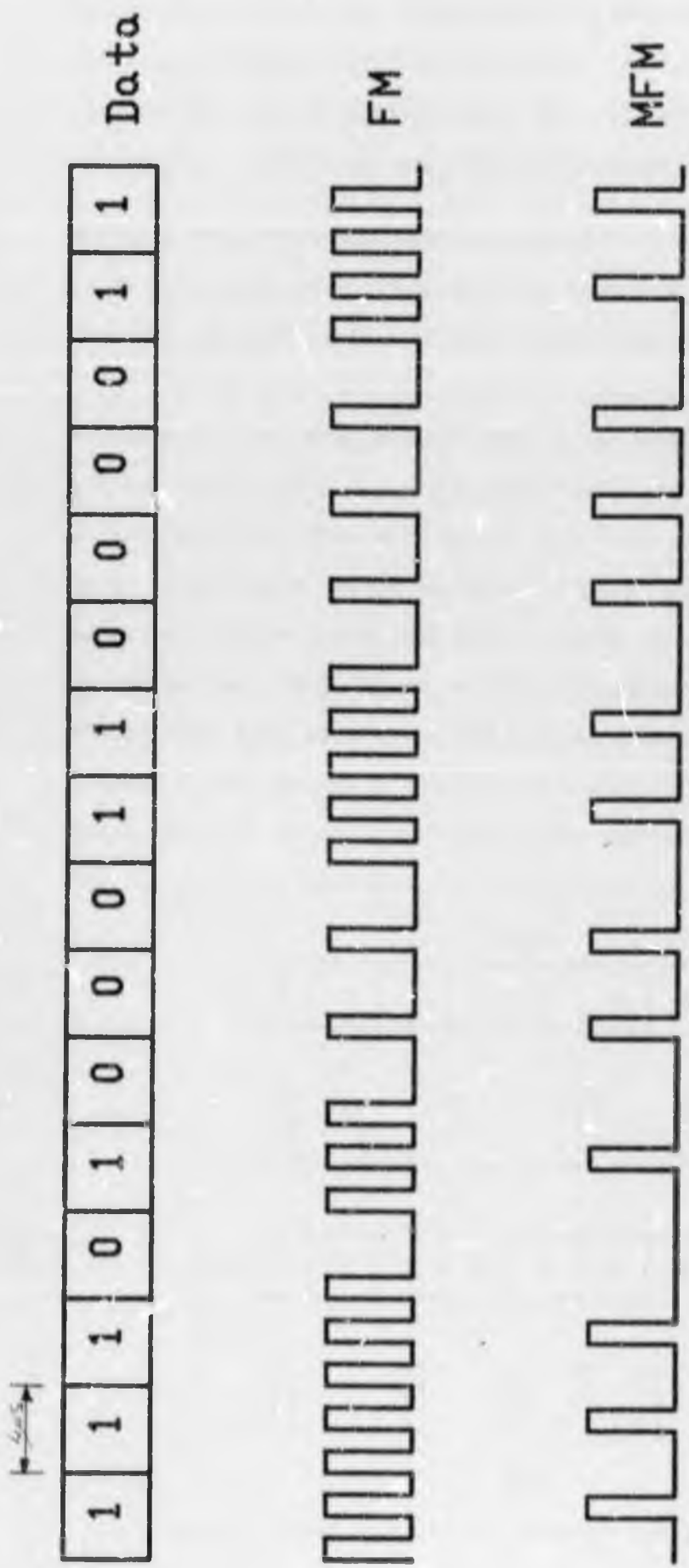


Figure D.1 Diagram Illustrating FM And MFM Encoding

The difference between single and double density is clearly evident in the figure. The bit cell in the single density case is twice the size of that in the double density name. The data per second ratio is therefore 2:1 between the two systems, hence the names.

Markers NOTE: This explanation applies to the double density technique. As we have seen, each track and each sector of each track has to have some form of identification. These identifiers require the generation of unique clock and data sequences to distinguish them from user data. A sequence of regularly encoded data bytes cannot be used for these identifiers since there is no guarantee that this data sequence may not occur in the user data. The user should also have no restriction on his data. To generate these unique markers, a clock bit is omitted. The modified bytes used in the index and address marks are C_{2H} and A_{1H} respectively. As can be seen from Figure D2 these bytes consist of an active data bit followed by four zero bits. This would result normally in pulses being present at the start of three consecutive bit cells (for the 3 zeros). The middle bit pulse is omitted (shown dotted) to make the byte unique. This is the only case where an 8μs gap occurs between 0 bits.

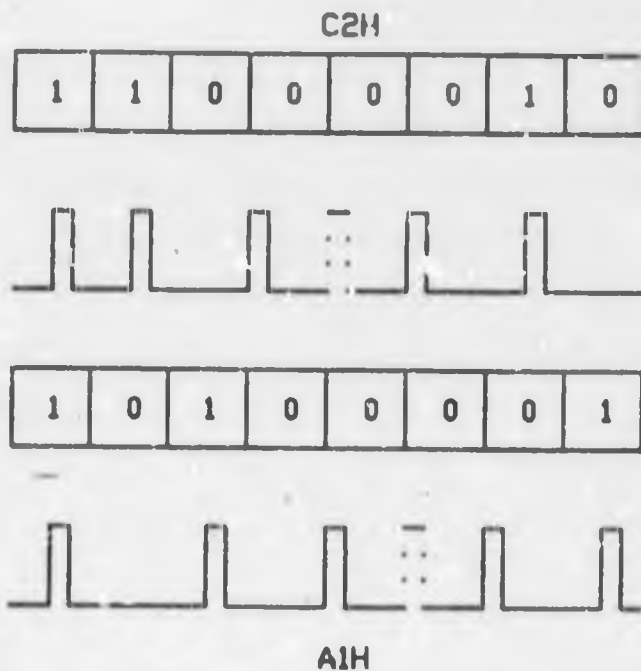


Figure D.2 Encoding Of The Address And I.D. Address Markers

Write Precompensation Peak shifting i.e. shifting of the position of the data pulses recorded on the disks is a characteristic of magnetic disks that results from interference of adjacent bit flux reversals which cause a flux reversal to be read slightly before or after its nominal time. Critical bit patterns are 011, 1000, 110 and 0001. If, for example, we consider the pattern 1101' the gap between the first two 1's is $4\mu\text{s}$, the second two i.e. 101 is $8\mu\text{s}$ and the last two $4\mu\text{s}$. The data pulses corresponding to the 1's on either side of the 0 appear to migrate towards each other thus reducing the $8\mu\text{s}$ gap. By writing the first of these pulses earlier than normal and the second later than normal, they appear to be in their correct positions when they are read. These early and late writes are performed by shifting the position of the data pulses 250ns either way of their nominal positions.

APPENDIX E THE DECODER CIRCUITRY

The function of this circuit is to take the data waveform WD i.e. write data, which would conventionally be written to the floppy disk, and retrieve the data and present it in a similar form to that which would be encountered on a synchronous communications link, in this case NRZ coding. It should be recalled that over and above the MFM encoding of the data, the wave-form has write pre-compensation incorporated as well. The decoder must also be capable of removing this compensation.

The circuit operates essentially according to the following principles:- The data waveform is 'sampled' to detect the data pulses being written to the floppy disk. These pulses are used to reset a counter circuit which obtains a certain count value in effect measuring the time between the pulses. The counter value is stored before the counter is reset. The counter value that has been stored is then, in effect, quantised thereby producing an active signal corresponding to one of the three possible nominal bit spacings i.e. 4, 6 or 8 μ s. This together with the previous data bit is used to calculate the present data bit. Clock information is also generated from the incoming data stream. The NRZ data stream is reversed on a byte by byte basis (See reason later) after being input to the Z8530 SCC (Synchronous Communications Controller) which performs serial to parallel conversion. The remainder of the circuitry is to cater for 'special case conditions' which generally occur at the beginning of the data stream.

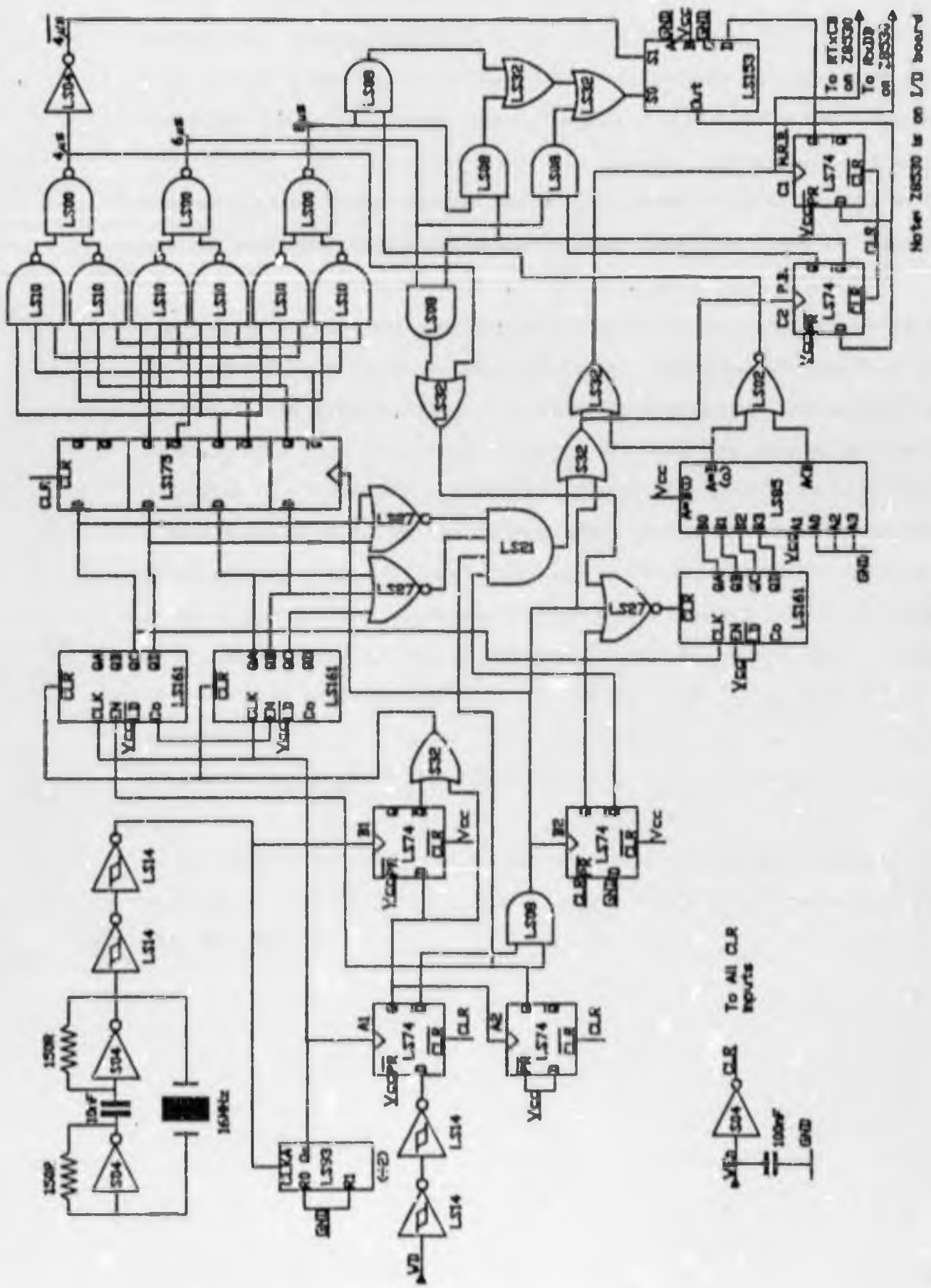


Figure E.1 The Decoder Circuit Diagram

Implementation details See Figure E1. The inputs to the circuit are the WD (Write Data) and WEn (Write Enable). These signals are obtained from the floppy disk interface after being buffered. A 16 MHz clock is obtained from a simple ring oscillator. The Write Enable (WEn) line is used as a master clear signal to keep the whole decoder circuit in a reset state when data is not being input. The pulses occurring on the Write Data (WD) decoder input are 250ns wide with nominal spacings of 4, 6 or 8µs between pulses. These spacings may vary by about 250ns either way of the nominal positions (Write Pre-compensation). The data input is 'sampled' at 8MHz to ensure detection of the pulses in accordance with the Nyquist sampling criterion. The sampling is performed by the D-type flip-flop A1. Note: The LS74 is positive edge sensitive. The input to this flip-flop is preceded by Schmitt-trigger inverters to ensure that the waveform has an acceptable form before the sampling process. The output of the A1 flip-flop is a 0 when WEn is inactive. The B1 flip-flop is used to provide a narrow reset pulse on the falling edges of the WD pulses to the LS161 pulse spacing counters. This occurs in the following manner:- The D input to B1 is high between pulses which results in a 0 output from the Q⁺ output of B1. When a negative edge occurs, the D input is now a 0. The two inputs to the S32 OR gate are both 0 until the B1 latch is clocked. (Clocking at 16MHz). When this occurs the latch output is set to a 1 thus terminating the reset pulse. See Figure E2.

If the WD line is monitored at the beginning of a write operation, the waveform shown in Figure E3. The beginning of the write operation occurs when WEn goes low.

10 nS/SAMPLE

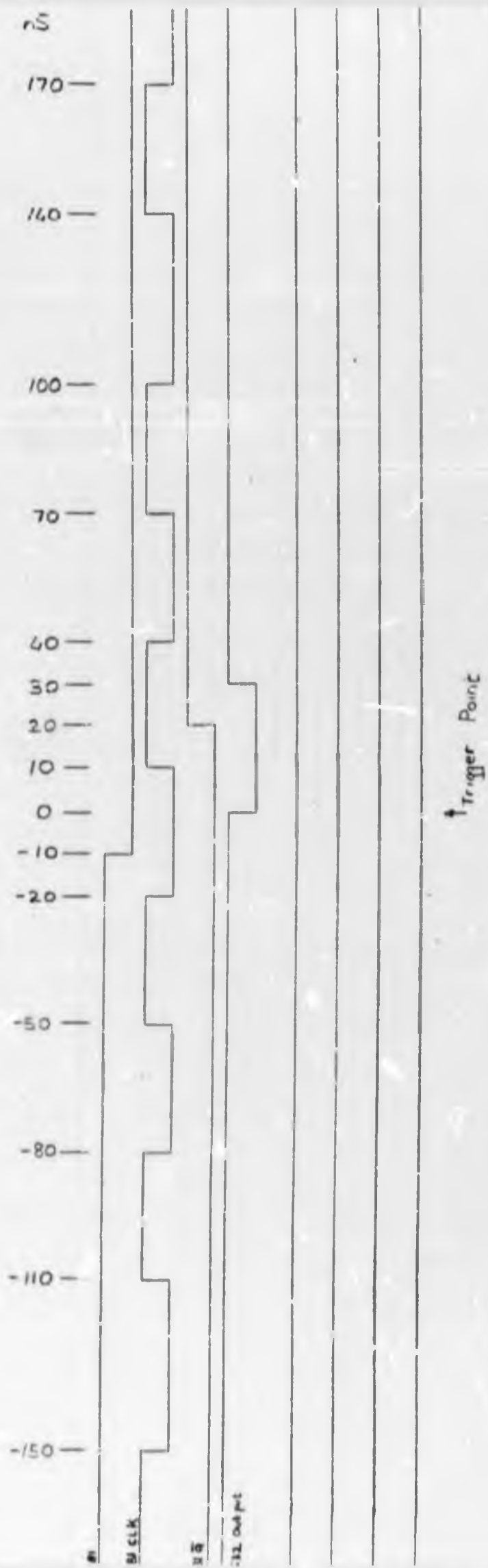


Figure E.2 Timing Diagram Of The Counter Preset Circuit

50ns/sample

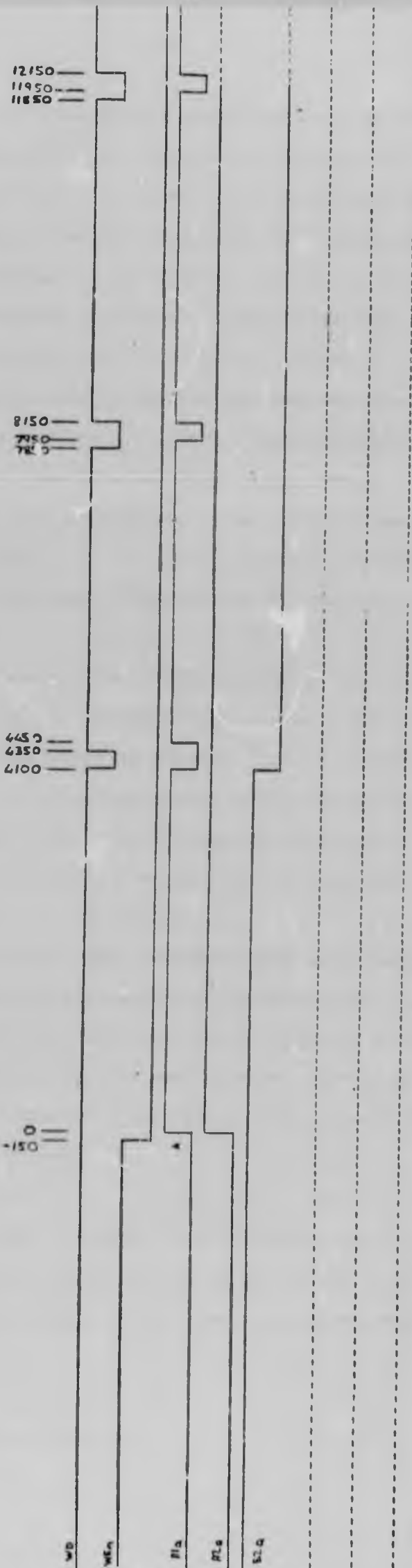


Figure E.3. Timing Diagram Of Write Data At The Start Of
A Write Operation

As was described above, the LS161 counters are reset on the falling edges of the sampled WD waveform i.e. The Q output of latch A1. This leads to an unacceptable situation at the start of the WD pulse train. The counters begin to count while the sampled WD remains low. The result is that the first count value latched on the falling edge of the first pulse is too large. The required period is from the first rising edge (pt A in Figure E3) to the falling edge of the first pulse. For this reason flip-flop A2 is included. This is used to disable the counters until a rising edge is encountered in the waveform. The output then keeps the counters enabled until the WEn line is deactivated. Flip-flop B2 keeps the third counter (see function later) in a cleared state and inhibits the pulse expander circuit (see later) until it is clocked. Latch B2 is clocked whenever a pulse occurs on WD and latch A2 has enabled the circuit.

The pulse spacing counters are clocked at 8MHz. The outputs of these counters form the input to two further sections of the circuit viz. the counter latch LS175 and the pulse expander (LS27's and LS21). The latter circuit is used to provide a high output on the LS21 output when all five counter outputs shown in the circuit diagram are all zero. This only occurs when the counter has been cleared (i.e. falling edge of WD). The WD pulses are only 250ns wide. By combining the counter's outputs whose least significant bit changes at 1MHz, and ORing this with the 250ns pulses, a more symmetrical clock is generated for the Z8530 SCC.

The data latch is used to store the counter value while the serial data output is being generated and the next counter value is being generated. The counter value is latched in on the falling edge of WD. i.e. just before the counters are reset.

As can be seen from Table 1, count values ranging from 24 to 39 make the 4µs output active while values from 40 to 55 and 56 to 71 make the 6µs and 8µs outputs active respectively. These outputs are obtained by gating

the outputs of the LS175 latch using the LS00's and LS10's. The use of these signals will be described shortly.

Flip-flops C1 and C2 are used to store the 'Most Recent Bit' (M.R.B.) produced by the circuit and the 'Previous Bit' (P.B.) respectively.

The 'previous bit' and the 4, 6 and 8 μ s pulse width signals are used to control the serial data generator according to the following conditions:-

Note: The 'previous bit' is in brackets. Bold face characters indicate where data pulses are present.

CONDITIONS	DATA
If 'previous bit' = 1 and 8 μ s is active	(1)01 (A)
= 0 and 8 μ s is active	(0)00 (B)

(Note: This second case is an exceptional case and only occurs in markers.
See Appendix D on Data Recording Techniques)

If 'previous bit' = 1 and 6 μ s is active	(1)00 (C)
= 0 and 6 μ s is active	(0)1 (D)

If 'previous bit' = 1 and 4 μ s is active	(1)1 (E)
= 0 and 4 μ s is active	(0)0 (F)

As can be seen from the above, cases A to C require an extra clock pulse to be generated since there are more data bits than pulses from the WD line.

10ns/sample

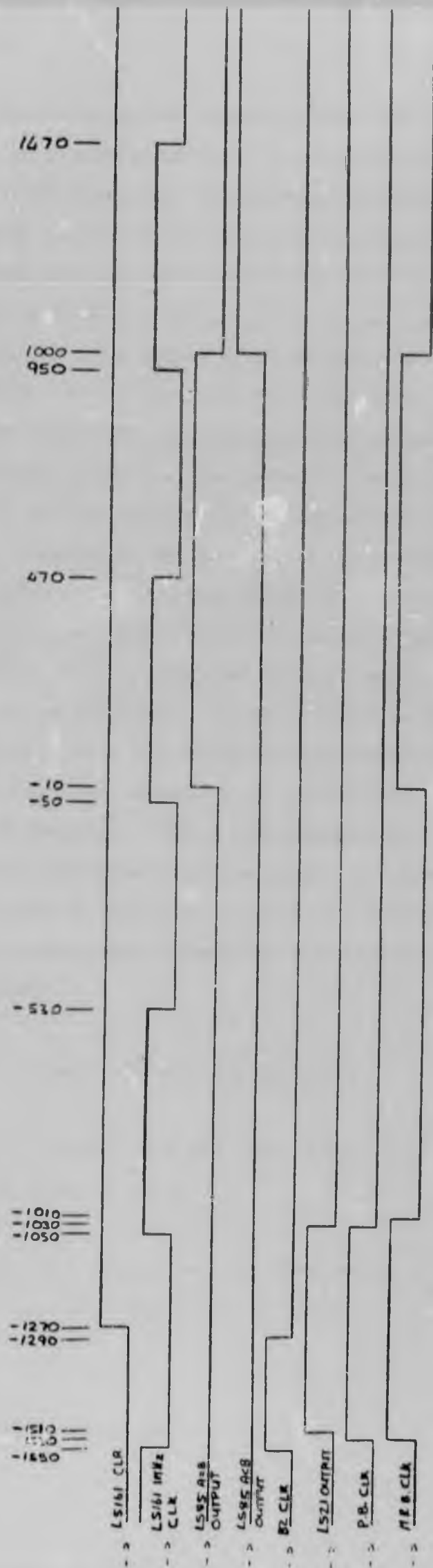


Figure E-6 Timing Diagram Of Clock Pulses Showing Expanded Pulses And Added Clock Pulses

On the output of the MFM decoder we require a clock edge for EACH data bit. To cater for these situations the third LS161 counter is used along with the LS85 magnitude comparator. This counter is cleared by each pulse in WD and the counter is kept in the cleared state when cases D to F are present. These cases are defined by (4 μ s being active) OR (P.B.=0 AND 6 μ s is active). In cases A to C this LS161 clocks at 1 MHz. During the second count period, the A = B output is active due to the wiring on the A inputs of the LS85. It is necessary that this output should become active on such a low count value since all possible bit combinations must be able to be processed in 4 μ s i.e. the shortest inter pulse gap. If this was not the case it may be possible that a new counter value would be latched before the previous one had been completely processed.

The 4, 6 and 8 μ s pulse width signals, the A < B output of the LS85 and the 'present bit' value are used to drive the select lines of the LS153 (a 1 of 4 multiplexor). This is connected in such a way so as to generate the required data bit on the output. The flip-flop C1 described above is clocked on every clock pulse at the output and thus stores the most recent data bit. The C2 flip-flop, meanwhile, is clocked only when there is a data pulse on the WD line. (i.e. stores the present bit).

The derivation of the next data bit in the serial data generator may have been done using fundamental gating but the use of a multiplexor made the design easier. The requirements of the circuit are shown below. The following should be noted:-

1. The 4, 6 and 8 μ s signals are mutually exclusive.
2. The 'Present Bit' differs from the 'Most Recent Bit' only when an additional clock pulse is added.
3. $\overline{(A=B)} + (A \cdot B)$ gives a 0 output when the clock pulse is generated and stays in this state until the counter is reset.

(A=B)+(A<B)	P.B.	4µs	6µs	8µs	Required output.
0	0	0	0	1	0
0	1	0	0	1	1
1	0	0	0	1	0
1	1	0	0	1	0
0	0	0	1	0	Doesn't occur
0	1	0	1	0	0
1	0	0	1	0	Doesn't occur
1	1	0	1	0	0
0	0	1	0	0	Doesn't occur
0	1	1	0	0	Doesn't occur
1	0	1	0	0	M.R.B. (=P.B.)
1	1	1	0	0	M.R.B. (=P.B.)

The 4µs line defines a unique requirement and can therefore be connected directly (after inversion) to one of the multiplexor select lines. The other select line is controlled by the following equation:-

$$S_0 = 8\mu s \bar{0} P.B. \bar{0} X + 8\mu s \bar{0} P.B. 0 X + 8\mu s 0 P.B. \bar{0} X + 6\mu s 0 P.B. \bar{0} X + 6\mu s 0 P.B. 0 X$$

This can be simplified to

$$S_0 = 8\mu s \bar{0} P.B. + 8\mu s 0 X + 6\mu s 0 P.B.$$

Where $X = (A=B) + (A<B)$ The multiplexor inputs on the circuit diagram correspond to the following select inputs:-

S ₁	S ₀	Input
0	0	D
0	1	C
1	0	B
1	1	A

The data input to the Z8530 SCC is obtained from the output of the M.R.B. latch (C1). The SCC clock consists of the WD pulses that are expanded as well as the additional clock pulses from the LS85. Examples are shown in Figure E4. The SCC is synchronised onto the NRZ data stream using synchronisation characters in the data stream (See Appendix O on Emulation Software). The SCC converts the serial data into parallel form for the microprocessor system. In the I/O circuitry it should be noted that the data is reversed i.e. the most significant bit is made the least significant etc. since the data written to the disk is written most significant

bit first and the SCC expects the least significant bit first. This has more serious ramifications for the Encoder circuitry and will be explained more fully there.

The MFM decoder performs a vital function in the emulator by converting the write pre-compensated MFM encoded waveform into an NRZ coded waveform suitable for a SCC. Each building block of the decoder is simple in isolation but are combined to perform a complicated task.

TABLE 1: Counter Values

Count	Q _{B2}	Q _{A2}	Q _{D1}	Q _{C1}	Q _{B1}	Q _{A1}
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	1	0
7	0	0	0	0	1	1
8	0	0	0	0	0	0
9	0	0	0	0	0	1
10	0	0	0	0	1	0
11	0	0	0	0	1	1
12	0	0	0	0	0	0
13	0	0	0	0	0	1
14	0	0	0	0	1	0
15	0	0	0	0	1	1
16	0	0	0	0	0	0
17	0	0	0	0	0	1
18	0	0	0	0	1	0
19	0	0	0	0	1	1
20	0	0	0	0	0	0
21	0	0	0	0	0	1
22	0	0	0	0	1	0
23	0	1	0	1	1	1
24	0	1	1	0	0	0
25	0	1	1	0	0	1
26	0	1	1	0	1	0
27	0	1	1	0	1	1
28	0	1	1	1	0	0
29	0	1	1	1	0	1
30	0	1	1	1	1	0
31	0	1	1	1	1	1
32	1	0	0	0	0	0
33	1	0	0	0	0	1
34	1	0	0	0	1	0
35	1	0	0	0	1	1
36	1	0	0	1	0	0
37	1	0	0	1	0	1
38	1	0	0	1	1	0
39	1	0	0	1	1	1
40	1	0	1	0	0	0
41	1	0	1	0	0	1
42	1	0	1	0	1	0
43	1	0	1	0	1	1
44	1	0	1	1	0	0
45	1	0	1	1	0	1
46	1	0	1	1	1	0
47	1	0	1	1	1	1
48	1	1	0	0	0	0
49	1	1	0	0	0	1
50	1	1	0	0	1	0
51	1	1	0	0	1	1
52	1	1	0	1	0	0
53	1	1	0	1	0	1
54	1	1	0	1	1	0
55	1	1	0	1	1	1
56	1	1	1	0	0	0
57	1	1	1	0	0	1
58	1	1	1	0	1	0
59	1	1	1	0	1	1
60	1	1	1	1	0	0
61	1	1	1	1	0	1
62	1	1	1	1	1	0
63	1	1	1	1	1	1
64	0	0	0	0	0	0
65	0	0	0	0	0	1
66	0	0	0	0	1	0
67	0	0	0	0	1	1
68	0	0	0	1	0	0
69	0	0	0	1	0	1
70	0	0	0	1	1	0
71	0	0	0	1	1	1
72	0	0	1	0	0	0



The following gating is required to get these signals --
 Note The Q's have been dropped.

$$4\mu s = D1 \cdot A2 \cdot \overline{B2} + \overline{D1} \cdot \overline{A2} \cdot B2$$

$$6\mu s = D1 \cdot \overline{A2} \cdot B2 + \overline{D1} \cdot A2 \cdot \overline{B2}$$

$$8\mu s = D1 \cdot A2 \cdot B2 + \overline{D1} \cdot \overline{A2} \cdot \overline{B2}$$

These are implemented as:-

$$4\mu s = \overline{\overline{D1 \cdot A2 \cdot \overline{B2}} \cdot \overline{\overline{\overline{D1} \cdot \overline{A2} \cdot B2}}}$$

$$6\mu s = \overline{\overline{D1 \cdot \overline{A2} \cdot B2} \cdot \overline{\overline{\overline{D1} \cdot A2 \cdot \overline{B2}}}}$$

$$8\mu s = \overline{\overline{D1 \cdot A2 \cdot B2} \cdot \overline{\overline{\overline{D1} \cdot \overline{A2} \cdot \overline{B2}}}}$$

APPENDIX F THE ENCODER CIRCUITRY

The encoder circuitry is required to perform the process of providing data in such a form to the floppy disk interface of the computer so that it appears to be receiving this data from the disk itself. As the problem is examined more closely we can see that this requires extensive designing in both the hardware and software fields since they have a very close association.

Requirements It must be remembered that following the initial formatting of a disk for use, the floppy disk controller expects to find all the data recorded during formatting on subsequent disk read operations. For this reason ALL the data, i.e. including that between sectors needs to be simulated. The waveform present on the read data line from the disk-drive to the floppy disk controller was examined using a logic analyser to ascertain the exact characteristics of the wave-form. If this wave form could be reproduced as accurately as possible, the floppy disk controller must surely be convinced that it was in fact receiving the data from a disk and thus function as required.

Implementation The wave form consists of an MFM encoded data stream. The pulse width was measured at 1 μ s with nominal pulse spacing of 4, 6 and 8 μ s when measured from the beginning of one pulse to the beginning of the next. The voltage levels encountered were typical TTL levels as expected. The data bytes are received MOST significant bit first. On the 5 1/4 inch disk drives as presently found on the IBM PC type computers, the average bit rate is 250 kbits per second.

A Serial Communications Controller (SCC) is used to provide a number of fundamental functions viz. parallel to serial conversion and CRC calculation. An important fact must be emphasised at this stage. In a typical data communication system using this or similar devices, the data is

transmitted LEAST significant bit first. It is immediately evident that this is in direct conflict with what was stated above. As shown in the circuit for the I/O devices (See Appendix H) it is possible to reverse the data being sent or read from the SCC. The CRC on the data that is written by the Floppy Disk Controller is calculated on the data most significant bit first. The SCC performs the calculation on the least significant bit first. The data must be reversed BEFORE the CRC calculation.

At this stage we may assume that we have a serial data stream using NRZ encoding but with the most significant bit of the byte transmitted first. This signal then just has to be encoded in the MFM format. The principal behind the encoder used here is the following:- Each bit can be considered to have a 'bit cell'. In this case the length of this bit cell is 4 μ s since the bit rate is 250k bits/s. In each bit cell there may be a pulse of 1 μ s duration that may occur either at the beginning or the middle of the bit cell i.e. either during the first or the third microsecond of the bit cell. See Figure F1.

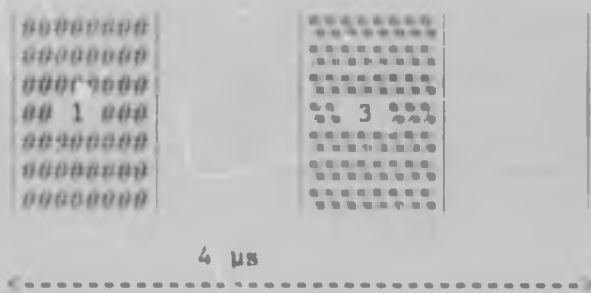


Figure F1: Bit Cell Structure. Pulses May Occur During The First Or Third μ s.

A counter circuit is used to provide one μ s pulses at both the first and third μ s points. Using the values of the present and of the previous bits, these pulses are gated thus producing the required 4, 6 and 8 μ s pulse spacing according to the MFM encoding technique. This then takes care of

the encoding of most of the data. The only other encoding that is required is that of address markers. It should be recalled that the address markers (C2H - Index address mark and A1H I.D. address mark) consist of one active data bit followed by four zero bits. See Figure F2. As can be seen from the figure, the four zero bits in the markers generate three pulses at the start of three consecutive bit cells. The markers are identified by leaving the middle of these three pulses out. The circuit used generates the three pulses as would be the case for a normal data byte but the second of the three is suppressed. Therefore all the encoding requirements are met.

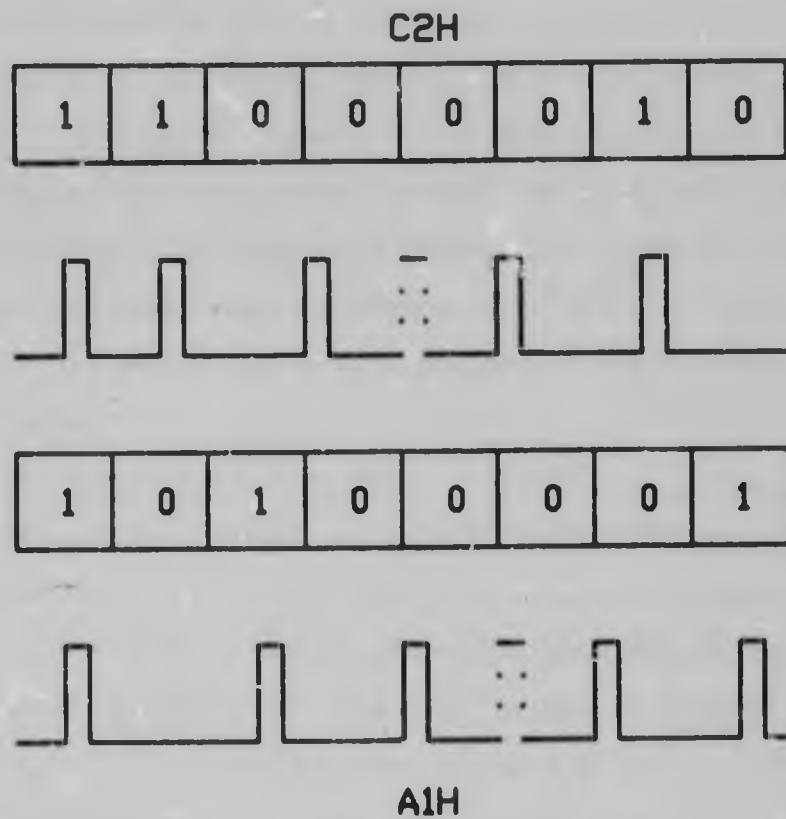


Figure F.2 Encoding Of The Address And I.D. Address Markers

Detailed Description of Encoder Circuit Operation See Figure F4. A 2 MHz symmetrical clock is derived from the 8MHz CPU clock and is used to drive the LS161 synchronous counter. The counter is permanently enabled but has its clear input controlled by the inverted RTS line from the SCC. This is used to keep the output of the counter zero when required. The RTS line is also used to keep the other flip-flops in a cleared or preset state as required. The outputs of the counter provide clocks at 1MHz, 500kHz and 250kHz. The 250kHz is used to provide the fundamental data rate while the other two are used in the encoding process. The output waveforms of the counter are shown in Figure F3.

The data is clocked out of the Z8530 (SCC) on the falling edge of the 250kHz clock which is latched into the LS74 data latches by the rising edge of the 'latch load' signal. This signal is obtained by ANDing the 250kHz, 1MHz and inverted 500kHz signals. The 'start cell' and 'mid-cell' signals are produced by ANDing the 500kHz signal with the inverted 250kHz signal and the 250kHz signal respectively. Gating of these two signals according to the serial bit stream provides a simple method to perform the MFM encoding.

Consider the following bit patterns:-

Note:- Q_i denotes the present and $Q_{(i-1)}$ denotes the previous bit

- a. $Q_i = 0$ and $Q_{(i-1)} = 0$, the 'start cell' signal is allowed through since the output of the NOR gate is a 1.
- b. $Q_i = 1$ and $Q_{(i-1)} = 0$, the 'mid-cell' signal is allowed through since the NOR output is now a 0.
- c. $Q_i = 0$ and $Q_{(i-1)} = 1$, in this case neither the 'mid-cell' or the 'start cell' signals is allowed through since the Q_i output is zero thus disabling the 'mid-cell' signal and the NOR output is zero, thus disabling the 'start cell' signal.
- d. $Q_i = 1$ and $Q_{(i-1)} = 1$, here the 'mid-cell' signal is passed through since the Q_i output is a 1 and the NOR output is zero.

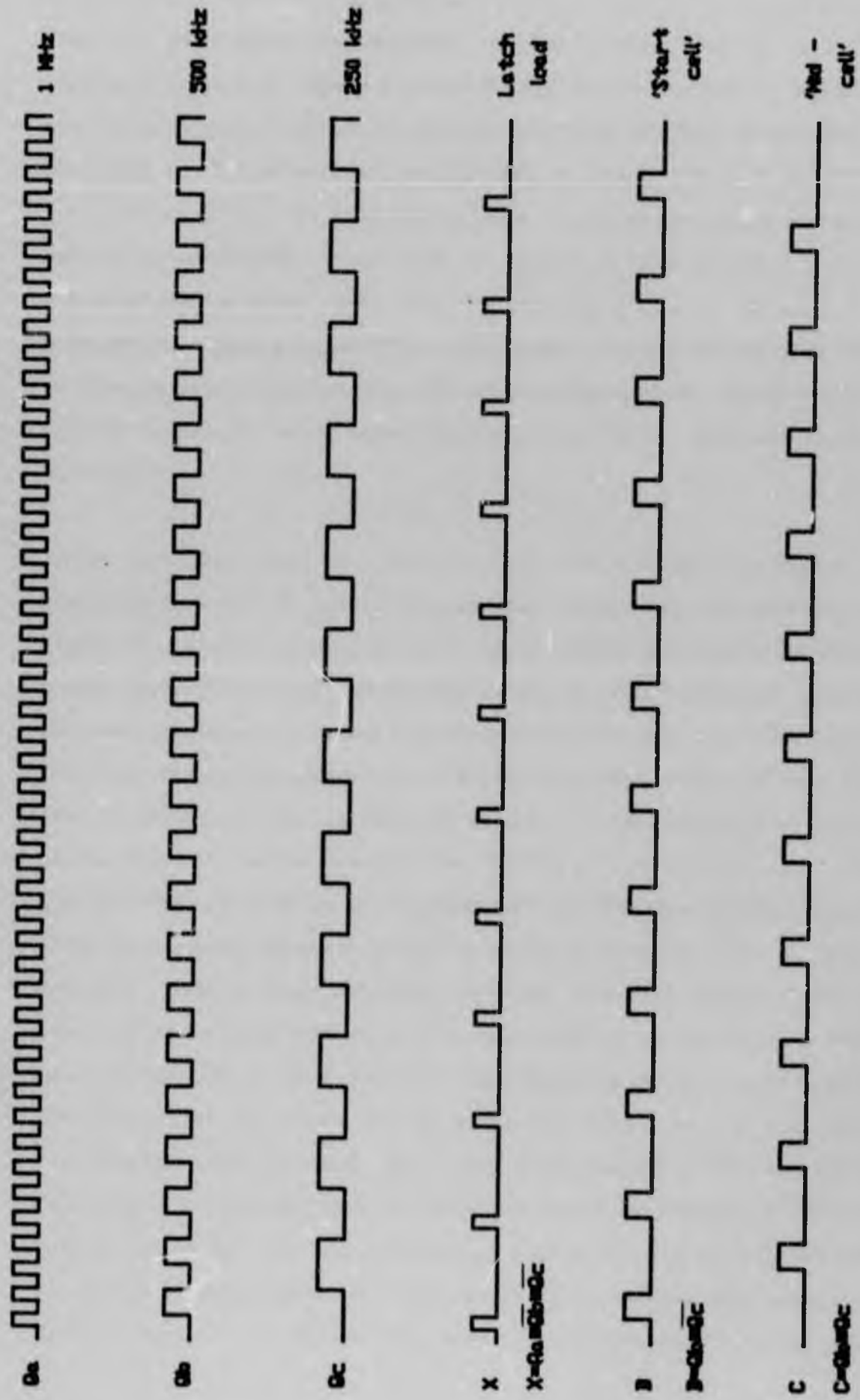


Figure F.3 Counter Output Waveforms

If we consider some three bit, bit sequences the operation may become more clear. If we consider the sequence 101 for example, when Q_i is a 1 the 'mid-cell' signal is always allowed through as can be seen from cases b and d above, thus a pulse in the middle of the bit cell is present for the first 1. In the next bit period then we have $Q(i-1) = 1$ and $Q_i = 0$ i.e. case c above. No pulse is allowed at either the start of the bit cell or in the middle. In the next bit period we have $Q(i-1) = 0$ and $Q_i = 1$ i.e. case b above. Here only the mid cell pulse is allowed. If we consider the spacing between the pulses, we can see that the required 8 μ s spacing is present between the two mid cell pulses. Using a similar process it can be easily shown that the 6 and 4 μ s gaps are produced correctly.

As was described above the generation of address markers requires that certain pulses be left out of the encoded stream. As can be seen from Figure F2 the pulses that are left out to identify the bytes as markers always occur at the start of the bit cells. An extra piece of circuitry has been introduced to remove 'start cell' pulses when this is required. The heart of this circuit is the J-K flip-flop. The output of this flip-flop is ANDed with the 'start cell' signal i.e. the signal is suppressed if the flip-flop output is a 0. The flip-flop is held in a preset state i.e. $Q=1$ when the encoder is disabled (via the DTR line and AND gate) and it is also preset whenever a midcell pulse is produced (i.e. a 1 bit is encoded). This is done to ensure that the flip-flop always allows the required 'start cell' bits to pass through until it is necessary to remove these pulses. The J and K inputs to the flip-flop are both tied high and thus the output toggles on each applied clock. (Negative edge sensitive). What happens then (assuming the 'start cell' stream to be clocking the flip-flop i.e. the AND gate on the clock input by-passed) is that the output, initially 1 and thus allowing passage of the signal, changes state to 0 and thus prevents the next pulse from reaching the output. The falling edge of this pulse does, however, restore the flip-flop output

to the 1 state. It is immediately evident therefore that this flip-flop must only be allowed to operate during address marker generation. This is done via the remaining D-type flip-flop whose output is held at 0 by the DTR line of the Z8530 that controls the clear input of the flip-flop. The output is only allowed to attain the 1 state when it is clocked by a 1 in the data stream from the Z8530. This is advantageous since the DTR line can be set as soon as the address mark data is written to the Z8530 without having to worry about the delays encountered in the encoder circuit. A necessary condition for this to work is that there are no 1 bits in the data stream between the DTR line being asserted and the address mark data arriving at the data latches. This condition is satisfied since the address markers are always preceded by 12 00H bytes. See Appendix A. The J-K flip-flop needs to be reset for each address mark byte since the output would toggle to 0 on the first pulse, toggle to 1 on the second and back to 0 on the third. It must, however be in the 1 state at the start of the next address mark byte. Care must be taken when writing the software that controls the DTR line since this line must remain active until all three address markers have passed through the encoder. The inclusion of the flip-flop chain on the J-K preset line ensures that only three address markers can be generated. The DTR line can be left in the active state longer thus making the software less critical. After the three markers have been generated the flip-flop is forced into the preset state.

The waveform available from the MFM encoder is buffered before being applied to the floppy disk interface. The waveform produced by the circuitry is identical to that obtained from a floppy disk drive unit. All that is required is the software to provide the data expected by the Floppy Disk Controller and to control the SCC control lines. This is described in Appendix P.

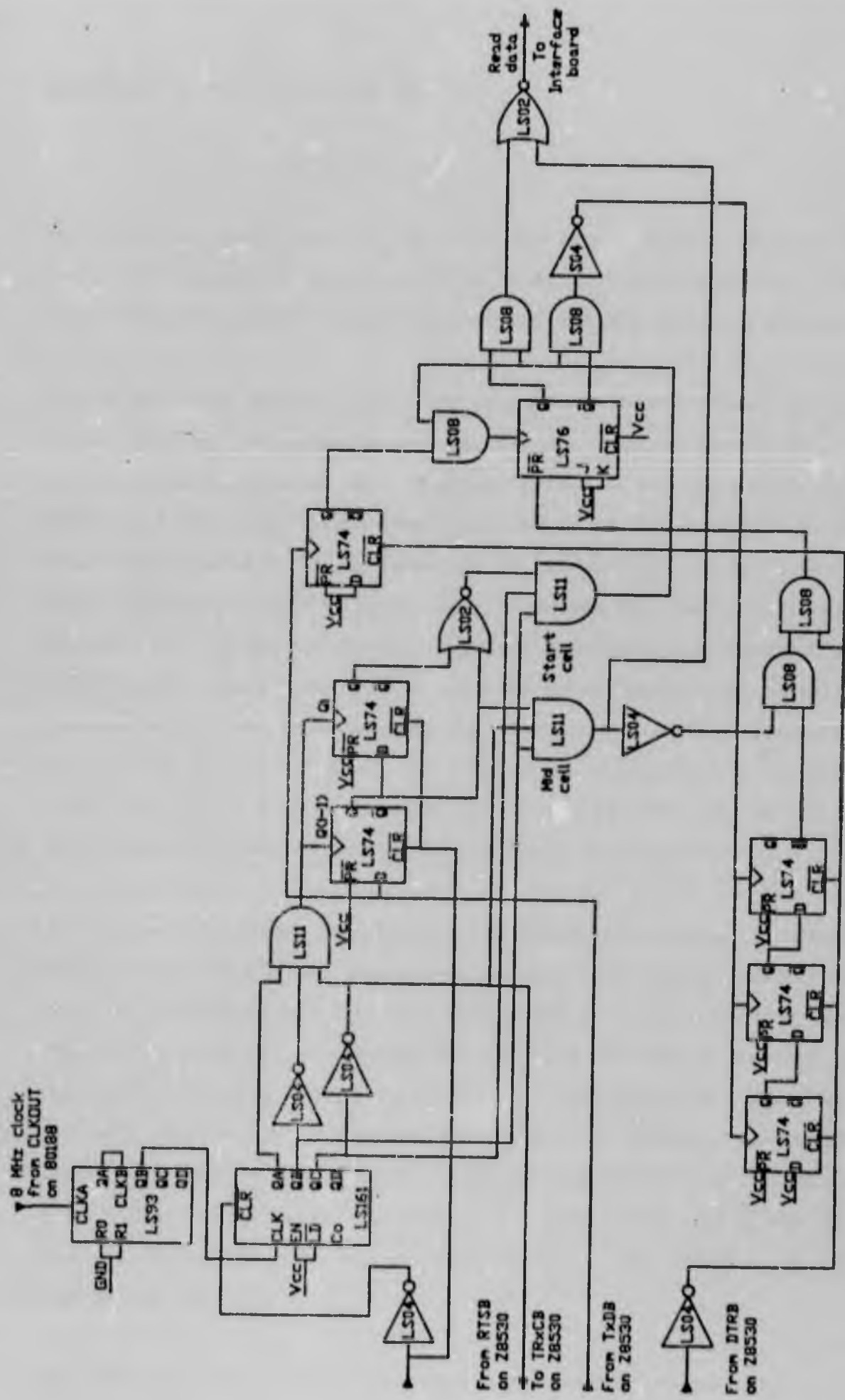


Figure F4 The Encoder Circuit Diagram

APPENDIX G THE INTERFACE BOARD

The interface board contains the input and output buffers required to interface the emulator circuitry to the floppy disk interface cable. The track counter to monitor the head movement is included in this circuitry.

Most of the lines on the interface cable are 'bus' signals except for the select lines for the individual drives. All the lines are however driven by open collector drivers. All the inputs therefore require pull-up resistors and 150 Ω resistors are used since this is the value used on normal disk drive units. All the inputs except the select input are buffered by D-type latches. In this way the state of the bus is retained when the emulator is no longer selected. The select line itself is inverted to enable these latches as well as to cause the output NAND drivers to drive the bus. Most of the signals sources and destinations are self explanatory but some are worthy of mention. The Index pulse is generated by software which controls the ITR line of channel A of the Z8530 SCC. The WP signal is permanently disabled by grounding the input. See Figure G1.

The head motion is monitored by an 8 bit up/down counter that is driven by the DIR and STEP lines. Step pulses are generated for each track that is to be traversed. The DIR line determines in which direction the counting takes place i.e. towards the centre or outside of the disk. A low on DIR causes the count value to increase (equivalent to inward head movement). The counter outputs are ORed together to generate the TRACK0 signal which, as its name implies, is active when the counter's value is zero. A simple port allows the counters' outputs to be read by the CPU when the PCS3 and RD lines are low simultaneously. Note:-PCS3 is an I/O chip select line from the 80188 CPU.

This board provides a simple yet critical function in the emulator.

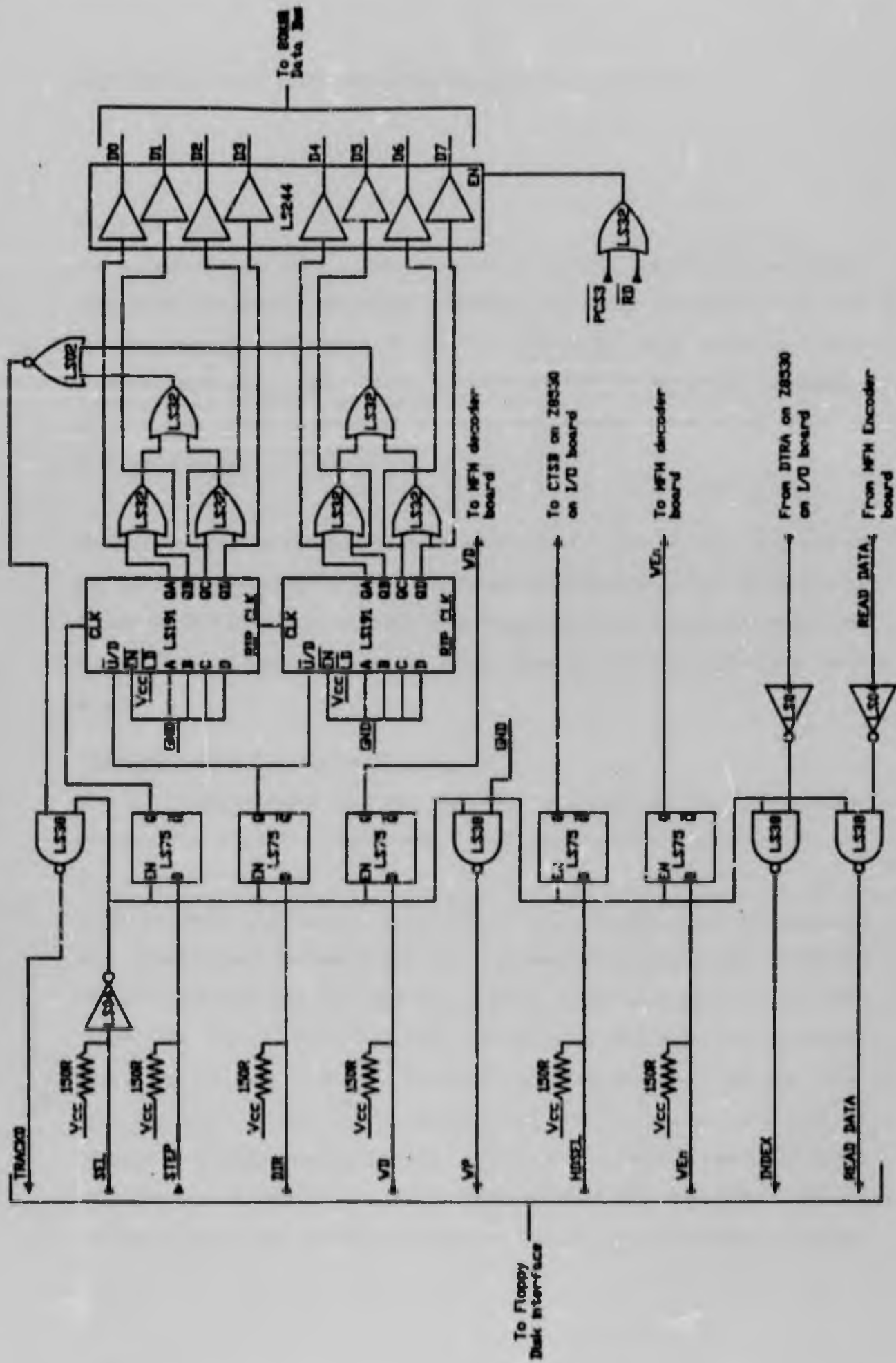


Figure G.1 The Interface Circuit Diagram

APPENDIX H THE MICROPROCESSOR CONTROLLER UNIT

See Figure H1.

The microprocessor unit's functions are to control the network and floppy disk drive emulation. The 80188 microprocessor forms the core of the unit and is supported by 2 Z8530 SCC's, a parallel input port (tri-state buffer), a parallel port (latch) as well as 256K of RAM which is expandable to just under 1 Mbyte in 256Kbyte increments. 8Kbytes of EPROM is also allowed for.

The microprocessor controller unit consists of 3 boards viz. CPU, memory and I/O boards and each will be discussed individually as far as possible. It should be noted however that eventhough the term boards has been used this refers to logical divisions rather than physically separate circuit boards.

The CPU board See Figure H2.

The only requirements for the internal oscillator of the 80188 microprocessor is a 16MHz crystal and 2 20pF capacitors as specified by the manufacturer. The 16MHz crystal provides an 8MHz processor clock and an 8 MHz 50% duty cycle clock on the CLKOUT line. The RES input is connected to a conventional RC reset circuit. The component values are $R=1K\Omega$ and $C=22\mu F$ which keep RES low long enough after power-up to satisfy the reset input requirements. The low to high transition on RES must occur no sooner than 50 μs after power-up[4]. The RESET output indicates that the 80188 is being reset internally, is synchronised with the system clock and can be used as a master reset. In this case it is only used to reset the Z8530 SCC. The unused inputs viz. HOLD, SRDY, ARDY, TEST, INT1, NMI, IT2 and IT3 are tied to Vcc or GND depending on their respective inactive states.

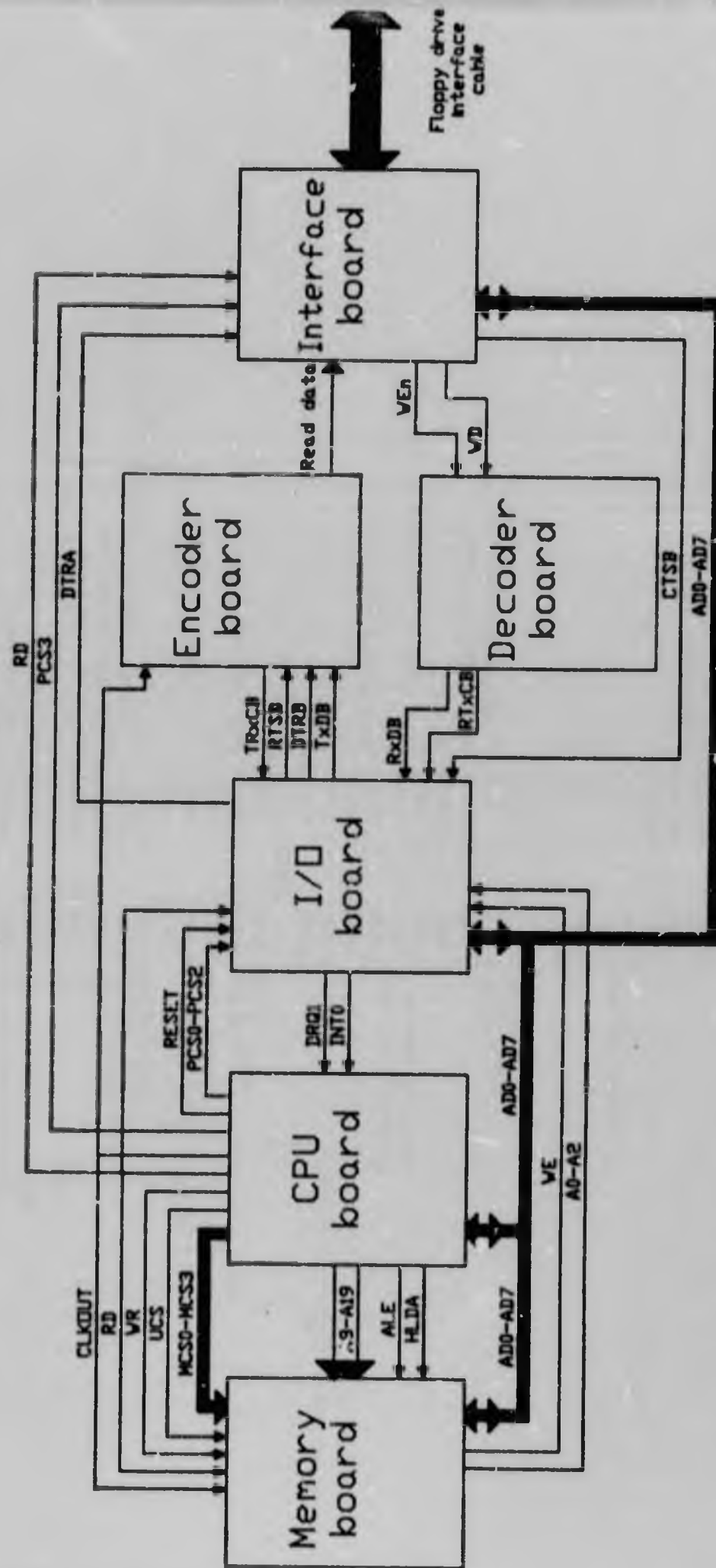


Figure H.1 Block Diagram Of The Floppy Disk Drive Emulator

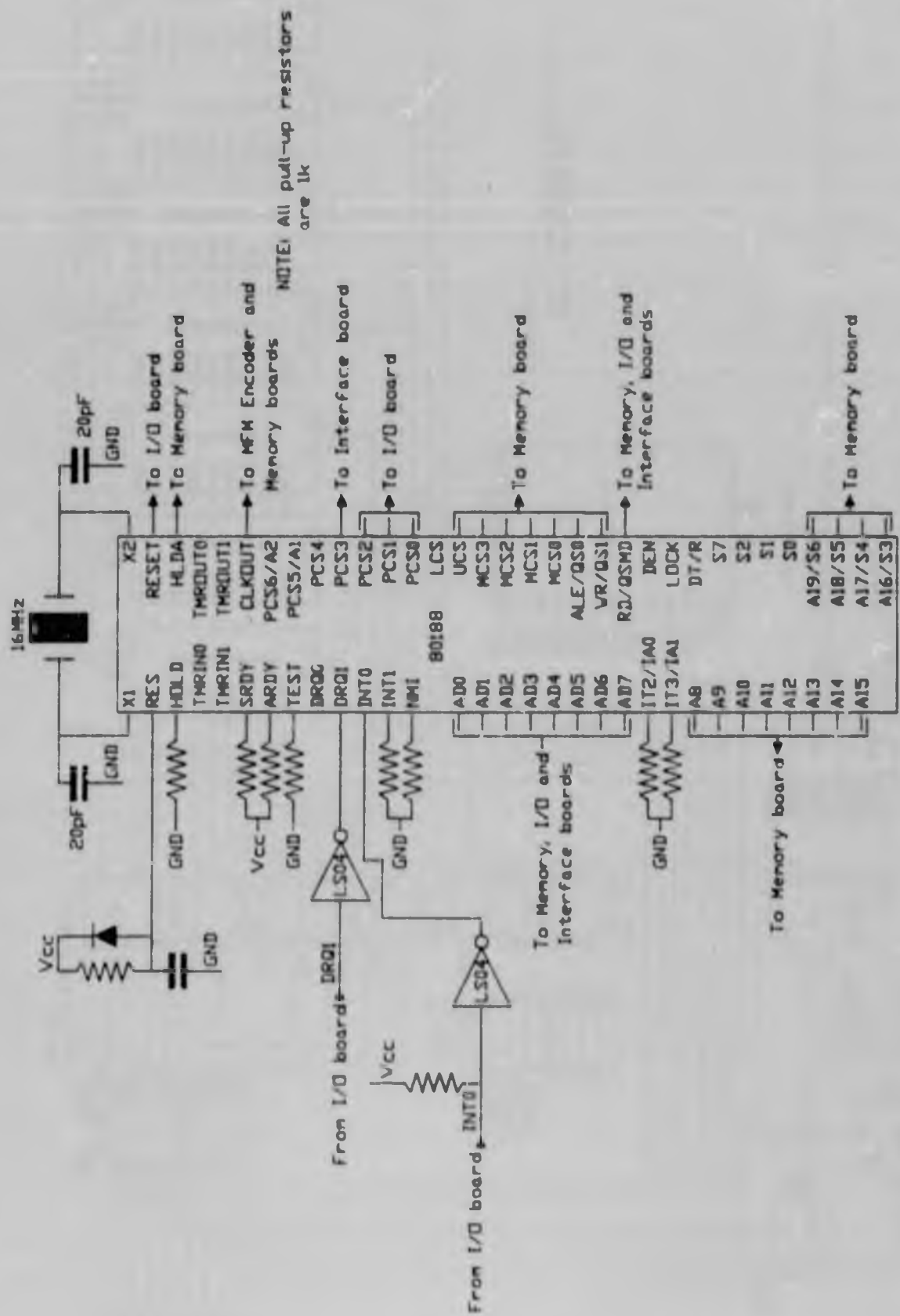


Figure H.2 The 80i88 CPU Circuit Diagram

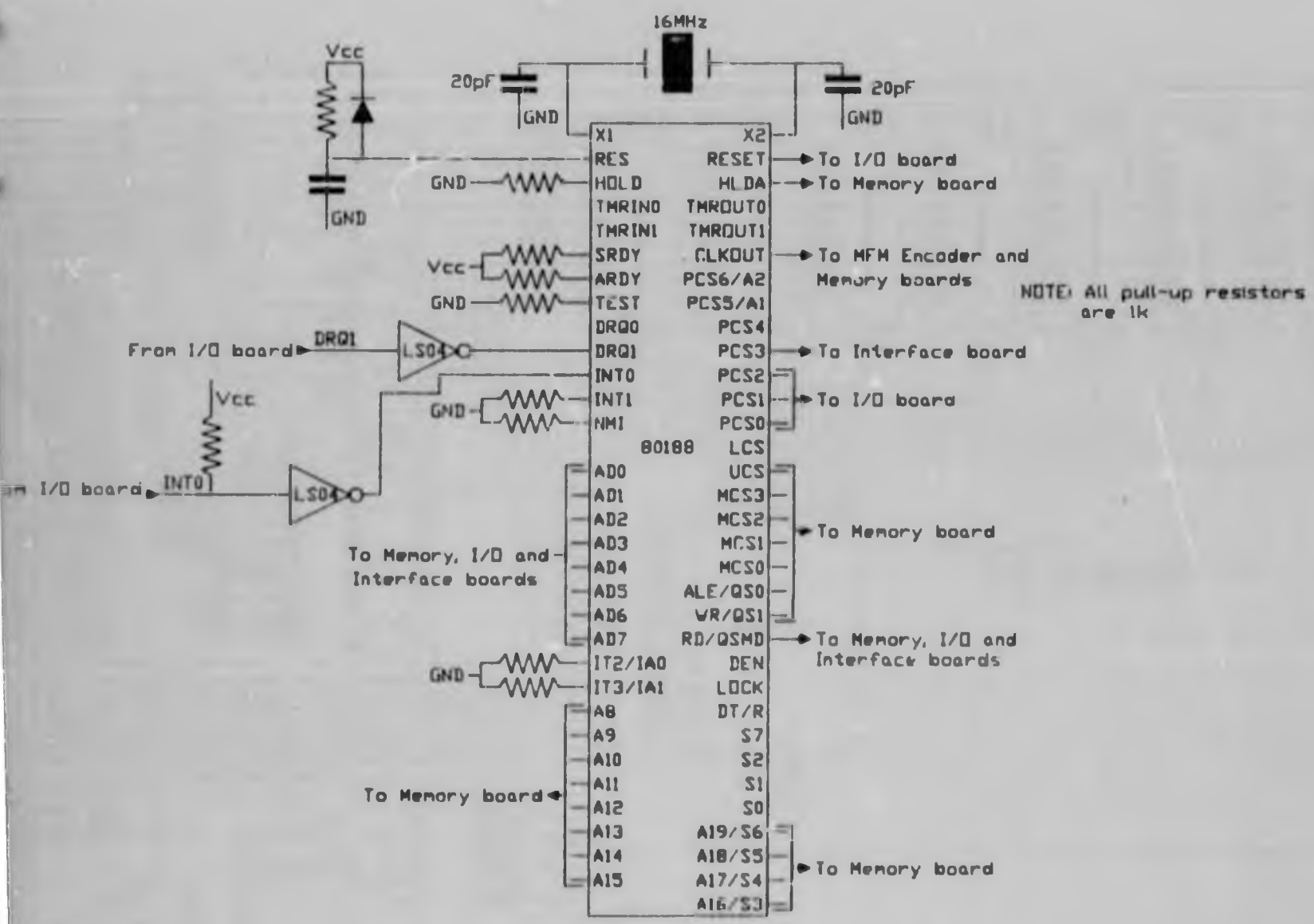


Figure H.2 The 80188 CPU Circuit Diagram

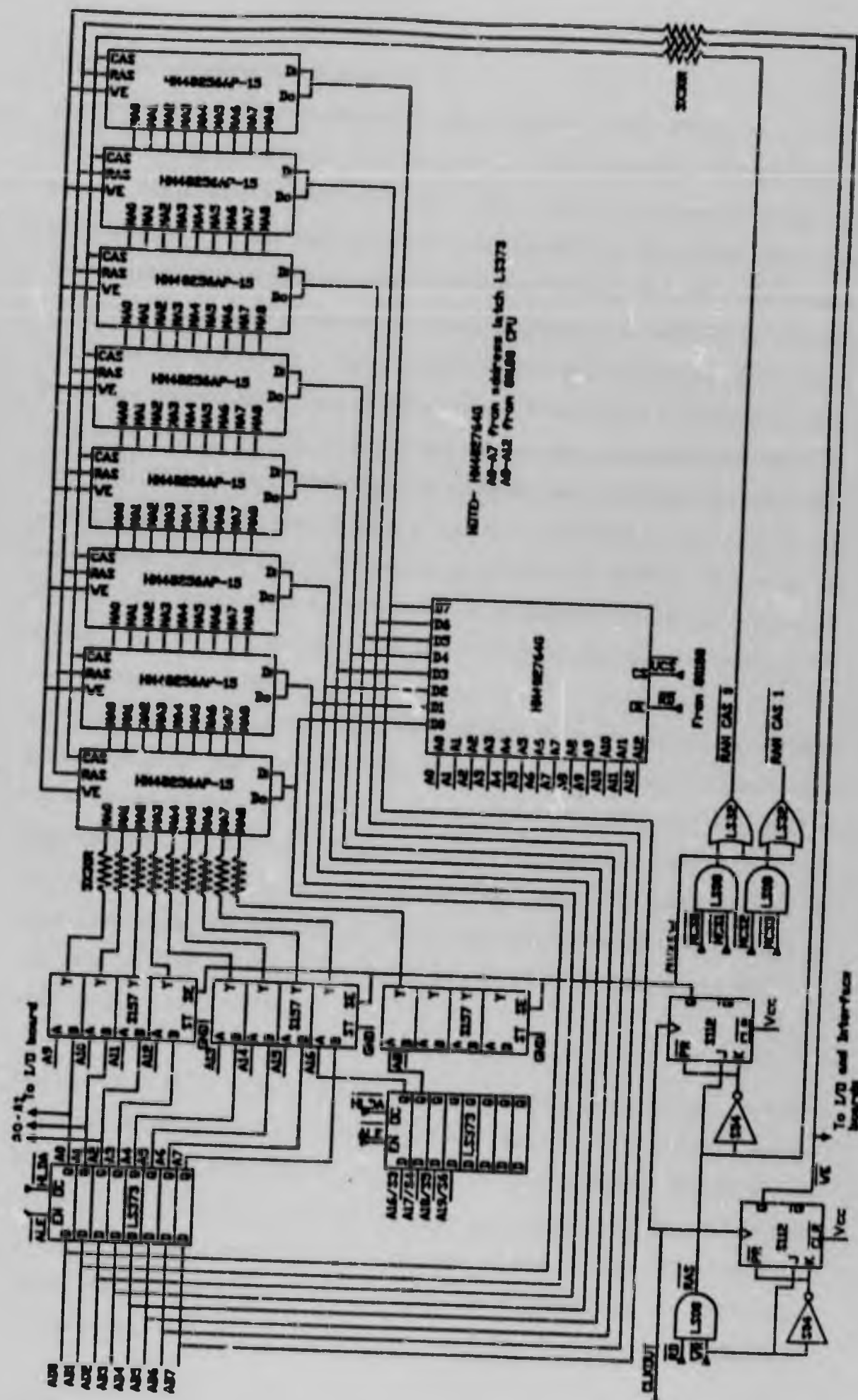


Figure H.3 The Memory Circuit Diagram

The Memory Board See Figure H3.

Note the sources of the address lines on the HN482764G EPROM.

The first requirement for both memory and I/O devices is for the lower address lines A0 - A7 and upper lines A16 - A19 to be de-multiplexed. On the 80188, the former group is multiplexed with the data lines while the latter group is multiplexed with some status lines. This is done simply using two LS373 latches which are enabled by the ALE (Address Latch Enable) line of the 80188. The outputs of the memory mapping unit in the 80188 are programmable thus allowing these lines to be activated at various positions in memory. The memory map of the microprocessor unit is shown in Figure H4. Address lines A0-A2 are provided in their de-multiplexed form for use on the I/O board. The EPROM is located at the top of memory since the reset vector is located at FFFF0H. The position of the UCS (Upper Chip Select) line is programmed to become active at FE000H. EPROMS with access times of up to 250ns can be used with a CPU clock frequency of 8MHz.

It is possible to use four banks of 256K RAM chips to give (1Mbyte-EPROM size)K bytes of RAM. If more than 512K is used, chip select lines have to be generated due to a peculiarity of the 80188. The four MSC (Mid - range Memory Chip Select) lines are used to enable the lower 512K i.e. 0 to 512K. The chip select lines for the area above this would have to be generated using UCS and A19 and A18. LCS (Lower Memory Chip Select) is not used in this design. MCS0 and MCS1 select 0 to 256K while MCS2 and MCS3 select 256 to 512K.

The 256K RAM chips have 9 multiplexed row/column address lines. Address lines A0 to A17 are multiplexed to provide the required 9 lines i.e. MA0 of the RAM is switched between A0 and A9 of the processor. The RD and WR lines of the processor are used to generate the RAS signal for the RAM. The RAS line also controls the operation of one of the S112 J-K flip-flops. When RAS is active and the flip-flop is clocked by the CLKOUT line of the 80188, the output changes thus switching the address multiplexor.

The multiplexor should therefore switch at the start of the T3 CPU cycle. See Figure H5. The actual CAS signals applied to the RAM chips is a combination of this multiplexor switching line and the Chip Select lines. Resistors are included on the inputs to the RAM chips in an attempt to keep the voltage overshoots and ringing that can occur to a minimum. The WE enable inputs of the RAM chips as well as that of the Z8530 require that the data to be written be stable on the data bus before this line is active. The WR line of the 80188 does not satisfy this condition and a WE signal is generated by allowing the output of the S112 J-K flip-flop controlled by WR only to become active on the rising edge of CLKOUT following the activation of the WR line. Again this should correspond to the start of the T3 CPU cycle.

The RAM chips require each row to be refreshed every 4ms. Considering that there are 256 rows that require refreshing, and assuming refreshing is done on a regular continuous basis, a refresh cycle is required every 15.625 μ s. The refreshing is done by programming one of the timers in the 80188 to request a memory DMA cycle. See Appendix O. Each of these cycles refreshes an entire row of the memory.

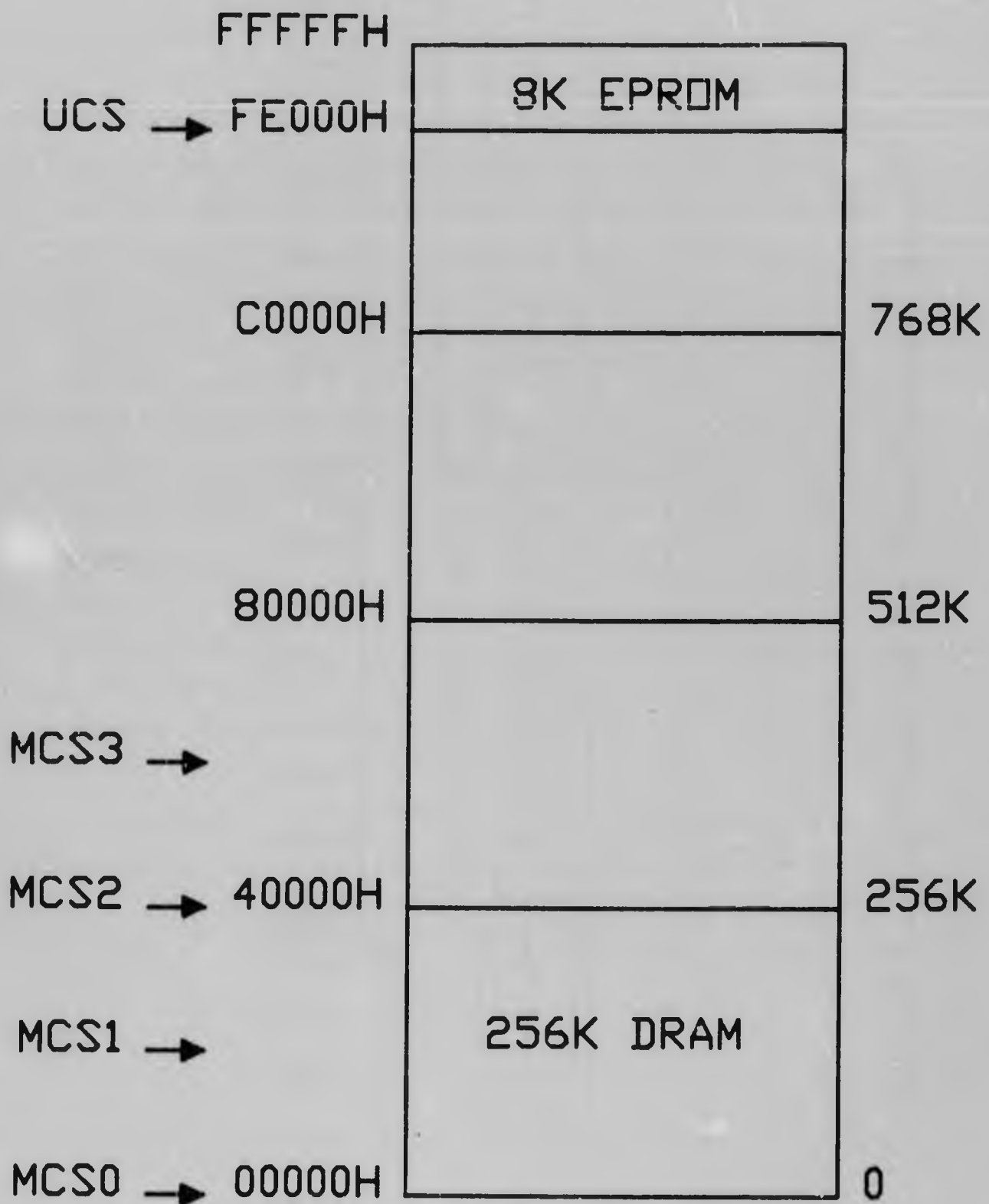


Figure H.4 The Memory Map Of The 80188

5nS/Sample

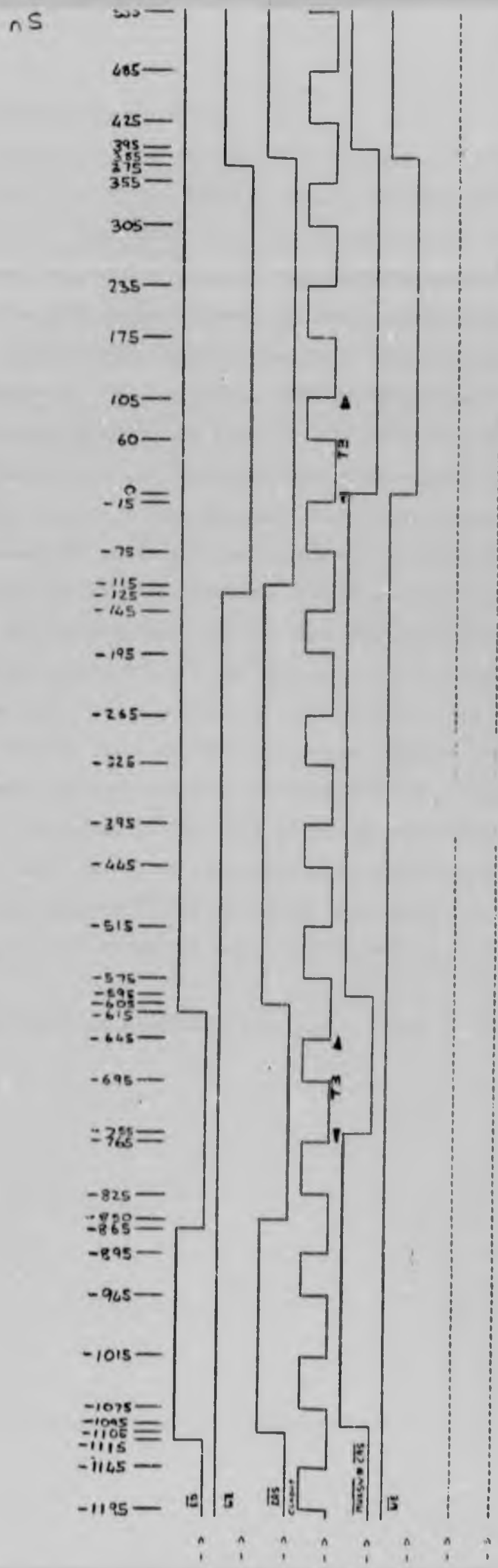


Figure H-5 Timing Diagram Of The RAS, CAS A1, WE Signals

nS

5nS/Scale

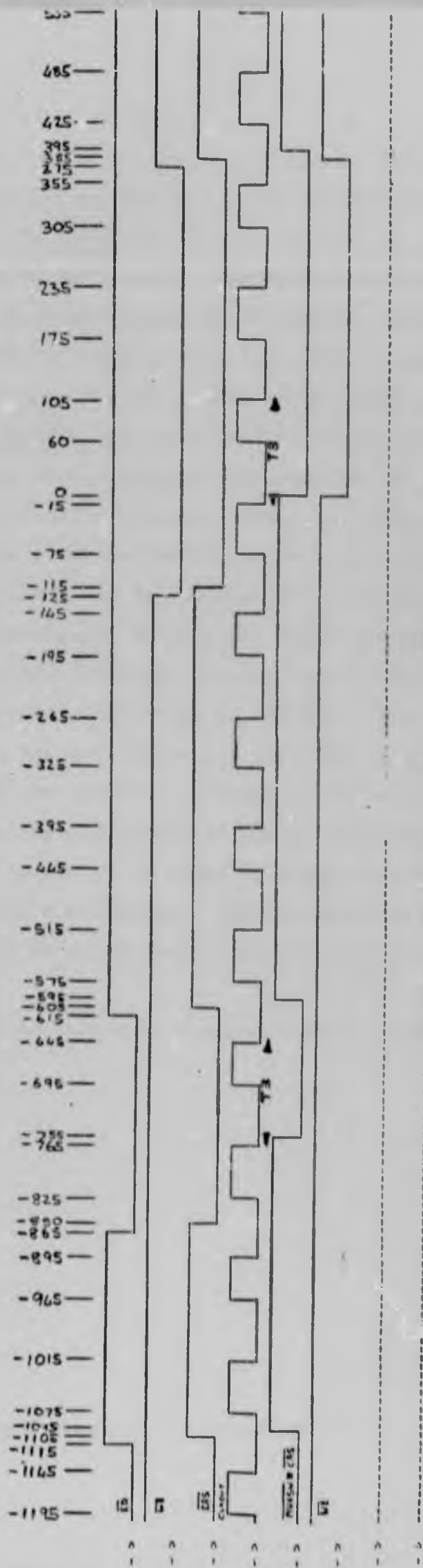


Figure H-5 Timing Diagram Of The RAS, CAS A1, and WE Signals.

The I/O Board See Figure H6.

The organisation of the I/O map is very simple. See Figure H7. The I/O devices used are the two Z8530's, a parallel input port and one parallel output port. The Z8530 used for the emulator is enabled by the PCS0 (Peripheral Chip Select) line 0. Address lines A0 and A1 from the memory board are used to select between the data/control registers and channels A and B respectively. Address line A2 is used to control the circuitry which allows for the reversal of data being written to or read from the Z8530. When A2 is high, the lower data transceiver (LS245) in the figure is enabled which allows unreversed data transmission. As can be seen from the figure when A2 is low the upper transceiver allows the data on line D0 to be applied to D7 etc. Both transceivers are only enabled when PCS0 is also low. The Z8530 used for the network is enabled using PCS1. No data reversal is required here and the data bus is therefore connected directly. Non-vectored interrupts are also allowed for. The DMA requirements are satisfied by inverting the RDY/REQA line of the Z8530 and applying this to the free DMA request input DRQ1 of the 80188. The clock requirements of the Z8530's are satisfied by a simple buffered ring oscillator operating at 6MHz. The RD and WE (described above) are gated with the RESET output of the 80188 to provide a hardware reset to the Z8530's. The Z8530's do not have a specific RESET input but can be reset by activating the RD and WR inputs simultaneously.

The output latch is enabled by PCS2 and WE being low simultaneously.

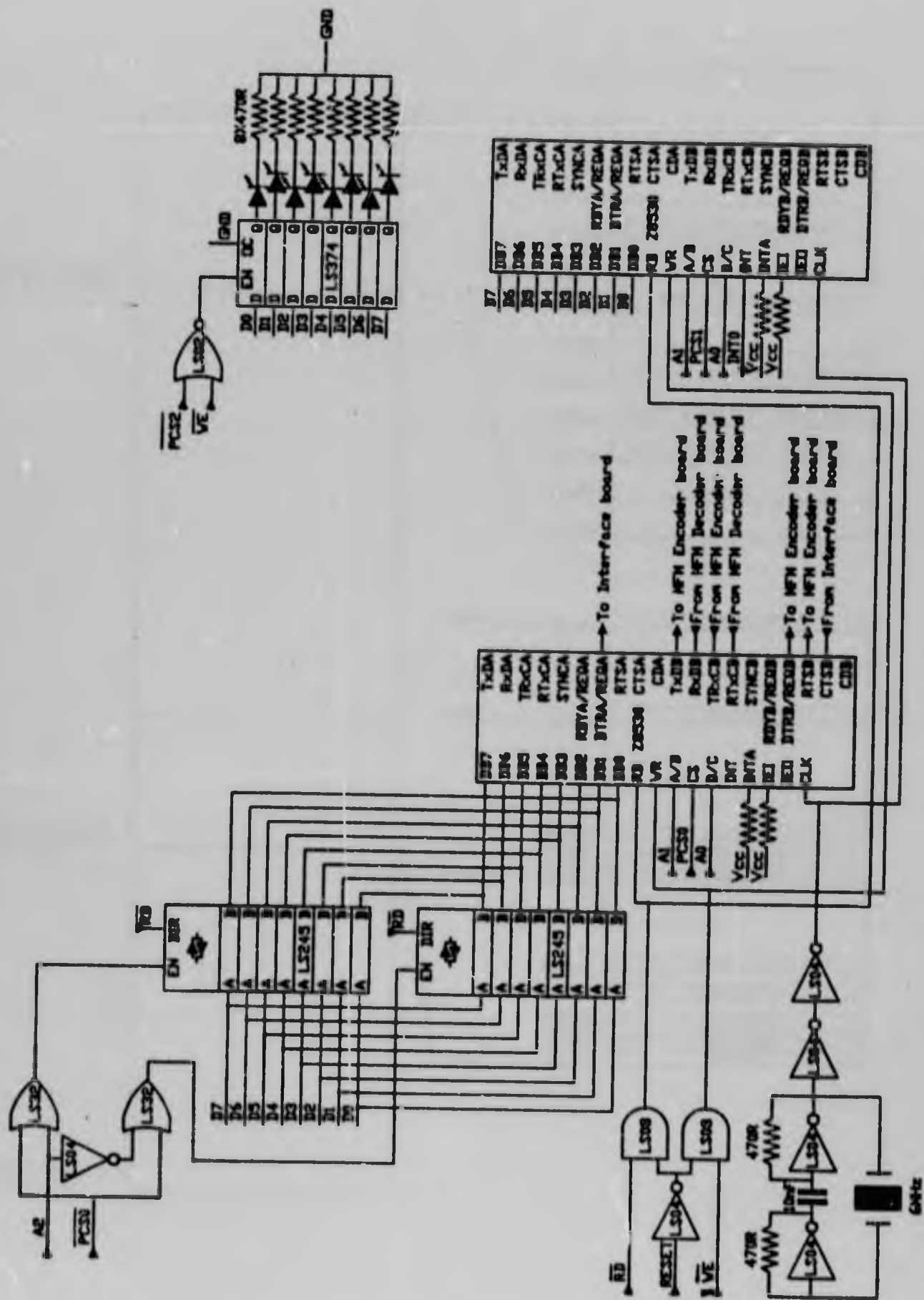


Figure H.6 The I/O Circuit Diagram

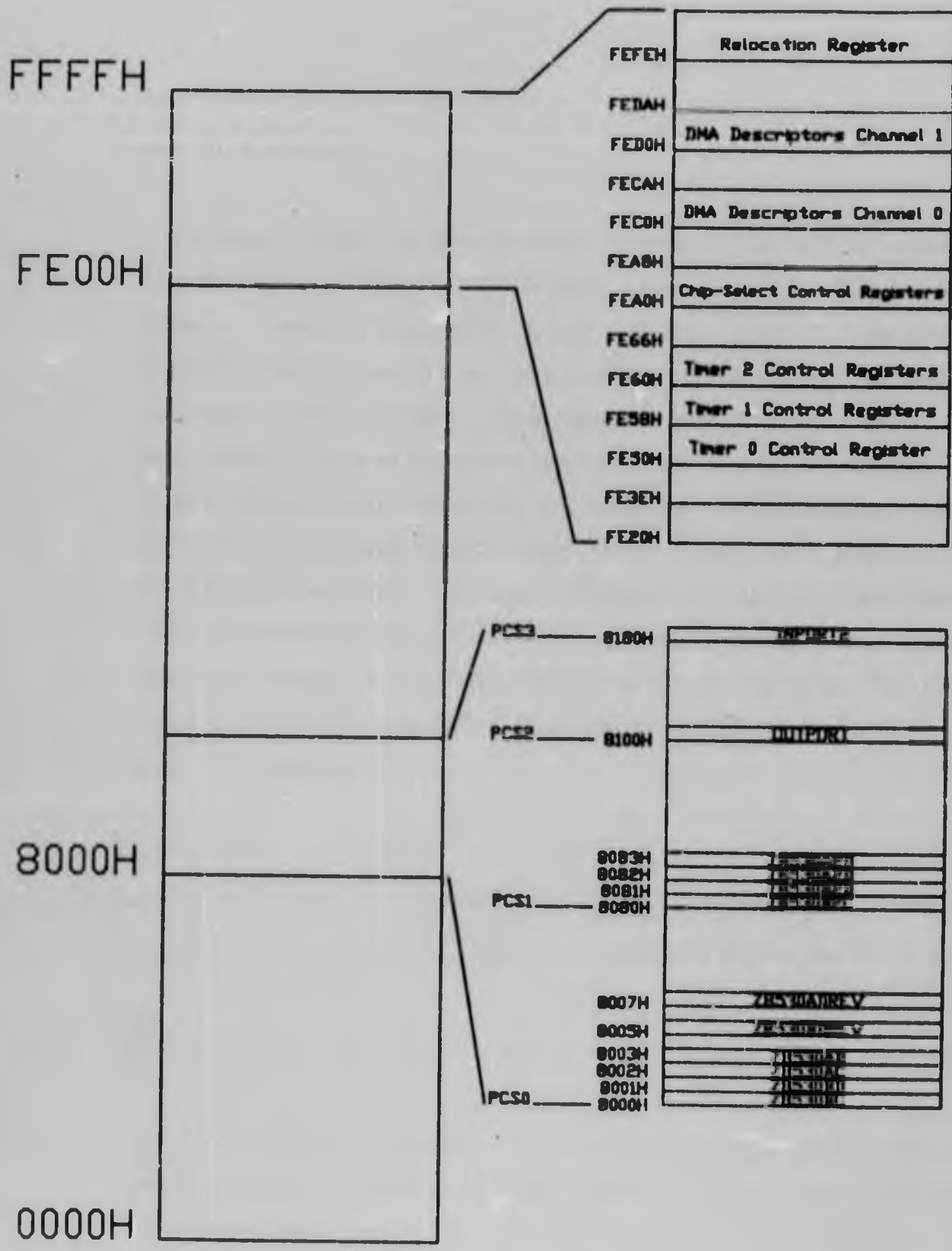


Figure H.7 The I/O Map

APPENDIX I THE TRACK CACHE

The implementation of the track cache and the cache replacement algorithm will be described.

The smallest 'block' of data that can be uniquely defined by the emulator is a particular track on a particular side of a disk. This block then forms the basis of the cache. Along with this each of these blocks is associated with a particular application program. As has already been described, each of these blocks requires some form of identification. Each block is defined by a track number, side number and an application number. Various other variables are required for the caching algorithm and will be discussed shortly. Each block of data is 4.5Kbytes. In the present implementation just under 192Kbytes of memory are available for the cache. This is therefore sufficient memory space for 42 blocks of data. The layout of the cache could take one of two forms. The first of which is similar to the layout on a disk viz. header, data, header, data etc. See Figure 11.

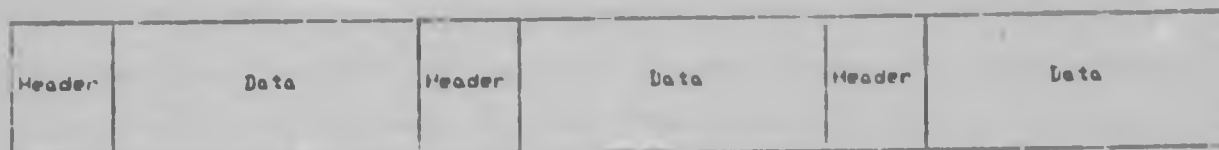


Figure 11 Cache Implementation With Each Data Segment Preceded By A Header

The header would contain the identification mentioned above. The second method consists of an area of memory dedicated to an 'index' and another area of memory dedicated to the data. The index includes a pointer to the data area corresponding to each entry. See Figure 12.

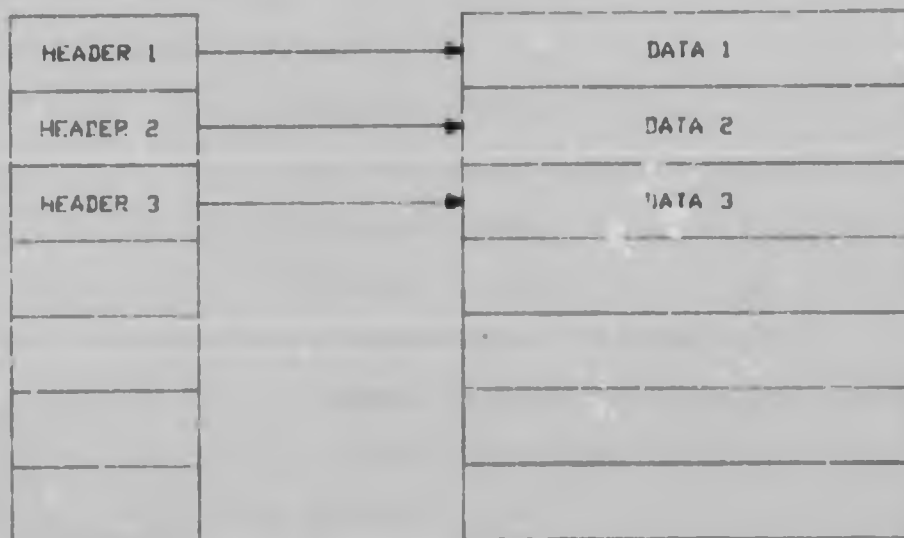


Figure 12 Separate Index And Data Areas For The Cache

Considering the way in which the 80188 manages its memory the second option was chosen. In the first situation when the cache is being searched for a particular entry, a memory area of more than 64K may have to be traversed to find the required header. The 80188 manages its memory in segments with a maximum size of 64K. The segment register would have to be updated then during the search which is undesirable. In the second option the segment register is assigned only once to scan the index since all the data lies well within 64K. An unusual way of implementing a pointer is used in the index as well. The number of bytes in each block of data is divisible by 16. Instead of implementing the pointer using two words i.e. one for the segment and one for the offset, the offset is always made zero and the segment register is made to point to the beginning of the data area for each track. The segment register alone is therefore included in the index.

The index can be described in high level language terms as an array of records as shown below.

```

Track_list_entry = RECORD
    Drive_no : byte;
    Application_no : word;
    Track_no : byte;
    Side_no : byte;
    Mask : byte; {For Caching}
    XBFREQ : word; {For Caching}
    XBAGE : word; {For Caching}
    Data_pointer : word;
END;
Index = ARRAY[1 .. no_of_sides] OF Track_list_entry;

```

This is shown diagrammatically in Figure 13.

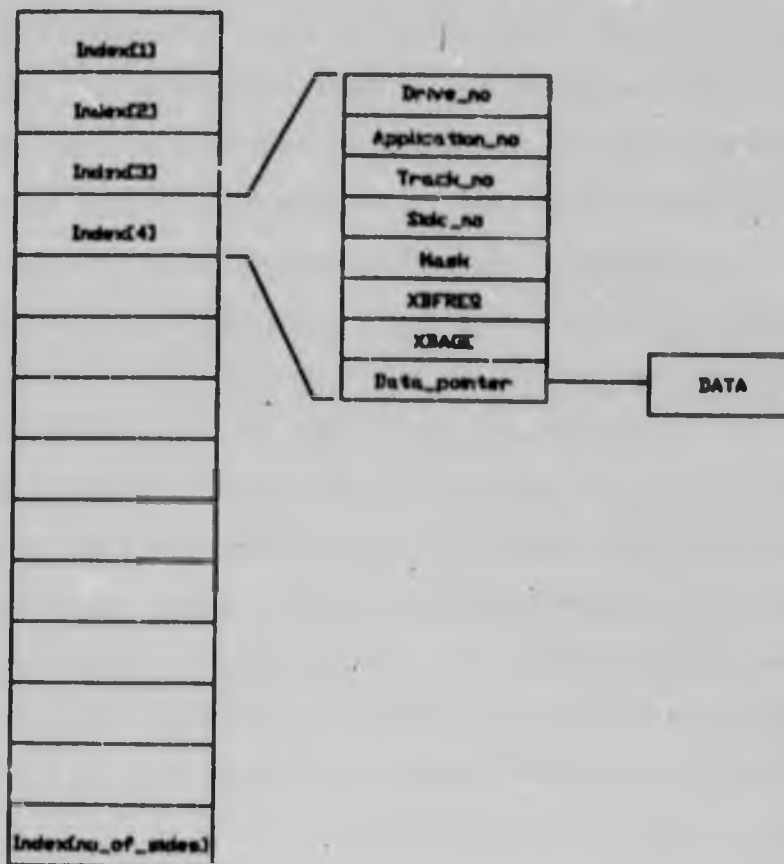


Figure 13 Diagrammatic Representation Of The Cache Data Structure

Managing the cache is a simple task until all the blocks in the cache are filled. The problem then arises as to which block should be overwritten. Decision algorithms in common use are random replacement, First In / First Out (FIFO) and Least Recently Used (LRU). In large caches the performance of these algorithms is much the same [5]. The key to the better technique proposed by McKeon [6] is to measure the frequency of access of a block and use this in combination with when last the block was used when making the replacement decision. The additional information in the RECORD above is to implement this algorithm. The Mask is to indicate whether a block is in use and whether a track has been updated i.e. written to, but has not yet been updated on the file server. XBFREQ is used for the frequency of access while XBAGE is for when last the block was used.

When a block is found in the cache (a 'hit'), the frequency of access parameter (XBFREQ) is increased. When a block is not in the cache (a 'miss') and no buffers are free, the cache is scanned to find a block with the lowest frequency of use i.e. lowest XBFREQ and this block is replaced. If multiple blocks with the same lowest frequency of use are found, then the block that has not been used for the largest amount of time is replaced. This decision is made using the XBAGE parameter. It is obvious that copies that stay in the cache have a better chance of being found and therefore the XBFREQ parameters will continue to increase. This parameter is 'decayed' over a period of time otherwise some blocks would remain permanently in the cache. The decay process is implemented by halving the XBFREQ parameter every sixteenth call to the cache. The reason for halving every sixteenth call is that these functions can be performed conveniently and quickly using masking and rotates. The XBFREQ parameter is increased by 128 for each 'hit'. Using these values, even in the worst case where the same block is accessed repeatedly, the XBFREQ parameter cannot overflow. The maximum that XBFREQ can attain is $16 \times 128 + \frac{1}{2} \times 16 \times 128 + \frac{1}{4} \times 16 \times 128 + \dots$ which can be recognised as a geometric

progression. In the limit then the maximum value that can be reached is $2 \times 16 \times 128$.

A high level description of the replacement algorithm, based on the data structure described above, is shown below.

```
CONST
    no_of_sides = 42;      {Maximum number of tracks in cache }
    in_use = 10000000B;   {Bit mask indicating block in use }

VAR
    devioage : word;      {Global variable}

BEGIN
    devioage := 0;        {Initialisa   of variable in main program}
PROCEDURE get_block (reqdrive, req   reqtrack : byte,
                    reqAppln_no      : word,
                    VAR data_point   : word);

VAR
    index_posn : byte;

BEGIN
    devioage := devioage + 1; {increment global age for each routine access

    IF devioage = 16 THEN
        FOR index_posn := 1 TO no_of_sides DO
            WITH index[i] DO
                IF ((Mask AND in_use) = 1) THEN
                    XBFREQ := XBFREQ DIV 2;
                    {If block in use then halve the frequency of access }
                Findblock(reqdrive, reqside, reqtrack, reqAppln_no, status, index_posn);
                IF status = successful THEN
                    WITH index[index_posn] DO
                        BEGIN
                            XBFREQ := XBFREQ + 128; {Record a hit}
                            XBAGE := devioage; {Update access age}
                            data_point := data_pointer {Set pointer to data area}
```

```

        status := successful;
    END
ELSE BEGIN (block not in cache)
    allocate_block (index_posn);
    WITH index[index_posn] DO
        BEGIN
            Drive := reqdrive;
            Side := reqside;
            Track := reqtrack;
            Application_no := reqAppln_no;
            Mask := in_use;
            XBAGE := devioage;
            XBFREQ := 0;
            data_point := data_pointer;
        END;
        get_data (data_pointer);
    END;

PROCEDURE Findblock (reqdrive, reqside, reqtrack : byte;
                    reqAppln_no : word;
                    VAR status index_posn : byte);

BEGIN
    index_posn := 0;
    REPEAT
        index_posn := index_posn + 1;
    UNTIL (index[index_posn].Drive = reqdrive AND
           index[index_posn].Side = reqside AND
           index[index_posn].Track = reqtrack AND
           index[index_posn].Application_no = reqAppln_no) OR
           (index_posn = no_of_sides);
    status := NOT (index_posn = no_of_sides);
END;

```

```

PROCEDURE allocate_block (VAR index_posn : byte);
  VAR
    age, freq, oldest : word;
    i, marker : byte;
  BEGIN
    index_posn := 0;
    REPEAT
      index_posn := index_posn + 1;
    UNTIL ((index[index_posn].Mask AND in_use) = 0) OR
      (index_posn = no_of_sides);
    status := NOT (index_posn = no_of_sides);
    IF NOT status THEN
      BEGIN
        freq := 32767;  {Find XBFREQ smaller than this}
        oldest := 0;
        FOR i:= 1 TO no_of_sides DO
          WITH index[i] DO
            IF ((Mask AND in_use) = 1) AND ((Mask AND write) = 0)
              THEN IF XBFREQ <= freq THEN
                BEGIN
                  freq := XBFREQ;
                  age := devioage - XBAGE;
                  IF (age > oldest) THEN
                    BEGIN
                      oldest := age;
                      marker := i;
                    END;
                END
              END
          END
        END
      END
    END;
  END;

```

The `get_block` procedure would be called each time data is required from the cache. The procedures are then called as required. The `get_data` procedure is not described here but this procedure's responsibility is to obtain the required data from the file-server and insert it at the area starting where `data_pointer` is pointing to.

APPENDIX J WHY 'STAR-BUS' NOT ETHERNET

The disk drive emulator board has a net data rate of 250Kbits per second between it and the P.C.. To provide each user with a response time comparable to that of a floppy disk, the network must be capable of providing data at a sufficient rate. To illustrate the needs of the network a loading situation that was described earlier to be possible in this type of environment will be considered viz. a number of users simultaneously requesting data that has to be fetched from the file-server. The emulation of a track of data takes approximately 200ms. If we take a number of emulators each successively requesting data from the file-server we can obtain a feel for the bit rate necessary for the network. A bit rate of 250Kbits per second would therefore sustain the needs of one emulator. If we take a system with 25 P.C.'s the required bit rate can be calculated.

$$\begin{aligned} \text{Data rate} &= 250\text{Kbits per second} \times 25 \\ &= 6,250 \text{ Mbits per second} \end{aligned}$$

An Ethernet type system with a bit rate of 10 Mbits per second would therefore satisfy the requirements from the bit rate point of view. This has some significant cost implications as will be discussed shortly.

Looking at a simpler twisted pair type network with transmission speeds of the order of 1 Mbit per second we find that the operation of a maximum of four emulators can be sustained. This therefore does not provide very extensive networking capabilities. Typical maximum transmission speeds that can be achieved with common communications controllers is 1.25Mbits per second. This increases the maximum number of emulators to five. The possibility of constructing a star type network with the file-server at the hub was considered. Each ray of the star would consist of a bus structure operating at 1.25Mbit per second. The number of these rays could

be increased until sufficient nodes could be catered for. In this way an equivalent effective throughput of the Ethernet system could be achieved.

As always these two systems have compromises. In the first instance the cost and complexity per emulator would be increased considerably since hardware capable of providing these high bit rates has to be provided. The file-server would only require one of these Ethernet type controllers and would therefore be less complex than that for the star system. For the second system each emulator need only be capable of transferring data at 1.25Mbits per second that can be handled by a communications controller. The emulator's network complexity is therefore limited. The file-server is however far more complex than in the first case. A communications controller is required for each of the rays of the star. The bus access is less contentious however having fewer nodes on each bus since there are a number of busses running in parallel.

The added complexity of the file-server is worthwhile when the costs for the Ethernet type system are considered. Although the system may be complex, the individual components on the file-server board are relatively cheap. The overall system is therefore substantially cheaper. The Ethernet type interface would make the goal of the P.C. interface price being comparable to that of a disk drive, impossible to attain. Typical prices for Ethernet controllers alone are of the order of R200 while the communications controllers in question are of the order of R20. The wiring and installation costs are also substantially cheaper than those associated with a coaxial cable based system.

APPENDIX K THE CSMA CA ACCESS PROTOCOL

The principle of operation of the CSMA CA (Carrier Sense Multiple Access with Collision Avoidance) access protocol will be described.

The purpose of this access protocol is to control the access to a shared bus system in an orderly fashion so as to reduce the probability of collision.

To describe the implementation of a CSMA CA system, the implementation used in the Apple-talk system will be described. In this implementation the bit rate is 250Kbits per second using a RS 422 twisted pair bus.

The dialogues (i.e. transactions on the bus) must be separated by a minimum Inter-Dialogue Gap (IDG) of 400µsec; the different frames of a single dialogue must follow one another with a maximum Inter-Frame Gap (IFG) of 200µsec. Consider the transmission of a data frame as shown in Figure K1. The transmitting node uses the physical layer's ability to sense if the line is in use. If the line is busy the node waits until it becomes idle (the node is said to *defer*). Upon sensing an idle line, the transmitter waits for a time equal to the minimum IDG (400µsec) plus a randomly generated amount. During this "wait", the transmitter continues to monitor the line. If the line remains idle throughout the wait period, then a RTS frame is sent to the intended receiver of the data frame. The receiver must, within the maximum IFG (200µsec), return a CTS frame to the transmitting node. Upon receiving this frame, the transmitter must send out the data frame within the maximum IFG.

The purpose of this collision avoidance algorithm is twofold: (1) to restrict the periods in which collisions are highly likely (this is during the RTS CTS exchange), and (2) to spread out in time several transmitters waiting for the line to become idle. The RTS CTS exchange, if successfully completed signifies that a collision did not occur, and that all intending

transmitters have heard of the coming data frame transmission and are deferring/waiting.

If a collision does occur during the RTS CTS exchange, a CTS will not be received, and the sending node will then *backoff* and retry. The sending node is said to presume a collision.

The range of the random wait is adjusted if such a collision is detected. In fact, this adjustment or back off is done using a linear back off algorithm that dynamically modifies this range in response to recent traffic history. The idea is that if collisions have been presumed for recently sent packets, this signifies heavier loading and higher contention for the bus. Then the random wait should be generated over a larger range, thus spreading out (in time) the different contenders for the line.

Two factors are used for adjusting the range: (a) the number of times the node had to defer, and (b) the number of times it had to backoff. This history is maintained in two 8-bit *history bytes*, one each for the deferrals and for backoffs. At each attempt to send a packet these bytes are shifted left one bit. The lowest bit of each byte is then set if the node had to defer or back off, respectively, on that attempt, else this bit is cleared. In effect the history bytes remember the deferral and back off history for the last eight attempts.

The history byte is used to adjust a *global backoff mask*. The mask takes values in the range 00H to 0FH. When the first attempt is made to send a particular frame, and the node must defer, the mask is OR'ed with 01H, thus setting its lowest bit.

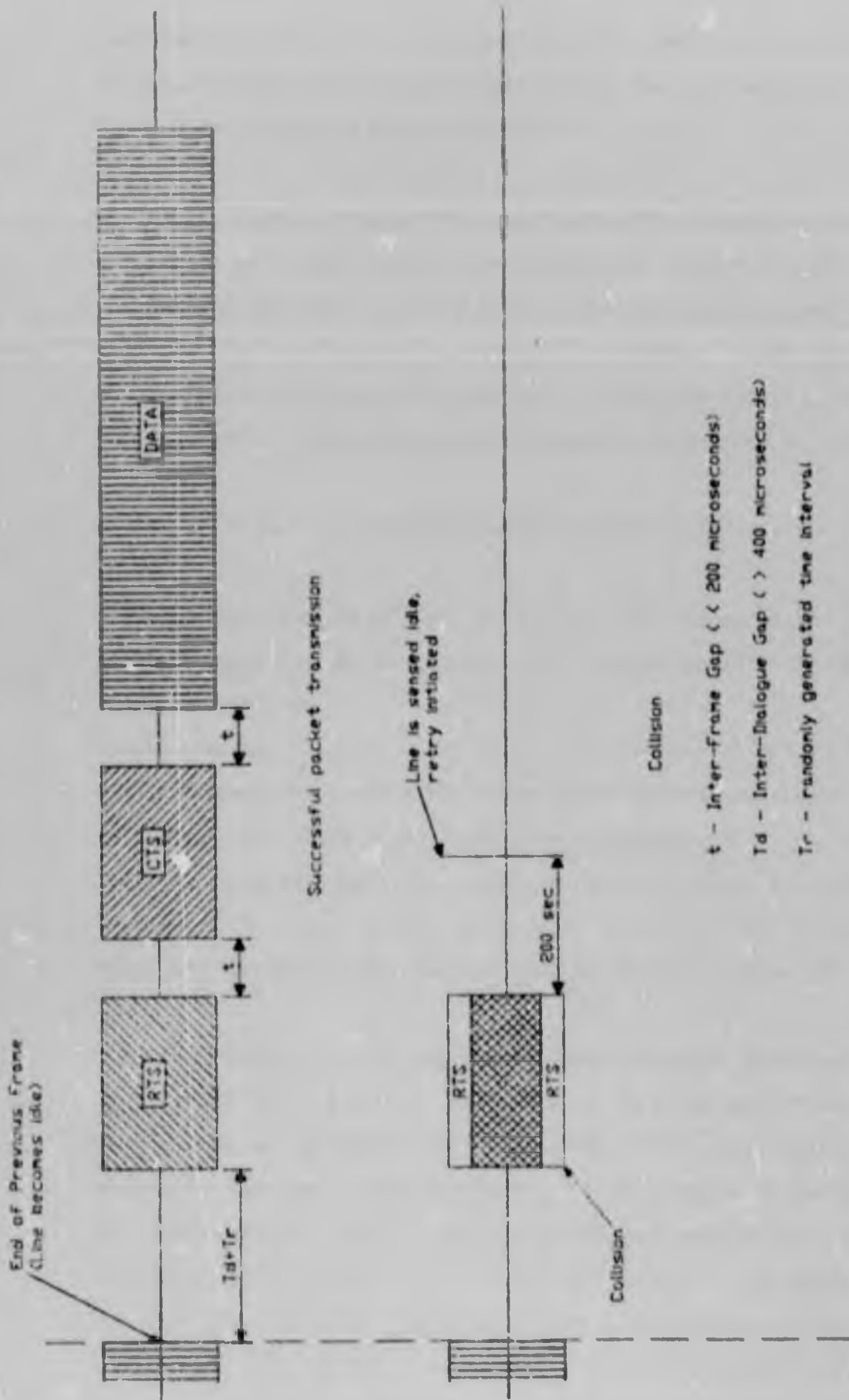


Figure K.1 Timing of A CSMA CA Network Dialogue

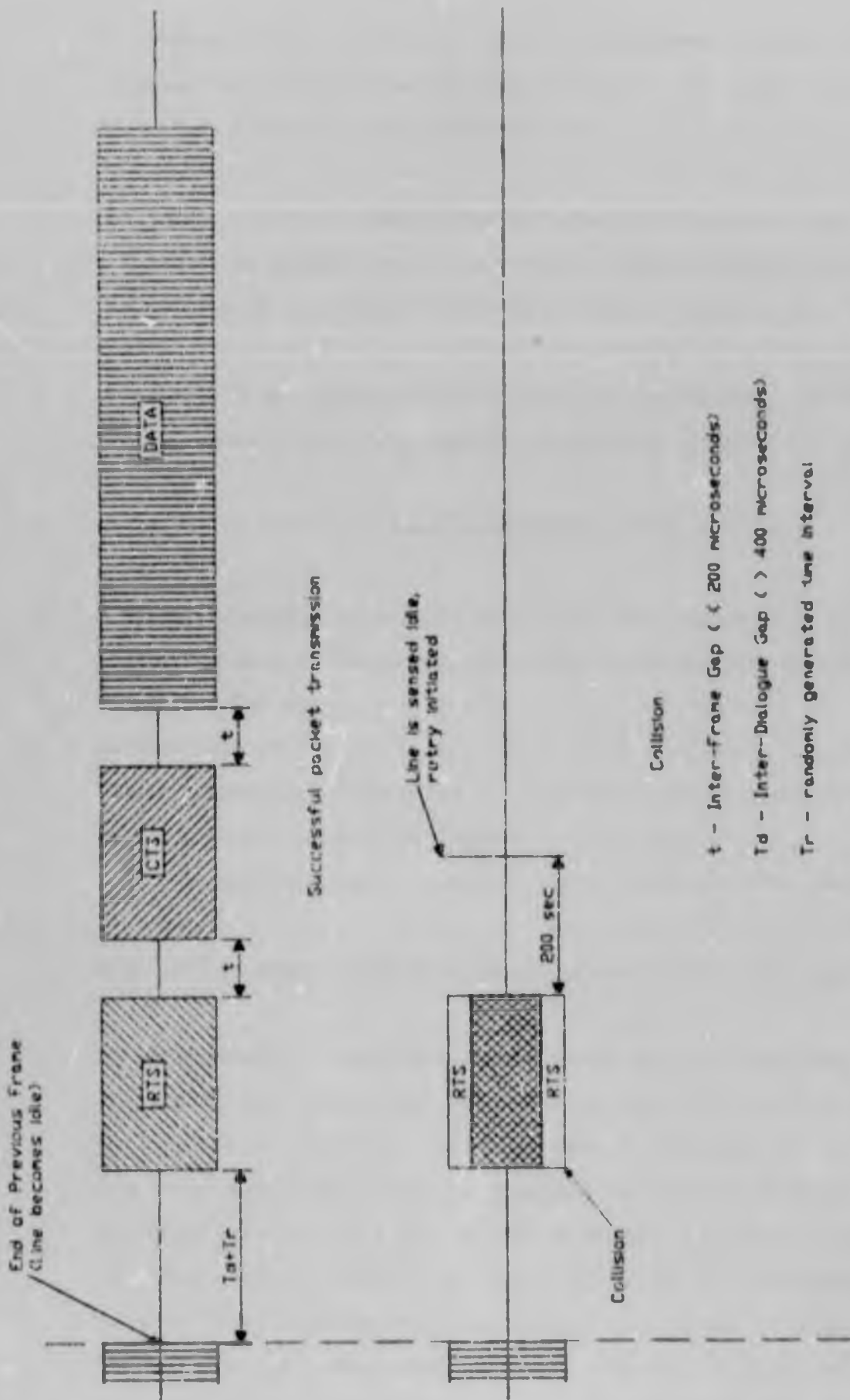


Figure K.1 Timing Of A CSMA CA Network Dialogue

The number of bits set in each history byte provides a count of the number of times the node had to defer (back off) in the last eight attempts. This is used to adjust the mask as follows:-

- o If the number of times the node backed off during the last eight attempts is greater than 2, the mask is extended by one bit (up to the maximum of 4 bits) and the back off history byte is then cleared.
- o Else, if the number of times the node had to defer is less than 2, the mask is reduced by one bit (down to the minimum of 1 bit).
- o Else, if neither of these apply, the mask is left as is.

A typical dialogue takes place as follows (See Figure K1)

L1)The transmitter senses the bus until the bus is idle for the minimum IDG time.

L2)The transmitter waits an additional time determined by a random number.

L3)The transmitter sends a RTS frame to the intended destination node.

L4)The receiver sends a CTS frame to the transmitter.

L5)The transmitter, upon successful reception of the CTS, sends a Data frame.

NOTE: All response frames must be sent within the maximum IFG time.

For each attempt in L2 a new random number must be generated. If after 32 attempts the transmitter is unable to send the data frame, an error is reported to the higher level software. The access method may seem clumsy in that three frames are required to transfer on packet of data. The extra transactions required must be weighed against the extra circuitry that would be required to detect collisions on the hardware level. We must also consider the frame format as is shown in Figure K2. The figure shows the general frame format. It should be noted that the frame is of the form used in SDLC links and thus the flags are 7EH bytes and

the frame check sequence is the same as that used in SDLC. Carrier is said to be present when the receiver has detected the start of a valid frame i.e. a flag has been found. The RTS and CTS consist only of the destination address, source address and type field. This type field is used to differentiate between an RTS, CTS and data frames. The data frame thus includes the data field. The frames used therefore to detect collisions are substantially smaller.

The CSMA CA access protocol provides a convenient means for bus control that does not require complex hardware or software.

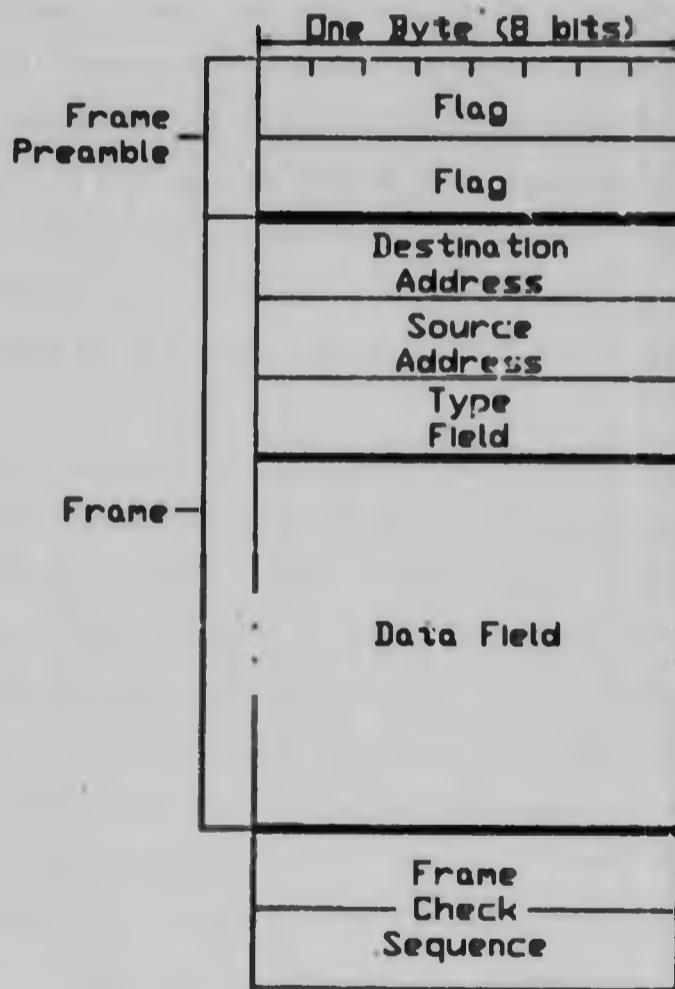


Figure K.2 CSMA CA Frame Format

APPENDIX L MESSAGE PASSING ON TRACK 40

The reason for using track 40 for passing messages between the P.C. and the floppy disk drive emulator will be explained as well as the method used.

The file server needs to know when the logical drives are changed on the P.C. and when application programs are selected in the menu program. The amount of information that has to be transferred between the P.C. and the file-server is small and in general only a few bytes are required. To keep the data transactions between the P.C. and the emulator the same at all times, the method of transferring information in the form of a sector was chosen. In the case of MS-DOS this restricts the maximum message size to 512 bytes which is more than sufficient. Another factor in favour of this method is the availability of BIOS routines in the ROM of the P.C. that allow single sectors to be written to the floppy disk drives.

Before discussing the details of the BIOS routine, let us examine the reason for using track 40. There are three main reasons for this choice. The first is the structure of floppy disks, the second is a hardware restriction and the third is time. The floppy disks being emulated have forty tracks numbered from 0 to 39. None of these tracks can be used for message passing since they carry the data. The track number must therefore be greater than 39. The second constraint is directly related to the floppy disk controller used in the P.C. (Intel 8272A, NEC μ PD765). This controller is designed for use in disk systems where the disks have less than 77 tracks. When a RECALIBRATE command is given to the floppy disk controller, the head is stepped outwards towards track 0 until the signal from the floppy disk drive unit indicates that the head is over track 0 (TRACK0). If after 77 step pulses (i.e. 77 tracks) have been issued by the controller and a TRACK0 signal has not been received, the disk con-

APPENDIX L MESSAGE PASSING ON TRACK 40

The reason for using track 40 for passing messages between the P.C. and the floppy disk drive emulator will be explained as well as the method used.

The file server needs to know when the logical drives are changed on the P.C. and when application programs are selected in the menu program. The amount of information that has to be transferred between the P.C. and the file-server is small and in general only a few bytes are required. To keep the data transactions between the P.C. and the emulator the same at all times, the method of transferring information in the form of a sector was chosen. In the case of MS-DOS this restricts the maximum message size to 512 bytes which is more than sufficient. Another factor in favour of this method is the availability of BIOS routines in the ROM of the P.C. that allow single sectors to be written to the floppy disk drives.

Before discussing the details of the BIOS routine, let us examine the reason for using track 40. There are three main reasons for this choice. The first is the structure of floppy disks, the second is a hardware restriction and the third is time. The floppy disks being emulated have forty tracks numbered from 0 to 39. None of these tracks can be used for message passing since they carry the data. The track number must therefore be greater than 39. The second constraint is directly related to the floppy disk controller used in the P.C. (Intel 8272A, NEC μ PD765). This controller is designed for use in disk systems where the disks have less than 77 tracks. When a RECALIBRATE command is given to the floppy disk controller, the head is stepped outwards towards track 0 until the signal from the floppy disk drive unit indicates that the head is over track 0 (TRACK0). If after 77 step pulses (i.e. 77 tracks) have been issued by the controller and a TRACK0 signal has not been received, the disk con-

troller indicates a fault with the equipment [3]. This is not acceptable and therefore we cannot step further than 77 tracks from track 0. The permissible range for the message passing track is then between tracks 39 and 77.

Stepping between tracks is a slow process (of the order of milliseconds) and should be avoided as much as possible. It is an advantage therefore to have the track used for messages as close to the data tracks as possible. Hence the choice of track 40.

The BIOS routine for accessing the floppy disks is an interrupt routine viz. INT 13H. Various parameters are passed to this routine via the registers of the 8088/8086 CPU of the P.C..

The parameters are as follows [7]

Inputs: AH = 0 Reset disk system

= 1 Read status of disk system into AL

= 2 Read desired sectors into memory

= 3 Write desired sectors from memory

= 4 Verify desired sectors

= 5 Format desired track

DL - Drive Number (0-3 allowed, value checked)

DH - Head Number (0-1 allowed, not value checked)

CH - Track Number (0-39, not value checked)

CL - Sector Number (1-9, not value checked)

AL - Number of Sectors (Max = 9, not value checked)

ES : BX - Address of Buffer

Outputs: AH = Status of Operation

No Carry - Successful operation (AH = 0 on return)

Carry Set - Operation failed (AH has error reason)

The message to be written is assembled in memory and ES is set to the segment containing the message and BX to the offset of the message. The

required function viz. write desired sectors from memory is chosen by setting AH to 3. An assembler program segment will be given for the sake of completeness.

Note: It is assumed that the message is in a byte array called BUFFER defined as

```
BUFFER DB 512 DUP(?)
```

The program segment follows:

```
MOV AX,SEG BUFFER ;LOAD SEGMENT IN WHICH BUFFER LIES
MOV ES,AX ;INTO ES
MOV BX,OFFSET BUFFER ;LOAD BUFFER OFFSET IN BX
MOV AH,3 ;SELECT WRITE SECTORS FUNCTION
MOV DL,1 ;SELECT PHYSICAL DRIVE B
MOV DH,0 ;SELECT SIDE 0
MOV CH,40 ;SELECT TRACK 40
MOV CL,1 ;SELECT SECTOR 1
MOV AL,1 ;WRITE ONLY ONE SECTOR
INT 13H ;CALL BIOS ROUTINE
:
:
```

The floppy disk drive emulator need only emulate this one sector on track 40 since this is the only sector that the floppy disk controller will look for.

This method is an effective way of transferring the required data between the P.C. and the emulator and file server.

APPENDIX M THE MS-DOS DEVICE DRIVER

The device driver creates 24 logical drives on the P.C. as required by the network as well as being responsible for passing messages on track 40 to the emulator when logical drives are changed. This file is called up during the booting up process of the P.C. and forms part of the CONFIG.SYS file. Various criteria for the structure of device drivers are specified and since these are standard they will not be discussed in detail here. Only the important features will be explained since the code (later in the Appendix) is well commented. Numbers in brackets refer to offsets in the program. This is the left-hand most column of numbers in the program listing.

In the device header, the constant DRVMAX (000A) corresponding to the Name/unit field in the standard device header [8] is set to 24 by the INITIALISATION routine (0650) of the device driver. If fewer drives are required the 24 can be reduced. MS-DOS assigns the letters of the alphabet, starting at A, to block devices such as the disks. Hence the standard two floppy disk system has drives A and B. Subsequent drives are assigned the letters up to Z. If more than 24 drives are requested in this device driver, the drive identifiers become the ASCII characters following capital Z in the ASCII character table. This makes drive identification somewhat difficult. The remainder of the parameters in the device header are the pointer to the next device header, attribute of the device, the pointer to the device strategy and the pointer to the device's interrupt routine in that order.

DRVTBL (000B) contains a number of pointers to routine to be called depending on the function required when the device driver is accessed. As can be seen from the listing a number of these point to the exit routine directly since these functions are not supported by the device driver.

The EXIT (0077) routine by which, as the name implies, the device driver terminates, is responsible for returning status information regarding the operations performed, to the calling program. The most significant bit of AH, when set, indicates that an error has occurred. The least significant bit of AH indicates that the operation is complete. If an error has occurred, the error code is returned in AL. The number of successful I/O operations that did take place is updated before the machine state is restored and control is handed back to the calling program.

The MEDIASCHK (008B) routine is called by DOS before an operation is performed by DOS on a logical device. This is necessary so that DOS knows whether this data relates to the actual media in the logical device or not. If we consider a floppy disk system where the medium is removable the information in the buffer may be from a previous disk because the media can be changed between accesses to the disk. In the case of a hard disk which is not removable we know the buffer will have valid information. This media check can return one of three values to DOS viz. The media has been changed, the media hasn't been changed or I don't know if the media has changed. It is obviously an advantage to be able to say that the media has not been changed. In the case of the emulator unfortunately we cannot say that the medium has not changed because a floppy disk may be mapped out by the file-server for example between accesses. The device driver therefore always responds with I don't know if the medium has been changed.

The BUILDBP routine (00AB) generates the required BIOS parameter block according to the given media descriptor byte. The routine usually generates the parameter block for a double sided disk with nine sectors per track. It is possible to have 8 sectors per track and a single sided disk if so desired.

This routine is followed by a number of variables, the first of which records the current drive. It may be recalled that both the device driver

in the P.C. as well as the emulator are required to remember which was the most recent logical device that was accessed. This variable performs this function and is initialised to a value that would not normally exist. This is done so that as soon as the first access is made to the one of the logical drives, a message is sent to the emulator. The remaining variables store temporary values during the actual data transfer.

When DOS calls a block device driver i.e. a device driver for a disk or similar system where the data is stored in blocks, the request is in terms of a logical start sector and a number of sectors after that, that have to be read or written. DOS also gives a Transfer Address where this data is to be transferred to or from. This method of requesting data is useful in the general sense that it is not device specific. It is upto the individual device driver to interpret exactly where on the physical device being used, the data must be transferred. In this case therefore we require a start track, start side and starting sector on the physical device. Before the calculations are done to find these parameters the current logical device must be conveyed to the disk drive emulator. In the SETUP (0135) routine after the jump is made to INRANGE (0147), the current drive is compared to the requested drive number. If they are the same all is well, if not a message is passed to the emulator via track 40 (See Appendix L). The message itself is inserted at this stage into the the buffer called TEMP_BUFFER (0450). The maximum length of the message is restricted to 512 bytes. The contents of the message itself are not important as long as they correspond to some message format that the emulator will understand i.e. the emulator's software and the device driver can be modified to send and receive message formats as may be required.

The message passing along with making the physical drive B in all cases takes care of the mapping of all the logical devices into one physical device. This is followed by the calculation of the physical location of

the data required from the logical sector specified by DOS. The number of sectors to be transferred is stored for later use. To calculate the starting sector the logical sector is divided by the number of sectors per track. The remainder of this calculation gives the starting sector between 0 and one less than the number of sectors per track. This value is incremented and stored. The quotient of this calculation is then divided by the number of heads on the device, in this case two, to determine on which side of the disk the data transfer begins. The remainder of this division thus gives the head required, while the quotient gives the starting track. We can now commence the data transfer.

The actual data transfer software has to be inside a loop, the reason for this being that the maximum number of sectors that can be transferred by each BIOS call is in this case nine i.e. one whole track. The number of sectors requested by DOS may exceed this value or we may have the situation where only two sectors may be required but the first of these is the last sector on a track. Two BIOS calls have to be made in this case. The loop counter therefore is the number of sectors that have to be transferred. The loop performs the following:- A retry counter is reset for each I/O operation. Four retries are performed for each operation before an error status is returned to DOS. The `RESET (03C0)` routine checks for when the end of a track has been reached. If the track end has been reached and the operation has just been performed on side zero then the side is changed to 1 and the physical sector set to 1 i.e. the start of the track on the other side of the disk. If the end of the track on side one is reached, the track number is incremented, the physical sector set to one and the side set back to zero. The first time this routine is called, none of these situations will arise since the starting parameters will just have been calculated. The parameters for the BIOS routine (See Appendix L) are loaded and the routine called. This BIOS routine returns with the carry flag set if an error has occurred. AL returns the number of sectors that have been transferred. Taking initially the case when

the routine returns without an error, the transfer address has to be updated by an amount equal to the number of sectors transferred times the sector size. The segment register (ES) is constantly updated to prevent the offset register overflowing. The transfer address is not updated by the BIOS routine since the data transfer is performed using DMA. The correct value of the transfer address must be obtained for subsequent data transfers. The number of sectors still to be transferred is updated and then the loop repeated until all the required data has been transferred.

When an error condition is returned by the BIOS suitable error handling has to be performed (BIOS_ERROR). The exact cause of this condition is returned in the AH register. If a DMA error is returned (AH=9) it is handled differently to the others since it is the only error that may not disappear if a retry is attempted. Since the cause and handling of this error are somewhat different to that for the others it will be discussed later. The retry counter is decremented and the operation re-attempted until the operation is either successful or the number of retries has been exceeded. In the latter case the BIOS error number is transformed to DOS via ERROR_EXIT (0292)

The DMA error results from a hardware design aspect of the P.C.. The DMA controller used is only capable of addressing 64K of memory directly. The P.C.'s memory size is 1 Mbyte and for this reason an extra register has been added to accommodate the extra memory i.e. the DMA controller controls address lines A0 - A15 while the register controls lines A16 - A19. The DMA error occurs in the following situation, DOS may allocate a data transfer area that causes the address register of the DMA controller to overflow. What we would like to happen is that the extra register be incremented automatically. Before commencing the data transfer the BIOS routine checks for this situation and if it will occur the error is returned. The following procedure is then adopted: the number of whole sectors that can be transferred before the overflow occurs is calculated.

A request to transfer these sectors is then given. The sector in which the overflow occurs is then transferred to a temporary buffer (TEMP_BUFFER) i.e. a request to transfer one sector is given with the transfer address being that of the temporary buffer. When this transfer has taken place, the temporary buffer is transferred byte by byte to the original data transfer area. The remaining sectors are then transferred in the normal manner. A similar approach is used when the problem occurs with a disk write. In this case however the data is transferred into the temporary buffer before the BIOS routine is called.

INITAB contains a pointer to a BIOS parameter block for each of the logical devices. The DRV\$INIT routine is called to initialise the device driver. The 24 that is loaded into AL in the first line of this routine is what must be changed if fewer device drivers are wanted.

PAGE .132 :SET PAGE WIDTH TO 132
 TITLE MULTIDISK Multiple logical floppy disk device driver

```

:THIS CREATES 24 LOGICAL FLOPPY DISK DRIVES
CODE SEGMENT
0000
0000

0000 0000H
ASSUME CS : CODE,DS : NOTHING,ES : NOTHING,SS : NOTHING

:DEVICE HEADER
LABEL WORD
DW -1,-1
DW 0000H

DRVDEV
STRATEGY
DRVSIN 24

DRVMAX
LABEL
DW DRVSINIT
DW MEDIASCHK
DW GETSBPB
DW CMDERR
DW DRVSREAD
DW EXIT
DW EXIT
DW EXIT
DW DRVSWRIT
DW DRVSWRIT
DW EXIT
DW EXIT
DW EXIT

:COMMAND CODES
:INIT
:MEDIA CHECK
:BUILD BPB
:LOGICAL INPUT
:INPUT (READ)
:NON-DESTRUCTIVE INPUT
:INPUT STATUS
:INPUT FLUSH
:OUTPUT WRITE (WITH VERIFY)
:OUTPUT STATUS (CHAR DEVICES)
:OUTPUT FLUSH (CHAR DEVICES)
:LOGICAL

PTRSAV
:STRATEGY 0

STRATP
PROC
MOV WORD PTR (PTRSAV),BX
MOV WORD PTR (PTRSAV+2),ES
RET
ENDP

:OFFSETS IN REQUEST HEADER
= 0
= 1
= 2
= 3
= 13
= 14
= 18
= 20

CMDLEN
UNIT
CMDOC
STATUS
MEDIA
TRANS
COUNT
START

```

```

0034 56          DRVSIN:          ;SAVE MACHINE STATE
0035 50          PUSH AX
0036 51          PUSH CX
0037 52          PUSH DX
0038 57          PUSH DI
0039 55          PUSH BP
003A 1E          PUSH DS
003B 06          PUSH ES
003C 53          PUSH BX
003D 2E: C5 1E 0025 R  ;GET POINTER TO REQUEST HEADER
0042 8A 47 01     MOV AL, BYTE PTR(BX).UNIT ;UNIT CODE
0045 8A 67 00     MOV AH, BYTE PTR(BX).MEDIA ;MEDIA DESCRIPTOR
0048 8B 4F 12     MOV CX, WORD PTR(BX).C/NMT ;COUNT
004B 8B 57 14     MOV DX, WORD PTR(BX).START ;START SECTOR
004E 50          MOV AX
004F 8A 47 02     MOV AL, BYTE PTR(BX).CMDC ;COMMAND CODE
0052 3C 08     CMP AL, 11
0054 77 10     JNA
0056 98     CBR
0057 F1 E0     SHL AX, 1
0059 B1 0008 R   MOV SI, OFFSET DRVTB. ;CONVERT TO WORD
005C 03 F0     ADD SI, AX ;INDEX TABLE
005E 58 7F 0E     POP AX ;GET BACK MEDIA AND UNIT
005F C4 7F 0E     LES DI, DWORD PTR(BX).TRANS ;ES: DI = TRANSFER ADDRESS
0062 01     CS
0063 1F     DS
0064 FF 24     JMP WORD PTR(SI) ;DO COMMAND

0066 58          ASSUME DS: NOTHING
0067 B0 03     MOV AX, 3 ;CLEAN UP STACK
0069 EB 08     MOV EB, 0 ;UNKNOWN COMMAND ERROR
006B 2E: C5 1E 0025 R  ;SHORT ERRSEXIT
0070 29 4F 12     SUB WORD PTR(BX).COUNT, CX
0073 25 81     MOV AH, 10000001B ;MARK ERROR RETURN
0075 EB 02     JMP SHORT ERR1

0077 B4 01     PROC FAR
0078 2E: C5 1E 0025 R  ;AH, 00000001B ;STATUS DONE
0079 89 47 03     MOV WORD PTR(BX).STATUS, AX ;MARK OPERATION
007E          ;COMPLETE

0081 5B          POP BX
0082 07          POP ES
0083 1F          POP DS
0084 5D          POP BP
0085 5F          POP DI
0086 5A          POP DX
0087 59          POP CX
0088 58          POP AX
0089 5E          POP SI
008A CB          RET
008B          ;RESTORE REGISTERS AND RETURN

0066 58          ASSUME DS: NOTHING
0067 B0 03     MOV AX, 3 ;CLEAN UP STACK
0069 EB 08     MOV EB, 0 ;UNKNOWN COMMAND ERROR
006B 2E: C5 1E 0025 R  ;SHORT ERRSEXIT
0070 29 4F 12     SUB WORD PTR(BX).COUNT, CX
0073 25 81     MOV AH, 10000001B ;MARK ERROR RETURN
0075 EB 02     JMP SHORT ERR1

0077 B4 01     PROC FAR
0078 2E: C5 1E 0025 R  ;AH, 00000001B ;STATUS DONE
0079 89 47 03     MOV WORD PTR(BX).STATUS, AX ;MARK OPERATION
007E          ;COMPLETE

0081 5B          POP BX
0082 07          POP ES
0083 1F          POP DS
0084 5D          POP BP
0085 5F          POP DI
0086 5A          POP DX
0087 59          POP CX
0088 58          POP AX
0089 5E          POP SI
008A CB          RET
008B          ;RESTORE REGISTERS AND RETURN

```



```

008B      33 FF      0025 R
008B      C5 1E      0025 R
0091      89 7F      0E
0094      EB E1
0096
0096      26 8A 25
0099      E8 00AB R
009C      C5 1E      0025 R
00A0      88 67      0D
00A3      89 7F      12
00A6      8C 4F      14
00A9      EB CC
00AB
00AB      50
00AC      51
00AD      52
00AE      53
00AF      8A CC
00B1      80 E1      F8
00B4      80 F9      F8
00B7      74 02
00B9      B4 FD
00BB      B0 01
00BD      BB 4008
00C0      B9 0140
00C3      BA 0101
00C6      BF 040B R
00C9      16 C4      02
00CC      75 07
00CE      FE C0
00D0      FE C3
00D2      83 C1      2J
00D5      F6 C4      01
00D8      74 08
00DA      03 C9
00DC      B7 70
00DE      FE C6
00E0      FE C2
00E2      88 75      02
00E5      88 7D      06
00E8      89 4D      08
00EB      88 65      0A
00F1      88 45      0B
00F1      88 5D      0D
00F4      88 55      0F
00F7      B8 0200
00FA      89 05
00FC      B8 0001
00FF      89 45      03

MEDIASCHK:
ASSUME DS I CODE
DI,DI ; I DON'T KNOW
RX,(PIRSAV)
WORD PTR(BX).TRANS,DI
EXIT

GETSBPB:
ASSUME DS I CODE
AH,BYTE PTR ES:(DI)
BUILD BP
BX,(PIRSAV)
(BX).MEDIA,AH
(BX).COUNT,DI
(BX).COUNT+2,CS
EXIT

BUILD BP:
ASSUME DS I CODE
:AH IS MEDIA BYTE ON ENTRY
:DI POINTS TO CORRECT ENTRY BPB ON RETURN
AX
CX
DX
BX
CL,AH
CL,0F8H
CL,0F8H
GOODID
AH,0FDH
AL,1
BX,64*256+8
CX,40*8
DX,256+1
DI,OFFSET DRVBPB
AH,0000010B ; TEST FOR 9 SECTORS
HAS8
AL
BL
CX,40
AH,00000701B
HAST
CX,CA
BH,112
DH
DL
BYTE PTR(DI).2,DH
BYTE PTR(DI).6,BH
WORD PTR(DI).8,CX
BYTE PTR(DI).10,AH
BYTE PTR(DI).11,AL
BYTE PTR(DI).13,BL
BYTE PTR(DI).15,DL
AX,512
WORD PTR(DI).0,AX
AX,1
WORD PTR(DI).3,AX

:SAVE MEDIA
:NORMALIZE
:COMPARE WITH GOOD MEDIA BYTE
:DEFAULT TO 2 SIDES 9 SECTORS
:NUMBER OF FAT SECTORS
:DIR ENTRIES AND MAX SECTORS
:SIZE OF DRIVE
:HEAD LIMIT + SECS/ALLOC UNIT
:TEST FOR 9 SECTORS
:INCREASE NO OF FAT SECTORS
:INCREASE SECTOR MAX
:INCREASE DISK SIZE
:TEST FOR TWO HEADS
:DOUBLE SIZE OF DISK
:INCREASE NO OF DIR ENTRIES
:INC SECS/ALLOC UNIT
:INC HEAD LIMIT

```

```

0102 B0 02
0104 8F 45 05
0107 5B
0108 5A
0109 59
010A 58
010B C3

010C FF
010D 0000
010F 00
0110 00
0111 0000
0113 00
0114 0000

0116 00
= 0008
= 000D
= 000F
= 0004

AL,2 PTR(DI).5,AL ;SAVE PARAMETERS
MOV BX
POP DX
POP CX
POP AX
RET

CUR DRV
SECNT
CURHD
CURSEC
CURTRK
DRIVE
NO_OF_SECTS

RETRY CNT
;BPR OFFSEIS
DRVLIM
SECLIM
HDLIM
NO_OF_TRIES

CURRNT DRIVE
;NO OF SECS TO BE TRANSFERRED
;CURRENT HEAD NUMBER
;CURRENT SECTOR NUMBER
;CURRENT TRACK NUMBER
;CURRENT DRIVE
;NO OF SECTS TRANSFERRED IN
;BIOS CALL
;RETRY COUNTER

= 8
= 13
= 15
= 4

DISK I/O HANDLERS
AH = DRIVE NUMBER
CX = MEDIA DESCRIPTOR
DX = SECTOR COUNT
DS = FIRST SECTOR
ES : DI = TRANSFER ADDRESS

IF SUCCESSFUL CARRY FLAG = 0
ELSE IF CF = 1 THEN AL CONTAINS MS-DOS ERROR CODE
CX CONTAINS NUMBER OF SECTORS NOT TRANSFERRED

ASSUME DS : CODE
JCXZ DSKOK
CALL SETUP
JC DSKSIO
CALL DISKRD
JMP SHORT DSKSIO

ASSUME DS : CODE
JCXZ DSKOK
CALL SETUP
JC DSKSIO
CALL DISKWRT

ASSUME DS : NOTHING
JNC DSKOK
JMP ERRSCNT
JMP EXIT

DSKSIO:
DSKOK:

```



```

018B 80 01      MOV     SKIP_FORTY:
018D A2 0113 R  MOV     (DRIVE),AL
0190 89 0E 010D R MOV     (SECCNT),CX
0194 92        XCHG   AX,DX
0195 33 D2      XOR     DX,DX
0197 F7 75 0D  DIV     WORD PTR(DI),SECLIM
019A FE C2      INC     DL
019C 88 16 0110 R MOV     (CURSEC),DL
01A0 8B 4D 0F   MOV     CX,WORD PTR(DI),HDLIM
01A3 33 D2      XOR     DX,DX
01A5 F7 F1      DIV     CX
01A7 88 16 010F R MOV     (CURHD),DL
01AB A3 0111 R  MOV     (CURTRK),AX
01AE F8        CLC
01AF 87 DF     XOR     BX,DI
01B1 C3        RET

01B2
01B2 8B 0E 010D R  ASSUME DS:CODE
01B3 C6 06 0116 R  MOV     CX,(SECCNT)
01B4 E8 0303 R     MOV     (RETRY_CNT),NO_OF_TRIES;INITIALISE RFRTRY
01B5 51        CALL   PRESET
01B6 57        PUSH  CX
01B7 57        PUSH  DI
01B8 01 04      MOV     CL,4
01B9 03 EF     SHR    DI,CL
01BA 50        PSH   AX
01BB 8C C6     MOV     AX,ES
01BC 03 C7     ADD    AX,DI
01BD 8E C0     MOV     ES,AX
01BE 58      POP    AX
01BF 5F      POP    AX
01C0 81 E7 000F AND    DI,000000000000011111B;LEAVE REMAINDER IN DI
01C1 8A 16 0113 R MOV     DL,(DRIVE)
01C2 8A 36 010F R MOV     DH,(CURHD)
01C3 8B 0E 0111 R MOV     CX,(CURTRK)
01C4 8A E9     MOV     CH,CL
01C5 8A 0E 0110 R MOV     CL,(CURSEC)
01C6 A1 0114 R  MOV     AX,(NO_OF_SECTS)
01C7 87 DF     XOR     BX,DI
01C8 B4 02     MOV     AH,2
01C9 CD 13     INT    13H
01CA 87 DF     MOV     AND RESET AND RETRY
01CB 72 25     JC     BX,DI
01CC 59        XCHG   BIOS_ERROR
01CD 52        POP    CX
01CE 8B 16 040B R MOV     DX,WORD PTR DRVBPB
01CF A1 0114 R  MOV     AX,(NO_OF_SECTS);AL HAS NO OF SECTS LOADED
01D0 17 F2     MUL   DX
01D1 5A        POP    DX
01D2 03 F8     ADD    DI,AX;UPDATE POINTER

DISKRD:
RDLP:
01D2 8B 0E 010D R  ASSUME DS:CODE
01D3 C6 06 0116 R  MOV     CX,(SECCNT)
01D4 E8 0303 R     MOV     (RETRY_CNT),NO_OF_TRIES;INITIALISE RFRTRY
01D5 51        CALL   PRESET
01D6 57        PUSH  CX
01D7 57        PUSH  DI
01D8 01 04      MOV     CL,4
01D9 03 EF     SHR    DI,CL
01DA 50        PSH   AX
01DB 8C C6     MOV     AX,ES
01DC 03 C7     ADD    AX,DI
01DD 8E C0     MOV     ES,AX
01DE 58      POP    AX
01DF 5F      POP    AX
01E0 81 E7 000F AND    DI,000000000000011111B;LEAVE REMAINDER IN DI
01E1 8A 16 0113 R MOV     DL,(DRIVE)
01E2 8A 36 010F R MOV     DH,(CURHD)
01E3 8B 0E 0111 R MOV     CX,(CURTRK)
01E4 8A E9     MOV     CH,CL
01E5 8A 0E 0110 R MOV     CL,(CURSEC)
01E6 A1 0114 R  MOV     AX,(NO_OF_SECTS)
01E7 87 DF     XOR     BX,DI
01E8 B4 02     MOV     AH,2
01E9 CD 13     INT    13H
01EA 87 DF     MOV     AND RESET AND RETRY
01EB 72 25     JC     BX,DI
01EC 59        XCHG   BIOS_ERROR
01ED 52        POP    CX
01EE 8B 16 040B R MOV     DX,WORD PTR DRVBPB
01EF A1 0114 R  MOV     AX,(NO_OF_SECTS);AL HAS NO OF SECTS LOADED
01F0 17 F2     MUL   DX
01F1 5A        POP    DX
01F2 03 F8     ADD    DI,AX;UPDATE POINTER

;TRAP ERRORS HERE I.F. CHECK CARRY AND RESET AND RETRY
;BIOS_ERROR
;DI AND BX RESTORED
;RESTORE COJNTFR
;PREPARE TO MULTIPLY
;LOAD SECTOR SIZE
;AL HAS NO OF SECTS LOADED
;NUMBER OF BYTES LOADED
;UPDATE POINTER

```

```

01FE A0 0110 R
0201 B4 00
0203 03 06 0114 R
0207 A2 0110 R
020A 2B 0E 0114 R
020E 83 F9 00
0211 7F A3
0213 F8
0214 C3
0215 80 FC 09
0218 74 0C
021A FE 0E 0116 R
021E 74 72
0220 B4 00
0222 CD 13
0224 EB AB
0226 51 C0
0227 8C C0
0229 B1 04
022B 03 E0
022D 03 C7
022F B9 0400
0232 03 06 0408 R
0236 72 03
0238 41 F7
0239 EB F7
023B E3 07
023D 89 0E 0114 R

0241 59
0242 EB 8D
0244 06
0245 57
0246 8C C8
0248 8E C0
024A BF 0450 R
024D C7 06 0114 R 0001
0253 8A 10 0113 R
0257 8A 36 010F R
025B 88 0E 0111 R
025F 8A L9
0261 8A 0E 0110 R
0265 A1 0114 R
0268 87 DF

026A B4 02
026C CD 13
026F 87 DF
0270 8B 0E 040B R

0274 5F
0275 07
0276 73 0C
0278 FE 0E 0116 R
027C 74 14

; NOTE IT IS ACTUALLY THE NO OF SECTORS X SECTOR SIZE
AL, (CURSEC)
AH, 0
AX, (NO OF SECTS)
(CURSEC), AL
CX, (NO OF SECTS)
CX, 0
RDLP

; CLEAR CARRY FLAG
; 1st DMA ERROR ?
; RESET FDC
; LOAD SEGMENT IN USE
; ROTATE COUNTER
; MULTIPLY BY 16
; INITIALISE COUNTER
; ADD SECTOR LENGTH
; INCREMENT COUNTER
; TRANSFER TO TEMP BUFFER
; LOAD SECTORS UP TO
; DMA OVERLAP

AH, 9
RDMA_ERROR
(RETRY_CNT)
ERROR_EXIT
AH, 0
13H
RETRY_ENTRY
CX
AX, ES
CL, 4
AX, CL
AX, DI
CX, 0
AX, WORD PTR DRVBPB
END_WHILE
CX
WHILE
RDMA_OVERLAP
(NO_OF_SECTS), CX
CX
RETRY_ENTRY
ES
DI
AX, CS
ES, AX
DI, OFFSET TEMP_BUFFER ; DIA = TEMP_BUFFER
(NO OF SECTS), 1
DL, (DRIVE)
DH, (CUPHD)
CX, (CURTRK)
CH, CL
CL, (CURSEC)
AX, (NO OF SECTS)
BX, DI

; NO OF SECTORS IN AL
; BX IS BUFFER OFFSET
; FOR BIOS
; READ FUNCTION REQUIRED
; BIOS DISK CALL
; DI AND BX RESTORED
; LOOP COUNT = SECTOR
; LENGTH
; RESTORE DTA SEGMENT
; RESTORE DIA OFFSET
; JUMP IF NO ERROR

; TRAP ERRORS HERE I.E. CHECK CARRY AND RESET AND RETRY
INT
XCHG
MOV
POP
POP
JNC
DEC
JZ
POP
POP
JNC
DEC
JZ

```

```

027E B4 00      MOV     AH,0
0280 CD 13     INT     13H
0282 EB C0     JMP     RDMA_OVERLAP
0284 56         PUSH    SI
0285 BE 0450 R MOV     SI,OFFSET TEMP_BUFFER
0288 57         PUSH    DI
0289 FC       C.LD   DI
028A F3/ A4    P.EP   DI
028C 5F         POP     DI
028D 5E       POP     SI
028E 59       POP     CX
028F E9 01F0 R   JMP     ERROR_RET

RDMA_OVERLAP:
MOV     AH,0
INT     13H
JMP     RDMA_OVERLAP
PUSH    SI
MOV     SI,OFFSET TEMP_BUFFER
PUSH    DI
C.LD   DI
P.EP   DI
POP     DI
POP     SI
POP     CX
JMP     ERROR_RET

ERROR_EXIT:
CMP     AH,3
JNZ     NR
MOV     AL,0
JMP     NR
NR:      AH,00H
CRC     CRC
AL,2
BE
AH,10H
SEEK   SEEK
AL,4
BE
AH,50H
SECT   SECT
AL,6
BE
AH,4
CMP     CHD
JNZ     SECT
MOV     BE
AH,4
CMP     BE
JNZ     SECT
MOV     BE
AH,1
CMP     BE
JNZ     SECT
MOV     BE
AH,3
JMP     BE
AL,0CH
CX

DS: CODE
CX,[SECT]
(RTRY_CNT),NO_OF_TRIES ; INITIALISE RETRY
PRESET
CX
DI
CL,4
DI,CL
AX
AX,ES
AX,DI

; RESET FDC
; TRY AGAIN
; SET AUTO INCREMENT
; TEMP TO PROPER BUFFER
; RESTORE INDEX REGISTERS
; WRITE PROTECT 7
; TIME OUT-NOT READY
; CRC ERROR
; SEEK ERROR
; SECTOR NOT FOUND
; GENERAL FAILURE
; SET CARRY FLAG
; STORE NO OF SECS
; ROTATE COUNTER
; DIVIDE DI BY 16
; UPDATE VALUE OF ES

```

```

02EB 8E C0
02ED 58
02EE 5F
02EF 81 E7 000F
02F3 8A 16 0113 R
02F7 8A 36 010F R
02FB 8B 0E 0111 R
02FF 8A E9
0301 8A 0E 0110 R
0305 A1 0114 R
0308 87 DF
030A 84 03
030C CD 13
030E 87 DF
0310 72 25
0312 59
0313 52
0314 8B 16 000B R
0318 A1 0114 R
031B 17 E2
031D 5A
031E 03 18
0320 A0 0110 R
0323 84 00
0325 03 06 0114 R
0329 A2 0110 R
032C 2B 0E 0114 R
0330 83 F9 00
0333 7F A3
0335 F8
0336 C3
0337 80 FC 09
033A 74 0C
033C FE 0E 0116 R
0340 78 78
0342 84 00
0344 CD 13
0346 EB AB
0348 51
0349 8C C0
034B B1 04
034D D3 ED
034F 03 C7
0351 89 0000
0354 03 06 000B R
0358 72 03
035A 41
035B EB F7
035D E3 07
035F 89 0E 0114 R
0363 59
0366 EB AD
; SET UP PARAMETERS FOR BIOS CALL
WRETRY_ENTRY:
MOV ES, AX
POP AX
POP DI
AND DI, 0000000000001111B ; LEAVE REMAINDER IN DI
MOV DL, (DRIVE)
MOV DH, (CURRD)
MOV CX, (CURTRK)
MOV CH, CL
MOV CL, (CURSEC)
MOV AX, (NO_OF_SECTS)
XCHG BX, DI
MOV AX, 3
INT 13H
XCHG BX, DI
AND RESET AND RETRY
WBIOS_ERROR:
JC CX
POP DX
PUSH DX
MOV AX, WORD PTR DRVBPB
MUL AX, (NO_OF_SECTS) ; AL HAS NO OF SECTS LOADED
POP DX
ADD DI, AX
MOV AL, (CURSEC)
AH, 0
MOV AX, (NO_OF_SECTS)
SUB CX, (NO_OF_SECTS)
CMP CX, 0
JG WRITLP
CLC
RET
WBIOS_ERROR:
JZ ERROR_EXIT1
JZ ERROR_EXIT1
IN:
JMP WRETRY_ENTRY
PUSH AX, ES
MOV CL, 4
MOV AX, CL
MOV CX, DI
MOV AX, WORD PTR DRVBPB
ADD END_WHILE
JC WHILE
INC JXZ
JMP WHILE
MOV JXZ, CX
MOV (NO_OF_SECTS), CX
POP JXZ
JMP WRETRY_ENTRY
; RESTORE AX
; RESTORE DI
; LEAVE REMAINDER IN DI
; DRIVE NUMBER
; HEAD NUMBER
; TRACK NUMBER
; SECTOR NUMBER
; NO OF SECTORS IN AL
; BX IS BUFFER OFFSET
; FOR BIOS
; WRITE FUNCTION REQUIRED
; BIOS DISK CALL
; DI AND BX RESTORED
; RESTORE COUNTER
; PREPARE TO MULTIPLY
; LOAD SECTOR SIZE
; AL HAS NO OF SECTS LOADED
; NUMBER OF BYTES LOADED
; UPDATE POINTER
; CLEAR CARRY FLAG
; IS DMA ERROR ?
; SET FDC
; LOAD SEGMENT IN USE
; ROTATE COUNTER
; MULTIPLY BY 16
; INITIALISE COUNTER
; ADD SECTOR LENGTH
; INCREMENT COUNTER
; TRANSFER TO TEMP BUFFER
; LOAD SECTORS UP TO
; DMA OVERLAP

```

```

0366 06          WDMA_OVERLAP:
0367 57          PUSH
0368 1E          PUSH
0369 8C          MOV     CX,ES
036B 8E          MOV     DS,AX
036D 8C          MOV     AX,CS
036F 8E          MOV     ES,AX
0371 56          PUSH    SI
0372 8B          MOV     SI,DI
0374 BF 0450 K   DI,OFFSET TEMP_BUFFER ;DIA = TEMP_BUFFER
0377 2E: 8B 0E 040B R CX,WORD PTR CS : DRVBPTR ;LOOP COUNT = SECTOR
                                ;LENGTH
                                ;SET AUTO INCREMENT
                                ;PROPER BUFFER TO TEMP

037C FC          CLO
037D F3/ AH    REP
037F 5E          POP
0380 1F          POP
0381 BF 0450 R ;DI,OFFSET TEMP_BUFFER ;DIA = TEMP_BUFFER
0384 C7 06 0114 R 0001 ;NO OF SECTS),1
038A 8A 16 0113 R DL,(DRIVE)
038E 8A 36 010F R DH,(CURHD)
0392 8B 0E 0111 R CX,(CURTRK)
0396 8A E9          CH,CL
0398 8A 0E 0110 R CL,(CURSEC)
039C A1 0114 R AX,(NO_OF_SECTS)
039F 87 DF          BX,DI

03A1 84 03          MOV     AH,3
03A3 CD 13          INT     13H
                                ;TRAP ERRORS HERE I.E. CHECK CARRY AND RESET AND RETRY
03A5 87 DF          XCHG
03A7 73 0C          JNC
03A9 FE 0E 0116 R DFC
03AD 74 0C          JZ
03AF 84 00          MOV
03B1 CD 13          INT
03B3 EB CC          JMP
03B5 5F          POP
03B6 07          POP
03B7 59          POP
03B8 F9 0312 R     JMP

03BB 5F          POP
03BC 07          POP
03BD F9 0292 R     JMP

03C0          PRESET:
03C0 2E: A0 0110 R   ASSUME DS : NOTHING
03C4 2E: 3A 47 0D   MOV     AL,(CURSEC)
03C8 76 1F          CMP     AL,CS : (BX).SECLIM
03CA 2E: 8A 36 010F R JBE
03C1 FE C6          MOV     DH,(CURHD)
03C1 2E: 3A 77 0F   INC     DH
03D5 72 07          JR     DH,CS : (BX).HDI IM
03D7 F2 F6          XOR     SETHEAD
03D9 2E: F1 06 0111 R DH,DH
03DF 2E: 8A 36 010F R INC     (CURTRK)
                                ;AFFECT NEW HEAD
                                ;AFFECT HEAD 21PO
                                ;ADVANCE TO NEXT TRACK
                                ;STORE HEAD NUMBER

                                WDMA_OVERLAP2:
                                WDMA_OVERLAP1:
                                FIX_STACK:
                                ERROR_EXIT:
                                SETHEAD:

```


03E3 80 01	MOV	AL,1	:	SET TO SECTOR 1
03E5 2E:A2 0110 R	PUSH	(CURSEC),AX	:	STORE SECTOR NUMBER
03E9 52	MOV	DX,0	:	MAKE A WORK REGISTER
03EA 84 00	MOV	AX,0	:	START SECTOR IN AX
03EC 88 D0	MOV	DX,AX	:	START SECTOR IN DX
03EE 03 D1	ADD	DX,CX	:	ADD TOTAL NUMBER OF
			:	SECTORS TO BE
			:	TRANSFERRED
03F0 2E:38 57 0D	CMP	DX,CS : (BX).SECLIM		
03F4 76 0E	JBE	ELSE		
03F6 2E:8B 57 0D	MOV	DX,CS : (BX).SECLIM		
03FA 28 D0	SUB	DX,AX		
03FC 42	INC	DX		
03FD 2E:89 16 0114 R	MOV	(NO_OF_SECTS),DX	:	NO. OF SECTS: =
			:	SECTOR LIMIT -
			:	START_SECTOR + 1
0402 5A	POP	DX		
0403 C3	RET			
0404 2E:89 0E 0114 R	MOV	(NO_OF_SECTS),CX		
0409 5A	POP	DX		
040A C3	RET			
040B 0200	DW	512	:	SECTOR SIZE IN BYTES
040D 02	DB	2	:	SECTORS PER ALLOCATION
040E 00D1	DW	1		
0410 02	DB	2		
0411 0070	DW	112		
0413 02D0	DW	9*40*2		
0415 FD	DB	1111101B		
0416 00D2	DW	2		
0418 00D9	DW	9		
041A 0062	DW	2		
041C 040B R	DW	DRVBPB		
041E 040E R	DW	DRVBPB		
0420 040B R	DW	DRVBPB		
0422 040B R	DW	DRVBPB		
0424 040B R	DW	DRVBPB		
0426 040B R	DW	DRVBPB		
0428 040B R	DW	DRVBPB		
042A 040B R	DW	DRVBPB		
042C 040B R	DW	DRVBPB		
042E 040B R	DW	DRVBPB		
0430 040B R	DW	DRVBPB		
0432 040B R	DW	DRVBPB		
0434 040B R	DW	DRVBPB		
0436 040B R	DW	DRVBPB		
0438 040B R	DW	DRVBPB		
043A 040B R	DW	DRVBPB		
043C 040B R	DW	DRVBPB		
043E 040B R	DW	DRVBPB		
0440 040B R	DW	DRVBPB		
0442 040B R	DW	DRVBPB		
0444 040B R	DW	DRVBPB		
0446 040B R	DW	DRVBPB		
0448 040B R	DW	DRVBPB		
044A 040B R	DW	DRVBPB		

```

044C 040B R
044E 040B R
0450 0200 ( 77 ) ; TEMP_BUFFER
DRVBPB
DRVB%B
512 DUP(7) ; TEMPORARY SECTOR BUFFER
BX, (PTRSAV)
AL, 24
BYTE PTR(BX), MEDIA, AL
(DRVMAX), AL
WORD PTR(BX), TRANS, OFFSET DRVSINIT
(BX), TRANS+2, CS
WORD PTR(BX), COUNT, OFFSET INITAB
(BX), COUNT+2, CS
EXIT

```

```

DRVSINIT:
ASSUME DS : CODE
ASSUME DS : NOTHING
LDS
MOV
MOV
MOV
MOV
MOV
MOV
MOV
JMP
CODE ENDS
END

```

Segments and Groups:

CODE	Name	Size	Align	Combine Class
0670	PARA	0670	PARA	NONE

Symbols:

Name	Type	Value	Attr
BE	L NEAR	0200	CODE
BIOS_ERROR	L NEAR	0215	CODE
BUILDBP	L NEAR	00AB	CODE
CMD	L NEAR	02C4	CODE
CMDC	Number	0002	CODE
CMDERR	L NEAR	0067	CODE
CMDERRP	L NEAR	0066	CODE
CMDLEN	Number	0000	CODE
COUNT	Number	0012	CODE
CRC	L NEAR	02A6	CODE
CURHD	L BYTE	010F	CODE
CURSEC	L BYTE	0110	CODE
CURTRK	L WORD	0111	CODE
CUR_DRV	L BYTE	010C	CODE
DISKRD	L NEAR	01B2	CODE
DISKPT	L NEAR	02D4	CODE
DRIVE	L BYTE	0113	CODE
DRVSIN	L NEAR	0034	CODE
DRVSINIT	L NEAR	0650	CODE
DRVSKTAD	L NEAR	0117	CODE
DRVSMR11	L NEAR	0123	CODE
DRVSMR12	L NEAR	0408	CODE
DRVIDV	L WORD	0000	CODE
DRVLM	Number	0008	CODE

DRVMAX	L	BYTE	000A	CODE
DRVIBL	L	WORD	000B	CODE
DSKIO	L	NEAR	012D	CODE
DSKX	L	NEAR	0132	CODE
ELSE	L	NEAR	0404	CODE
END WHILE	L	NEAR	023B	CODE
END WHILE	L	NEAR	035D	CODE
ERR\$CMT	L	NEAR	006B	CODE
ERR\$EXIT	L	NEAR	0073	CODE
ERR1	L	NEAR	0079	CODE
ERROR_EXIT	L	NEAR	0292	CODE
ERROR_EXIT1	L	NEAR	038D	CODE
ERROR_RET	L	NEAR	01F0	CODE
EXIT	L	NEAR	0077	CODE
EXITP	F	PROC	0077	CODE
FIX_STACK	L	NEAR	038B	CODE
GET\$PFB	L	NEAR	0096	CODE
GF	L	NEAR	02CE	CODE
GOODID	L	NEAR	008B	CODE
GUISEC	L	NEAR	03E9	CODE
HAST	L	NEAR	00E2	CODE
HASR	L	NEAR	00D5	CODE
HDLIM	Number		000F	
INITAB	L	WORD	041C	CODE
INRANGE	L	NEAR	0147	CODE
MEDIA	Number		000D	
MEDIA\$CHK	L	NEAR	008B	CODE
NO_OF_SECTS	L	WORD	0114	CODE
NO_OF_TRIES	Number		0004	
NR	L	NEAR	029C	CODE
PRESET	L	NEAR	03C0	CODE
PIRSAV	L	WORD	0025	CODE
RDL P	L	NEAR	01B6	CODE
RDMA ERROR	L	NEAR	0226	CODE
RDMA OVERLAP	L	NEAR	0244	CODE
RDMA OVERLAP1	L	NEAR	0284	CODE
RETRY CMT	L	BYTE	0116	CODE
RETRY_ENTRY	L	NEAR	01D1	CODE
SECCNT	L	WORD	010D	CODE
SECLIM	Number		0000	
SECT	L	NEAR	02BA	CODE
SEEK	L	NEAR	0280	CODE
SETBPB	L	NEAR	009C	CODE
SETHEAD	L	NEAR	03DE	CODE
SETUP	L	NEAR	0135	CODE
SKIP_FORTY	L	NEAR	018B	CODE
START	Number		0014	
STATUS	Number		0003	
STRATGY	L	NEAR	0029	CODE
STRATP	F	PROC	0029	CODE
TEMP_BUFFER	L	BYTE	0450	CODE
TRANS	Number		000E	
UNIT	Number		0001	
WRITE_ERROR	L	NEAR	0337	CODE
WRITE_ERROR	L	NEAR	0348	CODE
WRITE_OVERLAP	L	NEAR	0366	CODE
WRITE_OVERLAP1	L	NEAR	0385	CODE

Length = 0014

Length = 000B
Length = 02C0

```

NDMA_OVERLAP2. . . . .
WERROR_RET . . . . .
WHILE. . . . .
WRETRY_ENTRY . . . . .
WRITP. . . . .
WVHILE . . . . .

47012 Bytes free
Warning Severe
Errors 0

```

```

L NEAR 0381
L NEAR 0312
L NEAR 0232
L NEAR 0213
L NEAR 02D8
L NEAR 0354

```

```

CODE
CODE
CODE
CODE
CODE
CODE

```

APPENDIX N THE ENVISAGED FILE SERVER HARDWARE

The hardware required for the file-server of this star-bus network is described.

The network interface of the file-server should be fully self sufficient in so far as network management and control is concerned. The file-server as a whole can be considered as a multiprocessor environment where the one processor is dedicated to the network while the other(s) is dedicated to the mass storage system and possibly management of a cache. Communication between the two processors is by means of an input queue and an output queue which is maintained within the file-server processor's memory area. This memory area must obviously be accessed by both processors. A semaphore type system for the management of these queues would be required. Requests from the network would be submitted to the input queue of the file-server along with the appropriate data, if applicable. The file-server output would likewise be extracted from the output queue by the network processor

Each ray of the star shaped network requires a single SCC to provide the 1.5 Mbit per second data rate. These SCC's also require the use of DMA to sustain this data rate. A DMA controller such as the Intel 8237-5 would be capable of supporting four SCC's. The memory requirements of the network controller are not extensive since the only space really necessary is a buffer area capable of storing a track of data for each of the SCC's on the network. This would satisfy the situation where data movement is taking place simultaneously on all the SCC's at the same time.

The processor power required is not extensive but the Intel 80186 would be an ideal candidate for this application. A full 16 bit data bus is supported along with all the features pertaining to the 80188 described previously. This processor itself could provide a wide bandwidth DMA link

between the network processor memory and the file-server processor memory area.

Units built along these lines could be included in Multibus type file-servers or even in P.C.'s with a bus structure similar to the Olivetti M24 P.C. where it is possible to have a co-processor on the bus.

APPENDIX O EMULATION SOFTWARE

The software used during the testing of the emulator is described. The listing at the end of the appendix is divided into logical blocks for description purposes. Most of these blocks do however correspond to procedures in the software. A step by step description is not given since the principle aspects of the software have already been described and the listing is itself well commented.

Initialisation of the 80188 The initialisation of the parameters relevant to the emulation software in this appendix and Appendix P will be discussed in the order in which they appear.: **UMCS - UPPER MEMORY CHIP SELECT** The upper memory chip select line (UCS) of the 80188 is active from the location specified in the upper memory chip select up to location FFFFFH. After reset this line defaults to be active over 1K below FFFFFH. Since in this application we have a 8K EPROM which must be selected by the UCS line this register must be re-programmed. From the table in [4] if no wait states are required, as in this case, the value for UMCS is FE38H. In this case the ready inputs on the CPU are also ignored. This modifies the 3 least significant bits of the above value. The 3rd least significant bit is set if the ready inputs are ignored. Hence the value FE3CH.

Registers such as UMCS form part of a control block that may be relocated within the I/O or memory map of the 80188. In this case the control block is not relocated from its reset position. The control block starts at FFO0h in the I/C space. The UMCS register is at offset A0H from the control block base. Hence UMCSLCC (location of UMCS) is FFA0H.

In a similar manner to that described above, MPCS is programmed from a table in [4]. This register is for the mid-memory select lines MCS0-MCS3.

A 512K block, with an individual select size of 128K, is required. The ready bits (bits 0 to 3) in this register refer to the peripheral select lines PCS4 to PCS6. These bits are programmed as before, no wait states and ignore external ready inputs. Two additional bits are present MS and EX which are bits 6 and 7 respectively. MS is used to map the peripheral in the I/O or memory space. This is given the value 0 which places the peripherals in the I/O area. EX determines whether there are 7 peripheral select lines or if address lines A1 and A2 are latched on lines PCS5 and PCS6. EX is made 1 which selects 7 peripheral select lines. MPCS is therefore 01FCH and is located at offset A8H in the control block, hence MPCSLOC is FFA8H.

MMCS defines where the mid-memory block defined by MPCS starts as well as the ready bits for the mid-memory. The peculiarity of the 80188 chip select logic now rears its head. A 512K block can only start at one of two locations viz. 00000H or 80000H. This results in the mid-memory block either overlapping with the memory area that would be catered for by UCS if it is set at 80000H or with LCS (Lower Chip Select) if it starts at 00000H. In this application it is made to start at 00000H and therefore LCS is not and cannot in fact be used. For this reason if more than 512K of memory is used, the chip select lines have to be generated from UCS and the upper memory lines. Again no wait states are required and the ready inputs are ignored. MMCS is therefore 01FCH. This is located at offset A6H in the control block hence MMCSLOC is FFA6H.

PACS is used to control the peripheral chip select lines (PCS0 to PCS6). Each of these chip selects is active for one of the seven contiguous blocks of 128 bytes above a programmable base address. The only restrictions on the choice of base address is that it must be a multiple of 1K i.e. the lowest 10 bits are all 0. If the control block is in the I/O area then the addresses activated by the PCS lines must not coincide with the location of the control block. A start location of 0800H was

chosen. Two wait states are required for the 28530 SCC's [9] but again the ready inputs are ignored. PACS has a value of 085EH. for the above mentioned conditions. This register is located at offset A4H in the control block hence PACSLOC is FFA4H.

The dynamic RAM devices require that each row of the chips be refreshed every 4 mS. There are 256 rows. If we use a method where the time between refresh cycles is constant, we require that a refresh cycle be performed every 15.625 μ s. (4mS/256). Timer 2 of the 80188 is programmed to request a DMA cycle from DMA channel 0 to satisfy this requirement.

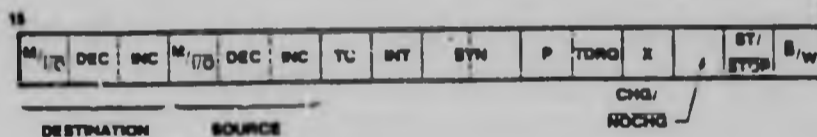
Firstly programming the timer:-

Timer 2 was chosen to perform this task since eventhough its capabilities are less than the other two, it does have what is required. It is also the only timer with an internal DMA request. Two registers need to be programmed viz. the maximum count value and the mode/control word.

The maximum count value required to give the necessary DMA request spacing is calculated as follows. The system clock frequency is 8MHz therefore the maximum clock input frequency is 2 MHz. (Hardware characteristic)[4] Therefore $T = 500\text{ns}$. The count value is thus $15.625\mu\text{s}/500\text{nS} = 31.25$. A count value of 31 will give a 15.5 μ s gap between refresh cycles which is acceptable. Hence T2MAXCNT is 31. This is loaded into location T2CNTALOC. The control word is set up so that the counter counts continuously, and interrupts from this counter are disabled. A number of other parameters are fixed as shown in [4]. This gives T2CONTW a value of C001H.

Programming DMA channel 0 will be discussed here eventhough the variables are only listed later. The DMA channel is programmed to perform memory to memory transfers. The source of these transfers is set initially at the base of the memory 00000H and is continuously incremented over the full 1Mbyte range. This has the required effect since the row addresses of the RAM chips are multiplexed from address lines A0 to A9 and thus

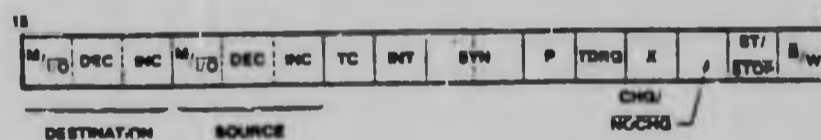
these change with each DMA cycle. All 256 rows are thus cycled through. The corresponding column addresses that may be generated are of no relevance. The destination address is fixed at a location in the EPROM range and hence the writes to this location have no effect. The terminal count is set to 65535. This gives the following values for the variables DMAOTC = 65535, DMAODPU (upper four bits of the destination address) = FFFFH, DMAODP = OFFFFH, DMAOSP = 0000H, DMAOSPU = 0000H. The suffixes SP and DP refer to source pointer and destination pointer respectively. The control word for the DMA channel is set up according to these criteria: The destination address is in the memory area and must not be incremented or decremented, the source address is in the memory area and must be incremented, DMA activity must continue after the terminal count value has been reached, no interrupt must be generated when the terminal count value is reached, the transfers are source synchronised (SYN), select highest priority (P), DMA requests from timer 2 are enabled (TDRQ), and DMA activity must start immediately (ST/STOP).



The first action taken after reset is to set the UCS line to FE000H and then a jump is made to this location where the remainder of the initialisation takes place.

The Memory Test The memory test does the following: The offset of each word is written into each word throughout the entire segment. The contents of each word, starting at the beginning of the segment, is compared with the offset of that word in the segment. These values should correspond. If they do not then the memory location at which the first mismatch occurs is faulty. A similar test is then performed except in this case the complement of the address is written in each case. Again after the whole

these change with each DMA cycle. All 256 rows are thus cycled through. The corresponding column addresses that may be generated are of no relevance. The destination address is fixed at a location in the EPROM range and hence the writes to this location have no effect. The terminal count is set to 65535. This gives the following values for the variables DMAOTC = 65535, DMAODPU (upper four bits of the destination address) = FFFFH, DMAODP = OFFFFH, DMAOSP = 0000H, DMAOSPU = 0000H. The suffixes SP and DP refer to source pointer and destination pointer respectively. The control word for the DMA channel is set up according to these criteria: The destination address is in the memory area and must not be incremented or decremented, the source address is in the memory area and must be incremented, DMA activity must continue after the terminal count value has been reached, no interrupt must be generated when the terminal count value is reached, the transfers are source synchronised (SYN), select highest priority (P), DMA requests from timer 2 are enabled (TDRQ), and DMA activity must start immediately (ST/STOP).



The first action taken after reset is to set the UCS line to FE000H and then a jump is made to this location where the remainder of the initialisation takes place.

The Memory Test The memory test does the following: The offset of each word is written into each word throughout the entire segment. The contents of each word, starting at the beginning of the segment, is compared with the offset of that word in the segment. These values should correspond. If they do not then the memory location at which the first mismatch occurs is faulty. A similar test is then performed except in this case the complement of the address is written in each case. Again after the whole

segment has been written to, the contents are read and compared with the complimented offset. Again if a mismatch is found this memory location is faulty. In this way each bit in the whole segment is checked to see if it can be set and reset.

The code to do this test is shown in PROCEDURE TESTRAM in the assembler listing at the end of this appendix. The unusual exit from the procedure is necessary because the first time the routine is used, a jump has to be made to the routine instead of a call. There is nowhere to put the return address for a call instruction because the stack can only be set up after the memory is checked. The routine is therefore 'jumped' to check the top most 64K segment. If this memory is functioning the stack is set up at the upper end of this segment. All subsequent memory checks are made using a call to the routine. Each phase of the memory check is indicated on the diagnostic L.E.D.'s.

Initialisation and programming of the Z8530 SCC (Emulator SCC) The method of programming of this device and the initialisation required for the emulation software is described. This is done in the Z8530B_INIT procedure.

If we consider only one channel of the SCC on the hardware level we find that only two I/O locations are used. The SCC however has nine read registers and sixteen write registers. Some of these registers are shared by both channels [10]. Read register 0 and write register 0 can be accessed directly by the hardware by addressing the control register of that channel. Write register 0 has its four least significant bits that can be programmed as a 'vector' to a required register. If one wants to write to register 10 for example a write is first made to write register 0 to set up the 'vector' to register 10. A write is again made to the same I/O location with the data required for register 10. The 'vector' is then reset back to 0 after this write. Similarly a read from a register other than read register 0 requires a write to register 0 to set up the vector

followed by a read from register 0. The transmitter buffer (Write register 8) and receiver buffer (Read register 8) are exceptions to this rule and can be accessed directly by addressing the data register of the appropriate channel on the hardware level.

The SCC is programmed to function as a Bi - sync receiver / transmitter using NRZ coding. It obtains both its transmitter and receiver clocks from external sources. The Bi - sync mode was chosen so that two synchronisation characters could be specified and also since the bit-stuffing used in SDLC/HDLC communications is not desired.

The programming of the individual registers will be discussed along with the programming method.

The procedure called WR was written to perform the dual writes required for most of the registers as described above. This routine requires that the DA register of the CPU contain the address of the port to be written to, the AH register contain the required SCC register to be written to and AL contain the value to be written to the SCC.

The equates used in the assembler are the following:

Z8530BC - Control register of channel B of the emulator SCC.

Z8530BD - Data register of channel B of the emulator SCC.

Z8530BDREV - Data register of channel B of the emulator SCC but with the data bits reversed.

Because of the dual write programming described above it is possible that when one starts programming the SCC the 'vector' in register 0 is set pointing to some other register. Before the device can be programmed therefore it should be brought into a known state. (In this application this could be regarded as a redundant step since the SCC is reset on power up). By performing a write to register 0 requesting register 0 and then a read from register 0 we can be sure the pointer is set back to zero. If the case where the vector is zero initially is considered, by re-

questing register zero we have left the vector unchanged and the read will be from register 0. If the vector has some other value in it then the writing of a zero will put a zero in which ever register the vector was pointing to and reset the vector to 0.

The channel is reset and then configured.

The configuration selected is: x 1 clock mode, 16 bit synchronisation character, synchronous communications mode enabled, no parity.

The clock sources are: No XTAL. I.e. the internal oscillator of the SCC is not used, the receiver clock is input on the RTxC pin (from the MFM decoder), the transmitter clock is input on the TRxC pin (from the MFM encoder).

Register 10 has various control bits that are programmed as follows: The CRC generator and checker are preset to all ones, NRZ coding is to be used, a full 16 bit synchronisation character is required.

The control bits in register 14 are programmed to disable the Digital Phase Locked Loop in the SCC. All the interrupt enable bits in registers 1 and 15 are all disabled.

The initial synchronisation characters programmed are 4EH bytes corresponding to the contents of the gap 1. See Figure P1. Note that the bits of these characters are reversed since when they are written to the SCC they are not reversed by the hardware.

Finally the receiver error bits are reset. The transmitter and receiver themselves are not enabled until the track emulation routine is called. The actual CRC polynomial required is chosen by one of the bits that is in the same register as the transmitter enable bit. The required CRC is that used in SDLC communications viz. $x^{16} + x^{12} + x^5 + 1$ and is selected by resetting bit 2 of register 5.

A brief description of each of the remaining routines.

VEC_TBL_INIT Eventhough interrupts are not used in the software the interrupt vector table is initialised to a known state. Each of the vec-

tors, which consists of the segment and offset at which the interrupt routine resides, is directed to a dummy interrupt routine DUMMY RET that merely returns to the point in the program at which the interrupt occurred.

NET_INIT Initialisation of the SCC channel consists of setting up the transmitter and receiver for SDLC communication. The DMA channel associated with the network is also initialised. Each of the parameters for both the SCC and the DMA controller are described in the order that they are used.

NET_CONFIG - x 1 clock, SDLC mode, synch modes enabled, no parity

NET_CLK_SOURCE - No xtal, Receive clock = RTxC PIN, Transmit clock = Output of the baud rate generator, TRxC is an output and carries the transmit clock. (Note in the testing described the transmit clock was from an external source)

NET_CRC_MODE - Preset CRC to all 1's, NRZ coding, send flags on an idle line, send flags after a transmitter underrun, disable loop mode.

BBRGENLO and **BBRGENHI** - Divisors used for the baud rate generators. (Note the values used here give a lower transmission speed than that used in testing)

FLAG - Contains the SDLC flag character (7EH)

NET_ADDRESS - Address that the receiver will respond to if programmed to operate in address search mode.

DIS_INT - Set all the interrupt enable bits to disabled

NET_DISDPLL - Local loop-back disabled, no auto echo, set baud rate generator source to the SCC clock input and enable the baud rate generator.

NET_RX_CONFIG - 8 character bits, all other functions disabled

DMA1TC - DMA terminal count set to 1 less than the network packet length.

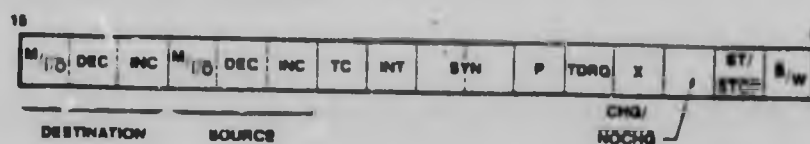
DMA1DPU - Upper four bits of the destination pointer set to zero.

DMA1DP - Destination of DMA transfers set as the data register of the network SCC.

DMA1SPU - Upper four bits of the source pointer set to zero.

DMA1SP - Source of DMA transfers set to the NETWORK_PACKET address.

NETDMA1CW - The control word is set up as follows - Destination in I/O map and address to remain unchanged, source in memory area and must be incremented after each transfer, DMA activity to cease after terminal count reached, no interrupt on terminal count, destination synchronisation (SYN), low priority channel (P), disable timer 2 DMA request (TDRQ), start immediately (ST/STOP).



NET_TX_CONFIG - 8 bits per character, all other functions disabled.

NET_SEND This routine generates data for the test network packet and initiates its transmission from the SCC. The information included in the packet includes the address of the node to which the packet is being sent as well as the current variables from the emulator. The first data byte is loaded into the transmitter before resetting the EUM latch for the CRC transmission at the end of the packet. The DMA request function for the transmitter is also enabled.

TRACK_LIST_INIT The track cache index has to be initialised before use. In this case the 42 available data blocks are assigned to the two sides of the first 21 tracks of the disk. The drive number and application number are initialised to values that would not normally exist during the emulation process. The various records making up the index are initialised as well as the pointers to the respective data areas.

TRACK0_INIT This routine initialises track 0 on the emulator to the state that the corresponding track on a physical disk would be in immediately after formatting. Procedures FIRST, SECOND and SIXTH are used to initialise individual sectors to their required states. FIRST is used to fill the bootstrap sector (sector 1) and the second sectors of each of the File Allocation Tables with zeros i.e. sectors 3 and 5. SECOND gen-

erates the first sectors of each of the File Allocation Tables. SIXTH generates the contents of each of the sectors occupied by the directory.

SCAN_TRACK_LIST This routine returns a status whether a particular track that is required has been found or not. If the search is successful a pointer to the data area required in the variable TRACK_SEG. Parameters passed to the routine include the required drive, application number, track number and side number.

Note that in the assembler listing that follows a number of brackets that would normally be square have been replaced by round brackets for typographical reasons.

SCC REGISTERS

REG0	EQU	000010000B	DISABLE INTERRUPTS
REG1	EQU	000010000B	X1 CLK, 16 BIT SYNCH, SYNCH EN, NO PARITY
REG2	EQU	000010000B	X1 CLK, 16 BIT SYNCH, SYNCH EN, NO PARITY
REG3	EQU	000010000B	REG13
REG4	EQU	000010000B	REG12
REG5	EQU	000010000B	REG11
REG6	EQU	000010000B	REG10
REG7	EQU	000010000B	REG9
REG8	EQU	000010000B	REG8
REG9	EQU	000010000B	REG7
REG10	EQU	000010000B	REG6
REG11	EQU	000010000B	REG5
REG12	EQU	000010000B	REG4
REG13	EQU	000010000B	REG3
REG14	EQU	000010000B	REG2
REG15	EQU	000010000B	REG1
DIS_INT	EQU	000010000B	DISABLE INTERRUPTS
CONFIG_B	EQU	000010000B	X1 CLK, 16 BIT SYNCH, SYNCH EN, NO PARITY
CONFIG_A	EQU	000010000B	X1 CLK, 16 BIT SYNCH, SYNCH EN, NO PARITY
BBREN11	EQU	000010000B	REG11
BBREN10	EQU	000010000B	REG10
CLK_SOURCE	EQU	000010000B	NO XTAL, R×CLK=RX×C PIN, TXCLK=TR×C PIN
CLK_SOURCE	EQU	000010000B	NO XTAL, R×CLK=RX×C PIN, TXCLK=TR×C PIN
A_TX_EN	EQU	000000010B	DIR MUST BE 0, USED IN B
BBREN_EN	EQU	000000010B	REG14
DMAENABLE	EQU	110000000B	REG1 CHANNEL A
RESET_B	EQU	010000000B	RESET CHANNEL B
RESET_A	EQU	100000000B	RESET CHANNEL A
DISDPL	EQU	011000000B	DISABLE DPLL
B_RX_EN	EQU	110000010B	8 BITS PER CHAR, R× EN
B_TX_EN	EQU	011010100B	8 BITS PER CHAR, TX EN, SDLC CRC, RTS
TX_CRC_EN	EQU	000000010B	ENABLE TX CRC
DIR	EQU	100000000B	ASSERT DIR
RISOFF	EQU	111111010B	DISABLE RTS
ENTER_HUNT	EQU	000100000B	ENTER HUNT MODE
RX_CRC_EN	EQU	000010000B	ENABLE CRC GENERATOR
RESET_TX_CRC	EQU	100000000B	RESET TX CRC GENERATOR
RESET_EOM_LATCH	EQU	110000000B	RESET EOM LATCH
SYNC_LD_INHIB	EQU	000000100B	INHIBIT SYNC LOADING
ERR_RESET	EQU	001100000B	RESET ERRORS
B_CRC_MODE	EQU	100000000B	NRZ_CRC PRESET = 1, 8 BIT SYNCH
CHAR_READY	EQU	000000010B	RECEIVE BUFFER HAS A VALID CHARACTER
TX_BUFFER_FMP_V	EQU	000001000B	TRANSMIT BUFFER IS READY FOR DATA
EOM	EQU	010000000B	END OF MESSAGE/TRANSMITTER UNDERRUN
SYNCRS	EQU	000000000B	SYNCH CHAR FOR TX (4EH)
SYNCRX	EQU	000000000B	SYNCH CHAR FOR RX (A1H)
NET_CLK_SOURCE	EQU	000101010B	
NET_DISDPL	EQU	000001010B	
NET_ADDR15	EQU	000000010B	
NET_ADDR14	EQU	000000010B	
NET_ADDR13	EQU	000000010B	
NET_ADDR12	EQU	000000010B	
NET_ADDR11	EQU	000000010B	
NET_ADDR10	EQU	000000010B	
NET_ADDR9	EQU	000000010B	
NET_ADDR8	EQU	000000010B	
NET_ADDR7	EQU	000000010B	
NET_ADDR6	EQU	000000010B	
NET_ADDR5	EQU	000000010B	
NET_ADDR4	EQU	000000010B	
NET_ADDR3	EQU	000000010B	
NET_ADDR2	EQU	000000010B	
NET_ADDR1	EQU	000000010B	
NET_ADDR0	EQU	000000010B	

SCC REGISTERS

REG0	EQU	0000
REG1	EQU	0100
REG2	EQU	0200
REG3	EQU	0300
REG4	EQU	0400
REG5	EQU	0500
REG6	EQU	0600
REG7	EQU	0700
REG8	EQU	0800
REG9	EQU	0900
REG10	EQU	0A00
REG11	EQU	0B00
REG12	EQU	0C00
REG13	EQU	0D00
REG14	EQU	0E00
REG15	EQU	0F00
DIS_INT	EQU	00010000B
CONFIG_B	EQU	00010000B
CONFIG_A	EQU	00010000B
DRIVER_INT	EQU	25
BURSTEN0	EQU	00001000B
CLK_SOURCE	EQU	00001000B
ACK_SOURCE	EQU	00001000B
A_TX_EN	EQU	01101010B
BURSTEN1	EQU	00000011B
DMAREABLE	EQU	11000000B
RESET_B	EQU	01000000B
RESET_A	EQU	10000000B
DISUPFL	EQU	01100000B
B_RX_EN	EQU	11000001B
B_TX_EN	EQU	01101010B
TX_CRC_EN	EQU	00000001B
DIR	EQU	10000000B
RSOFF	EQU	11111010B
ENTER_HUNT	EQU	00010000B
RX_CRC_EN	EQU	00001000B
RESET_TX_CRC	EQU	10000000B
RESET_FOM_LATCH	EQU	11000000B
SYNC_LD_INITB	EQU	00000010B
ERR_RESET	EQU	00110000B
B_CRC_MODE	EQU	10000000B
CHAR_READY	EQU	00000001B
TX_BUFFER_EMPTY	EQU	00000100B
E_O_N	EQU	01000000B
SYNCH5	EQU	72H
SYNCH4	EQU	85H
NET_CLK_SOURCE	EQU	00010101B
BIT_FLUSH	EQU	00000011B
BIT_ADDR155	EQU	0A1H
FLAG	EQU	7EH
NET_CRC_MODE	EQU	10000000B
NET_TX_EN	EQU	00001000B
NET_TX_CONFIG	EQU	01100000B
ADDR_SEARCH	EQU	00000000B
NET_CONFIG	EQU	00000000B
NET_RX_CONFIG	EQU	11000000B
NET_RX_EN	EQU	00000001B

DISABLE INTERRUPTS
 X1 CLK, 16 BIT SYNCH, SYNCH EN, NO PARITY
 X1 CLK, 16 BIT SYNCH, SYNCH EN, NO PARITY
 REG13
 REG 12
 NO XTAL, R×CLK=RX×C PIN, TX×CLK=TX×C PIN
 NO XTAL, R×CLK=RX×C PIN, TX×CLK=TX×C PIN
 DIR MUST BE 0, USED IN B
 REG14
 REG1 CHANNEL A
 RESET CHANNEL B
 RESET CHANNEL A
 DISABLE DPLL
 8 BITS PER CHAR, RX EN
 8 BITS PER CHAR, TX EN, SILEC CRC, RTS
 ENABLE TX CRC
 ASSERT DIR
 DISABLE RTS
 ENTER HUNT MODE
 ENABLE CRC GENERATOR
 RESET TX CRC GENERATOR
 RESET FOM LATCH
 INHIBIT SYNC LOADING
 RESET ERRORS
 NRZ CRC PRESET = 1, 8 BIT SYNCH
 RECEIVE BUFFER HAS A VALID CHARACTER
 TRANSMIT BUFFER IS READY FOR DATA
 END OF MESSAGE/TRANSMITTER UNDERRUN
 SYNCH CHAR FOR TX (REH)
 SYNCH CHAR FOR RX (ATH)

```

EQUATES FOR DMA CHANNEL 0
2  EQU
20 MESSAGE_LENGTH EQU
   DMA0CWL0C EQU
   DMA01CL0C EQU
   DMA0DPUL0C EQU
   DMA0DPLOC EQU
   DMA0SPUL0C EQU
   DMA0SPLOC EQU
   DMA0CWL EQU
   DMA01C EQU
   DMA0DP EQU
   DMA0DP EQU
   DMA0SP EQU
   DMA0SP EQU
   DMA1CL0C EQU
   DMA1CL0C EQU
   DMA1DP3LOC EQU
   DMA1DPLOC EQU
   DMA1SPULOC EQU
   DMA1SPLOC EQU
   DMA1CWL EQU
   MET_DMA1CWL EQU
   DMA1C EQU
   DMA1DPU EQU
   DMA1DPU EQU
   DMA1SP EQU
   DMA1SP EQU
   DRIVE_NO_OFF EQU
   APPL_NO_OFF EQU
   TRACK_NO_OFF EQU
   SIDE_NO_OFF EQU
   MASK_OFF EQU
   XBRREQ_OFF EQU
   XBRREQ_OFF EQU
   DATA_POINTER_OFF EQU
   RECORD_LENGTH EQU
   CACHE_MASK EQU
   WRITE_COUNT_MASK EQU

EQUATES FOR DMA CHANNEL 1
   REPROGRAM_UCS EQU
   JUMP_TO_EPROM_BASE EQU
   EPROM_BASE EQU
   INITIALISE_80188_MEMORY_MAP EQU
   INITIALISE_80188_I/O_MAP EQU
   INITIALISE_TIMER_2_FOR_RAM_REFRESH EQU

OFFSET_NETWORK_PACKET + 1
OFFSETS_WITHIN_RECORD_STRUCTURE
IR_TRACK_LIST
RESET_LOCATION
REPROGRAM_UCS
JUMP_TO_EPROM_BASE
EPROM_BASE
INITIALISE_80188_MEMORY_MAP
INITIALISE_80188_I/O_MAP
INITIALISE_TIMER_2_FOR_RAM_REFRESH

2  EQU
20 MESSAGE_LENGTH EQU
   DMA0CWL0C EQU
   DMA01CL0C EQU
   DMA0DPUL0C EQU
   DMA0DPLOC EQU
   DMA0SPUL0C EQU
   DMA0SPLOC EQU
   DMA0CWL EQU
   DMA01C EQU
   DMA0DP EQU
   DMA0DP EQU
   DMA0SP EQU
   DMA0SP EQU
   DMA1CL0C EQU
   DMA1CL0C EQU
   DMA1DP3LOC EQU
   DMA1DPLOC EQU
   DMA1SPULOC EQU
   DMA1SPLOC EQU
   DMA1CWL EQU
   MET_DMA1CWL EQU
   DMA1C EQU
   DMA1DPU EQU
   DMA1DPU EQU
   DMA1SP EQU
   DMA1SP EQU
   DRIVE_NO_OFF EQU
   APPL_NO_OFF EQU
   TRACK_NO_OFF EQU
   SIDE_NO_OFF EQU
   MASK_OFF EQU
   XBRREQ_OFF EQU
   XBRREQ_OFF EQU
   DATA_POINTER_OFF EQU
   RECORD_LENGTH EQU
   CACHE_MASK EQU
   WRITE_COUNT_MASK EQU

OFFF0000H
DX,IMCSLOC
AX,IMCS
DX,AX
0000H,OFF00H
0FED0000H
DX,MPCSLOC
AX,MPCS
DX,AX
DX,IMCSLOC
AX,IMCS
DX,AX
DX,PACSL0C
AX,PACS
DX,AX
DX,I2CNTAL0C
AX,I2MAXCNT

OFFF0000H
DX,IMCSLOC
AX,IMCS
DX,AX
0000H,OFF00H
0FED0000H
DX,MPCSLOC
AX,MPCS
DX,AX
DX,IMCSLOC
AX,IMCS
DX,AX
DX,PACSL0C
AX,PACS
DX,AX
DX,I2CNTAL0C
AX,I2MAXCNT

```

```

OUT DX, AX
MOV DX, I2CONTWLOC
MOV AX, I2CONTW
OUT DX, AX
MOV DX, DMA01CLOC
MOV AX, DMA01C
OUT DX, AX
MOV DX, DMA01DPULOC
MOV AX, DMA01DPU
OUT DX, AX
MOV DX, DMA01PLUC
MOV AX, DMA01P
OUT DX, AX
MOV DX, DMA01SPULOC
MOV AX, DMA01SPU
OUT DX, AX
MOV DX, DMA01SPLOC
MOV AX, DMA01SP
OUT DX, AX
MOV DX, DMA01CHLOC
MOV AX, DMA01C
OUT DX, AX
CLD
MOV DX, OUTPORT
OUT DX, AL
MOV DX, I000H
TESTRAM
JMP AX, I000H
MOV SS, AX
MOV SP, 0FFFFH
AL, 2
MOV DX, OUTPORT
OUT DX, AL
MOV DX, I000H
TESTRAM
CALL AL, 1
MOV DX, OUTPORT
OUT DX, AL
MOV DX, I000H
TESTRAM
CALL AL, 80H
MOV DX, OUTPORT
OUT DX, AL
MOV DX, I000H
TESTRAM
CALL AL, 0000H
MOV DX, I000H
TESTRAM
CALL VEC_TBL_INIT
MOV AX, I000H
DS, AX
MOV BYTE PTR(CURRENT_DRIVE), 0FFH
MOV WORD PTR(CURR_APPLN), 0FFFFH
CALL TRACK_LIST_INIT
CALL TRACK0_INIT
CALL Z8530B_INIT
CALL NET_INIT
MOV AX, I000H
MOV DS, AX

```

INITIALISE DMA CONTROLLER CHANNEL 0 FOR RAM REFRESH

INDICATE ON L.E.D.'S THAT BLOCK 3 OF MEMORY IS TO BE TESTED

SET DX TO THE LAST 64K SEGMENT OF MEMORY
JUMP TO RAMTEST

SET STACK SEGMENT TO END OF RAM
SET POINTER TO LAST ADDRESS

INDICATE ON L.E.D.'S THAT BLOCK 2 OF MEMORY IS TO BE TESTED
SET DX TO SECOND LAST 64K SEGMENT
CALL RAMTEST

INDICATE ON L.E.D.'S THAT BLOCK 1 OF MEMORY IS TO BE TESTED
SET DX TO SECOND 64K SEGMENT
CALL RAMTEST

INDICATE THAT BLOCK 0 IS TO BE TESTED

SET DX TO FIRST 64K BLOCK OF MEMORY
CALL RAMTEST
INITIALIZE THE INTERRUPT VECTOR TABLE

SET DS TO VARIABLES SEGMENT
BYTE PTR(CURRENT_DRIVE), 0FFH
WORD PTR(CURR_APPLN), 0FFFFH
CALL TRACK_LIST_INIT
CALL TRACK0_INIT
CALL Z8530B_INIT
CALL NET_INIT
MOV AX, I000H
MOV DS, AX

TESTRAM2

J2

```

MOV DX,Z8530BC
MOV AH,0
IN AL,DX
TEST AL,00100000B
JZ SIDED
MOV AH,1
MOV (SIDE_NO),AH
MOV DX,INPORT2
IN AL,DX
MOV (TRACK_NO),AL
CMP AL,40
JZ TRACK_40
MOV DX,OUTPORT
OUT DX,AL
MOV AH,(CURRENT_DRIVE)
MOV BX,(CURR_APPLN)
MOV DH,AL
MOV (SIDE_NO)
CALL SCAN_TRACK_LIST
CMP AH,D
JNZ WAS_TRACK_FOUND
MOV WORD PTR(SECTOR_NO),1
MOV WORD PTR(SECTS_PER_TRACK),9
MOV TRACK_EM
JZ WORD PTR(TRACK_SEG),0
MOV BYTE PTR(SECTS_PER_TRACK),1
MOV BYTE PTR(SECTOR_NO),1
MOV NET_SEMD
CALL TRACK_EM
JMP J2

PROC TESTRAM
MOV MEMIST
MOV XDR MEMIST_W1
MOV STOSW
LOOP MOV
MOV XDR MEMIST_R1
MOV MEMIST_R1
CMP MEMIST_ERR
INC JNE
INC LOOP
MOV MEMIST_W2
MOV STOSW
MOV XDR MEMIST_R2
MOV MEMIST_R2
MOV NOT
CMP AX,BX

```

SET TO SIDE 0
 READ SCC STATUS
 CHECK CTS (HEAD 1)
 SET TO SIDE 1
 STORE SIDE NUMBER
 TRACK COUNTER PORT
 READ TRACK REQUESTED
 STORE REQUIRED TRACK
 IS TRACK 40 ?
 PUT TRACK NUMBER OUT ON L.E.D.'S
 GET PARAMETERS REQUIRED FOR THE INDEX SCAN
 WAS TRACK FOUND
 SET STARTING SECTOR NUMBER
 SET THE NUMBER OF SECTORS ON A TRACK
 EMULATE THE REQUIRED TRACK
 SET DATA POINTER TO THE TRACK 40 MESSAGE BUFFER
 SET THE NUMBER OF SECTORS ON A TRACK
 SET THE STARTING SECTOR NUMBER
 SEND NETWORK MESSAGE
 EMULATE TRACK FORTY
 SET COUNTER TO 32K WORDS
 SET EXTRA SEGMENT REGISTER TO SEGMENT BEING TESTED
 SET DESTINATION OFFSET TO 0
 SET DATA TO BE WRITTEN EQUAL TO THE OFFSET
 STORE AX AT OFFSET DI AND INCREMENT DI
 LOOP BACK
 SET DATA SEGMENT REGISTER TO SEGMENT BEING TESTED
 SET BX REGISTER TO 0
 SET COUNTER TO 32K WORDS
 MOVE THE CONTENTS OF THE LOCATION AT OFFSET BX INTO AX
 ARE THE CONTENTS THE SAME AS THE OFFSET
 JUMP IF THEY ARE NOT THE SAME
 INCREMENT THE OFFSET
 TWICE SINCE READING WORDS
 LOOP BACK
 SET COUNTER TO 32K WORDS
 SET DATA TO BE WRITTEN EQUAL TO THE OFFSET
 INVERT THE DATA TO BE WRITTEN
 WRITE THE DATA
 LOOP BACK
 SET COUNTER TO 32K WORDS
 READ CONTENTS AT OFFSET BX
 INVERT THE DATA READ
 IS THIS DATA THE SAME AS THE OFFSET


```

MOV AL, BCLK_SOURCE
CALL SET_UP_CLOCK_SOURCE
MOV AH, REG10
MOV PRESET_CRC
MOV AL, B_CRC_MODE
MOV AH, REG14
MOV DISABLER_THE_DPLL
MOV AL, DISDPLL
MOV AH, REG15
MOV DISABLER_INTERRUPTS
MOV AL, DIS_INT
MOV AH, REG1
MOV DISABLER_INTERRUPTS
MOV AL, DIS_INT
MOV AH, REG6
MOV PROGRAM_SYNCH_CHARACTER (4EH REVERSED)
MOV AL, SYNCHS
MOV AH, REG7
MOV PROGRAM_SYNCH_CHARACTER (4EH REVERSED)
MOV AL, SYNCHS
MOV AL, ERR_RESET
MOV RESET_RECEIVER_ERROR_LATCH
MOV DX, AL

PROC
MOV AX
MOV AH, 0
MOV REG_WR
MOV AL, AH
MOV DX, AL
MOV WAIT
CALL RETRIEVE_PARAMETERS
MOV AX
MOV DX, AL
RET

PROC
MOV AX
MOV CX, 1
MOV W1
MOV CX
RET

PROC
MOV AX, SEG_VEC_TABLE
MOV DS, AX
MOV BX, SEG_DUMMY_RET_LOAD_BX_WITH_SEGMENT_OF_THE_DUMMY_ROUTINE
MOV AX, OFFSET_DUMMY_RET_LOAD_AX_WITH_THE_OFFSET_OF_THE_DUMMY_ROUTINE
MOV SI, 0
MOV DS, 255
MOV DS: (SI), AX
MOV SI
MOV SI
MOV DS: (SI), BX
MOV SI
RET

PROC
MOV AX
MOV CX, 1
MOV W1
MOV CX
RET

PROC
MOV AX, SEG_VEC_TABLE
MOV DS, AX
MOV BX, SEG_DUMMY_RET_LOAD_BX_WITH_SEGMENT_OF_THE_DUMMY_ROUTINE
MOV AX, OFFSET_DUMMY_RET_LOAD_AX_WITH_THE_OFFSET_OF_THE_DUMMY_ROUTINE
MOV SI, 0
MOV DS, 255
MOV DS: (SI), AX
MOV SI
MOV SI
MOV DS: (SI), BX
MOV SI
RET

PROC
MOV AX
MOV CX, 1
MOV W1
MOV CX
RET

```

INC	SI	INIT_LOOP	RESTORE DS
LOOP	DS		
POP			
RET			
PROC	FAR		
PUSH	DX		
PUSH	AX		
MOV	AL, 0FH		
MOV	DX, OUTPORT		
OUT	DX, AL		
POP	AX		
POP	DX		
IRET	DX	RETURN FROM INTERRUPT	
PROC	NEAR		
MOV	DX, 20530A20	;DX TO CHANNEL A CONTROL REGISTER	
MOV	AL, REG0		
OUT	DX, AL		
CALL	WAIT	;WRITE RECOVERY TIME	
IN	AL, DX	;ENSURE POINTER IS AT REG 0	
CALL	WAIT		
MOV	AH, REG9	;SELECT REG 9	
MOV	AL, RESET_A	;RESET CHANNEL A	
CALL	WR		
MOV	AH, REG4	;SELECT REG 4	
MOV	AL, NET_CONFIG	;CONFIGURE CHANNEL	
CALL	WR		
MOV	AH, REG11	;SELECT REG 11	
MOV	AL, NET_CLK_SOURCE	;SET UP CLOCK SOURCE	
CALL	WR		
MOV	AH, REG10	;SELECT REG 10	
CALL	AL, NET_CRC_MODE	;PRESET CRC	
MOV	AH, REG12	;SELECT REG 12	
CALL	AL, BBRGENLO	;BAUD DIVISOR (LO)	
MOV	AH, REG13	;SELECT REG 13	
CALL	AL, BBRGENHI	;BAUD DIVISOR (HI)	
MOV	AH, REG7	;SELECT REG 7	
CALL	AL, FLAG	;SDLC FLAG	
MOV	AH, REG6	;SELECT REG 6	
CALL	AL, NET_ADDRESS	;ADDRESS OF RECEIVER	
MOV	AH, REG15	;SELECT REG 15	
CALL	AL, DIS_INT		
MOV	AH, REG14	;SELECT REG 14	
CALL	AL, NET_DISDPOLL		
MOV	AH, REG3	;SELECT REG 3	
CALL	AL, NET_RX_CONFIG	;CONFIGURE RECEIVER	
MOV	DX, DMA11CLOC		
MOV	AX, DMA11C		

```

NET_SEND
OUT MOV PIRNET_PACKET(0),NET_ADDRESS
MOV PIRNET_PACKET(1),MODE_NO
OUT MOV AL,(CURRENT_DRIVE)
MOV PIRNET_PACKET(2),AL
MOV AL,(CURR_APPLN)
MOV WORD PIRNET_PACKET(3),A
MOV AL,(TRACK_NO)
MOV BYTE PIRNET_PACKET(5),AL
MOV BYTE PIRNET_PACKET(6),AL
MOV DX,Z8530A2D
MOV AL,NET_ADDRESS
OUT DX,AL
MOV DX,Z8530A2C
MOV AH,REG0
CALL ;SELECT REG 1
MOV AH,REG1
CALL ;ENABLE DMA REQUEST FUNCTION
OR AL,DMAENABLE
MOV AX
POP AX
RET

```

```

OUT DX,AX
MOV DX,DMAIDPULOC
MOV AX,DMAIDPU
OUT DX,AX
MOV DX,DMAIDPLOC
MOV AX,DMAIDP
OUT DX,AX
MOV DX,DMAISPULOC
MOV AX,DMAISPU
OUT DX,AX
MOV CX,DMAISPLOC
MOV AX,DMAISF
OUT DX,AX
MOV DX,DMAICHLDC
MOV AX,NET_DMA1CM
OUT DX,AX
MOV DX,Z8530A2C
MOV AH,REG5
CALL ;SELECT REG 5
MOV AL,NET_TX_CONFIG
CALL ;CONFIGURE TX
OR AL,NET_TX_EN
CALL ;ENABLE TRANSMITTER
OR AL,TX_CRC_EN
CALL ;ENABLE CRC GEN
MOV WR
MOV AH,REG0
CALL ;SELECT REG 0
MOV AL,ERR_RESET
CALL ;RESET RX ERRORS
MOV AL,RESET_TX_CRC
CALL ;RESET TX CRC GEN
MOV WR

```

```

NEAR
DS DS
AX AX
AX,0
DS,AX
BYTE PIRNET_PACKET(0),NET_ADDRESS
BYTE PIRNET_PACKET(1),MODE_NO
AL,(CURRENT_DRIVE)
BYTE PIRNET_PACKET(2),AL
AX,(CURR_APPLN)
WORD PIRNET_PACKET(3),A
AL,(TRACK_NO)
BYTE PIRNET_PACKET(5),AL
BYTE PIRNET_PACKET(6),AL
DX,Z8530A2D
AL,NET_ADDRESS
DX,AL
DX,Z8530A2C
AH,REG0
CALL ;SELECT REG 1
AH,REG1
CALL ;ENABLE DMA REQUEST FUNCTION
WR
WAIT
MOV AH,REG1
CALL ;SELECT REG 1
AL,DMAENABLE
CALL ;ENABLE DMA REQUEST FUNCTION
WR
AX
DS

```

TRACKEM: LAN INCLUDE TRACK EMULATION CODE

```

TRACK_LIST_INIT PROC
ASSUME
MOV
MOV
MOV
MOV
PUSH
DEC
CLC
RCR
JC
MOV
JMP
MOV
MOV
POP
MOV
MOV
MOV
ADD
ADD
LOOP
RET

```

TRACK_INIT_LP:

```

NEAR
DS:DATA,CS:PROG ASSUMES DS = 0
CX,42
SI,0
AX,4096
BYTE PTR(TRACK_LIST)(SI+DRIVE_NO_OFF),OFFH
WORD PTR(TRACK_LIST)(SI+APPL_NO_OFF),OFFFH
CX
CL
CL,1
SIDE ONE
BYTE PTR(TRACK_LIST)(SI+SIDE_NO_OFF),0
TRK_NO
BYTE PTR(TRACK_LIST)(SI+SIDE_NO_OFF),1
(TRACK_LIST)(SI+TRACK_NO_OFF),CL
CX
BYTE PTR(TRACK_LIST)(SI+MASK_OFF),0
WORD PTR(TRACK_LIST)(SI+XBREQ_OFF),0
WORD PTR(TRACK_LIST)(SI+XBAGE_OFF),0
WORD PTR(TRACK_LIST)(SI+DATA_POINTER_OFF),AX
AX,288
SI,RECORD_LENGTH
TRACK_INIT_LP

```

SIDE ONE: TRK_NO:

```

NEAR
AH,(CURRENT_DRIVE) ASSUMES DS = 0
BX,(CURR_APPLN) TRACK 0
DI,0
DH,0
SCAN_TRACK_LIST
ES,AX
DS,AX
DI,DI
FIRST
SECOND
FIRST
SECOND
FIRST
SIXTH
SIXTH
SIXTH
SIXTH
SIXTH
SIXTH
AX,0000H
DS,AX
AH,(CURRENT_DRIVE)
BX,(CURR_APPLN) TRACK 0
DI,0
DW,?
SCAN_TRACK_LIST

```

TRACKO_INIT

```

PROC
MOV
MOV
MOV
MOV
CALL
MOV
MOV
MOV
MOV
CALL
XOR
CALL
CALL
CALL
CALL
CALL
CALL
CALL
MOV
MOV
MOV
MOV
MOV
CALL

```

MOV	AX, (TRACK_SEG)		
MOV	ES, AX		
MOV	DS, AX	SET INDEX TO 0	
XOR	DI, DI		
CALL	SIXTH		
CALL	SIXTH		
CALL	SIXTH		
RET			
PROC			
MOV	NEAR AL, 00H		
MOV	CX, 512		
REP	STOSB		
RET			
PROC			
MOV	NEAR AL, 0FDH	MEDIA DESCRIPTOR	
STOSB			
MOV	AL, 0FFH		
STOSB			
MOV	AL, 0FH		
MOV	CX, 509	REMAINDER OF SECTOR	
REP	STOSB		
RET			
PROC			
MOV	NEAR CX, 16	NO. OF DIRECTORIES PER SECTOR	
PUSH	CX		
CALL	DIREC		
POP	CX		
LOOP	S1		
RET			
PROC			
MOV	NEAR AL, 00H		
STOSB			
MOV	AL, 0F6H		
MOV	CX, 31		
REP	STOSB		
RET			
PROC			
MOV	NEAR SI, 0		
MOV	CX, 42		
CMP	BYTE PTR (TRACK_LIST)(SI+DRIVE_NO_OFF), AH		
JZ	APPL_NO		
JMP	NEXT_ENTRY		
CMP	WORD PTR (TRACK_LIST)(SI+APPL_NO_OFF), BX		
JZ	TRAK_NO		
JMP	NEXT_ENTRY		
CMP	BYTE PTR (TRACK_LIST)(SI+TRACK_NO_OFF), DL		
JZ	SID_NO		
JMP	NEXT_ENTRY		
CMP	BYTE PTR (TRACK_LIST)(SI+SIDE_NO_OFF), DH		

; ASSUMES DS = 0, AH = REQD DRIVE
 ; BX = REQD APPL NO, DL = REQD TRACK NO
 ; DH = REQD SIDE NO
 ; LOAD COUNTER WITH MAX NO OF SIDES
 ; PIR (TRACK_LIST)(SI+DRIVE_NO_OFF), AH
 ; PIR (TRACK_LIST)(SI+APPL_NO_OFF), BX
 ; PIR (TRACK_LIST)(SI+TRACK_NO_OFF), DL
 ; PIR (TRACK_LIST)(SI+SIDE_NO_OFF), DH

```

FOUND_TRACK:
NEXT_ENTRY:
JZ      FOUND_TRACK
JMP     NEXT_ENTRY
MOV     AX,WORD PTR(TRACK_LIST)(SI+DATA_POINTER_OFF)
MOV     [TRACK_SEG],AX
MOV     AH,0
        SET STATUS SUCCESSFUL
SI,RECORD_LENGTH
DRIVE
AH,OFFH
        SET STATUS UNSUCCESSFUL

```

```

JZ
JMP
MOV
MOV
MOV
RET
ADD
LOOP
MOV

```

APPENDIX P THE TRACK EMULATION SOFTWARE

This routine forms the core of the entire disk emulation process. As its name implies the routine controls the emulation of a track of data that would normally be read off a floppy disk. This routine also accepts data that would be written to the floppy disk. The track that has to be emulated is shown in Figure P1 and is described in detail in Appendix A.

The track emulation procedure is described below in terms of a high level language. Procedure names that are in capital letters relate to procedures that can be directly associated with characteristic features of the track of the disk as shown in Figure P1. Procedure names that are in small letters are procedures that are required due to implementation details. Bracketed procedures indicate procedures that have been split but conceptually should be one procedure. The assembler listing is at the end of this appendix.

It is assumed that the communications controller has been programmed previously. See Appendix O.

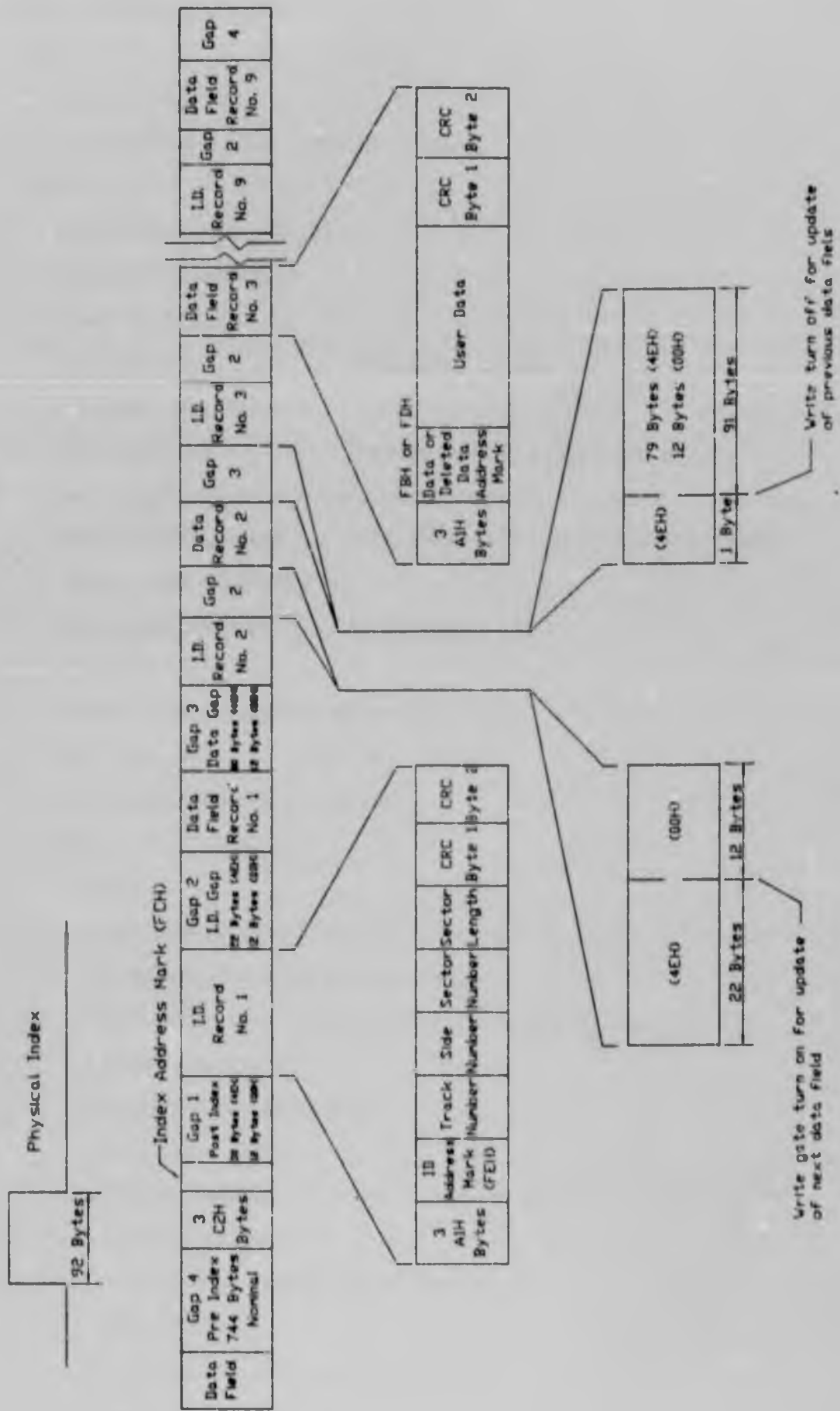


Figure P.1 Diagram Of The IBM System 34 Double Density Track Format


```

PROCEDURE Track_emulate;
VAR
    n,i : integer;
    track_number, prev_track_no : byte;
BEGIN
    set_data_source_pointer,
    INDEX_PULSE_ACTIVE;
    start_tx_clk;
    GAP (4EH,77);          {Remainder of gap 4 }
    set_synch_characters;
    GAP (00H,12);         {Synchronisation zeros of gap 4}
    enable_address_marker_generator;
    INDEX_HEADER_MARKER; {Three C2H bytes with missing clocks}
    INDEX_PULSE_INACTIVE;
    INDEX_ADDRESS_MARK;  {FCH byte}
    -GAP (4EH,1);
    |disable_address_marker_generator;
    ^GAP (4EH,49);        {Gap 1}
    n := no_of_sectors_per_track;
    FOR i:= 1 TO n DO
        BEGIN
            -GAP (00H,6);
            |enable_address_marker_generator;
            ^GAP (00H,6);  {Synchronisation zeros of gap 1}
            reset_eom_latch;
            reset_tx_crc_generator;
            START_CRC (A1H);
            HEADER_MARKER; {Three A1H bytes with missing clocks}
            enable_receiver;
            I D ADDRESS_MARKER; {FEH byte }
            SEND_TRACK_NO;
            disable_address_marker_generator;
        
```

```

SEND_SIDE_NO;
SEND_SECTOR_NO;      (=1)
SEND_SECTOR_SIZE;
SEND_CRC;
GAP (4EH,22);      {Gap 2}
rGAP (00H,6);
|enable_address_marker_generator;
L^GAP (00H,6);      {Synchronisation zeros for gap 2}
reset_eom_latch;
reset_tx_crc_generator;
START_CRC (A1H);
HEADER_MARKER;      {Three A1H bytes with missing clocks}
DATA_ADDRESS_MARKER;{FBH byte}
set_data_destination_pointer;
get_data_segment_pointer;
rGAP (dets[1],1);
|disable_address_marker_generator;
L^SEND_SECTOR (511); {Emulate actual sector data}
SEND_CRC;
all_characters_received;
rGAP (4EH,10);
|empty_receiver_buffer;
|disable_receiver;
L^GAP (4EH,40);      {Gap 3}
disable_crc_generator;
read (track_number);
if track_no <> prev_track_no then goto ABORT
read (head_number);
if head_number <> prev_head_no then goto ABORT
END;
GAP (4EH,100);      {Gap 4}
:ABORT BEGIN

```

```

        stop_tx_clk;
        return;
    END;
END;

```

Most of the above procedure names are self explanatory but three that should be explained here to complete the above description are GAP and START_CRC and SEND_SECTOR.

```

PROCEDURE GAP (          byt : byte,
                no_of_bytes : integer );

```

```

VAR
    i : integer;
BEGIN
    For i := 1 TO no_of_bytes DO
        BEGIN
            REPEAT
                UNTIL transmit_buffer_empty;
                transmit (byt)
            END;
        END;
    END;

```

The transmitter is polled until the transmitter buffer is empty. In the assembler implementation the byte to be transmitted is passed to this routine in the AL register and the no of bytes in the CX register.

```

PROCEDURE START_CRC (byt : byte);

```

```

BEGIN
    REPEAT
        UNTIL transmit_buffer_empty;
        enable_CRC_generator;
        transmit (byt)
    END;

```

The CRC generator, having been reset previously, is enabled just before the transmission of a specific character. As above the required character

```

        stop_tx_clk;
        return;
    END;
END;

```

Most of the above procedure names are self explanatory but three that should be explained here to complete the above description are GAP and START_CRC and SEND_SECTOR.

```

PROCEDURE GAP (      byt : byte,
                  no_of_bytes : integer );

```

```

VAR
    i : integer;
BEGIN
    For i := 1 TO no_of_bytes DO
        BEGIN
            REPEAT
                UNTIL transmit_buffer_empty;
                transmit (byt)
            END;
        END;
    END;

```

The transmitter is polled until the transmitter buffer is empty. In the assembler implementation the byte to be transmitted is passed to this routine in the AL register and the no of bytes in the CX register.

```

PROCEDURE START_CRC (byt : byte);

```

```

BEGIN
    REPEAT
        UNTIL transmit_buffer_empty;
        enable_CRC_generator;
        transmit (byt)
    END;

```

The CRC generator, having been reset previously, is enabled just before the transmission of a specific character. As above the required character

is passed to the procedure in the AL register. The transmit CRC enable bit should be in the desired state when the data character is loaded from the transmit data buffer into the transmit shift register. To ensure the proper state of this bit, the buffer state is monitored until it is empty i.e. the data would have been transferred to the shift register and only then is the CRC enabled and the data character loaded.

```
PROCEDURE SEND_SECTOR (no_of_bytes : word);
  VAR
    i,RxD_CHARS : integers;
    byt : byte;
  BEGIN
    RxD_CHARS := 0;
    For i := 1 TO no_of_bytes DO
      BEGIN
        IF char_ready AND (RxD_CHARS < 512)
          THEN BEGIN
            receive (byt);
            RxD_CHARS := RxD_CHARS + 1;
          END;
        IF transmit_buffer_empty THEN send (byt)
      END;
    END;
```

This procedure is responsible for transmitting the sector information during the emulation process while at the same time accepting data that is being written to the sector.

Most of the above mentioned procedures can be followed easily in the assembler listing at the end of this appendix. Only aspects of particular interest and importance will be discussed. The constants used will be self explanatory in most cases but all constants used are defined in Appendix O. The peculiarities of programming the SCC will also be described there.

An important aspect to remember throughout this routine is that all DATA written to or read from the SCC has its bit order reversed i.e. the reads and writes are made to the Z8530BDREV register. It must also be remembered that channel B of the SCC is used for the emulator but the DTR line of channel A is also used. The bit functions of the registers most frequently manipulated in this routine are shown in Figure P2.

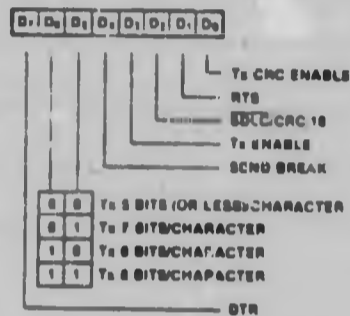


Figure P2 Bit Functions Of Write Register 5 Of The Z8530

set_data_source_pointer - As is explained in Appendix I on the track cache implementation the offset within the segment at which each data block of the cache starts is zero. The SI register which is the source index register must therefore be set to zero.

INDEX_PULSE_ACTIVE and INACTIVE - Setting and resetting of the DTR bit in register 5 of channel A sets the index pulse active and inactive respectively.

start_tx_clk - Setting RTS bit in register 5 of channel B enables the transmitter clock source.

set_synch_characters - The synchronization characters of the SCC are initially set to 4EH so that when emulation begins these characters will be transmitted. These characters are also used by the receiver of the SCC. The synchronisation characters are chosen to be the last A1H byte and the data marker of the sector i.e. FBH. In this way the first data character that will be assembled in the receiver is the first data byte of the sector being written to the disk emulator. These bytes have to be reversed before they are written to the SCC since they are not reversed by the hardware like the other data bits hence the DFH instead of FBH etc.

enable_address_marker_generator and **disable ...** - Setting and resetting the DTR bit in register 5 of channel B enables and disables the address marker generator.

reset_eom_latch - For the CRC to be appended to a block of data this latch has to be reset after the first data byte of the block has been loaded. When this latch is set again when no more data is supplied at the end of the message the CRC is automatically appended. This is done by issuing the reset EOM command to register 0 of channel B.

reset_tx_crc_generator - The CRC generator has to be preset before the calculation of the CRC on the forthcoming data stream can be performed. This is done by issuing the reset Tx CRC command to register 0 of channel B.

SEND_CRC - The SCC is forced to send the CRC calculated on the data by causing the transmitter to under-run i.e. stop providing the transmitter above. The EOM (End Of Message) latch must have been reset at some stage between the first and last character being loaded into the transmitter. The way that this is done here is by forcing the program into a loop after the last character of a block is sent. This loop is terminated by the EOM bit being set which indicates that the transmission of the CRC is in progress.

set_destination_pointer - The source pointer is constantly updated throughout the emulation of a track of data. The destination pointer is only updated during a disk write operation and since not all the sectors need be written to. The destination pointer is therefore updated at the start of each sector of data in anticipation of a disk write.

get_data_segment_pointer - As explained the track cache index stores the segment at which the data starts. The data segment register is loaded with this value.

all_characters_received - Characters are loaded from the SCC until the full sector being written to the emulator has been read in.

When the assembler listing is examined the gap sizes implemented in the program are actually smaller than those specified in the high level description. As was mentioned previously the gap sizes were reduced as much as possible to increase the efficiency of the emulation process. The only gap size that could not be reduced is that of gap 2 between the sector I.D. and the actual sector. When this gap was reduced data being written to the disk was lost at the beginning of the sector. This gap therefore is best left the correct length. The Floppy Disk Controller did not appear to be affected by the reductions in the other gap sizes.

MOV	AX,0000H	SET DATA SEGMENT TO VARIABLES
MOV	DS,AX	
MOV	CX,WORD PTR SECTS_PER_TRACK	
PUSH	CX	
MOV	AL,0	START OF SYNCHRONIZATION ZEROS
MOV	CX,6	FOR GAP 1 (00H)
CALL		
MOV	AL,REG5	SELECT REGISTER 5 OF CHANNEL B
OUT	DX,AL	ENABLE ADDRESS MARK GENERATOR
MOV	AL,B TX_EN	REST OF SYNCHRONIZATION ZEROS
OR	AL,DTR	
OUT	DX,AL	FOR GAP 1 (00H)
MOV	AL,0	AL,RESET_EDM_LATCH RESET THE EDM LATCH
MOV	CX,6	
CALL		RESET CRC GENERATOR
MOV	AL,RESET_EDM_LATCH	
OUT	DX,AL	ENABLE CRC GEN AND SEND 1 CHAR
MOV	AL,0A1H	OF 1.D., ADDRESS HEADER MARKER
CALL		REST OF 1.D. ADDRESS HEADER MARKER
MOV	AL,0A1H	SELECT REGISTER 3 OF CHANNEL B
MOV	CX,2	ENABLE RECEIVER
CALL		
MOV	AL,REG3	
OUT	DX,AL	1.D. ADDRESS MARKER
MOV	AL,B RX_EN	LOAD TRACK NUMBER
OUT	DX,AL	SEND TRACK NUMBER
MOV	AL,0FEH	SELECT RECISIER 5 OF CHANNEL B
MOV	CX,1	KEEP CRC GENERATOR ACTIVE
CALL		DISABLE ADDRESS MARK GENERATOR
MOV	AL,(TRACK_NO)	LOAD THE SIDE NUMBER
MOV	CX,1	SEND SIDE NUMBER
CALL		LOAD SECTOR NUMBER
MOV	AL,(SECTOR_NO)	SEND SECTOR NUMBER
MOV	CX,1	512 BYTES PER SECTOR
CALL		
MOV	AL,2	SEND SECTOR SIZE
MOV	CX,1	READ SCC STATUS
CALL		TEST FOR EDM
MOV	AL,DX	(CRC BEING SENT)
IN	AL,01100000B	GAP 2 (4EH) NOTE SIZE NOT REDUCED
TEST	JZ	START OF SYNCHRONIZATION ZEROS
MOV	AL,4EH	FOR GAP 2 (00H)
MOV	CX,22	
CALL		
MOV	AL,0	
MOV	CX,6	
CALL		

SECTOR:

EDM:

```

MOV AL, REG5
OUT DX, AL
MOV AL, B_TX_EN
OR AL, DIR
MOV DX, AL
MOV AL, 0
MOV CX, 6
CALL GAP
MOV AL, RESET_EOM_LATCH
OUT DX, AL
MOV AL, RESET_TX_CRC
OUT DX, AL
MOV AL, DATH
CALL START_CRC
MOV AL, DATH
MOV CX, 2
CALL GAP
MOV AL, LFBH
MOV CX, 1
CALL GAP
MOV DI, SI
INC BYTE PTR(SECTOR_NO)
MOV AX, TRACK_SEG
MOV DS, AX
MOV ES, AX
LDSB
MOV CX, 1
CALL GAP
MOV AL, REG5
OUT DX, AL
MOV AL, B_TX_EN
OR AL, TX_CRC_EN
OUT DX, AL
MOV CX, 511
CALL SEND_SECT
IN AL, DX
TEST AL, CHAR_READY
JZ EOM3
CMP BX, 512
JGE EOM3
MOV DX, Z8530BDREV
INSB
INC DX, Z8530BC
MOV AL, DX
IN AL, E_O_M
TEST EOM2
JZ AL, 4EH
MOV CX, 10
CALL GAP
IN AL, DX
MOV AL, 00000001B
CALL CONT
TEST DX, Z8530BD
JZ AL, DX
MOV IN
MOV

```

SELECT REGISTER 5 OF CHANNEL B
ENABLE ADDRESS MARK GENERATOR
AND DISABLE CRC GENERATOR
REST OF SYNCHRONIZATION ZEROS
FOR GAP 2 (DOH)
RESET EOM LATCH
RESET TX CRC GENERATOR
ENABLE CRC GEN AND SEND 1 CHAR
OF DATA HEADER MARKER
REST OF DATA HEADER MARKER
DATA ADDRESS MARKER
SET DATA DESTINATION POINTER
UPDATE SECTOR NUMBER
DATA SEGMENT TO TRACK DATA
LOAD DATA BYTE
SEND FIRST DATA BYTE
SELECT REGISTER 5 OF CHANNEL B
DISABLE ADDRESS MARK GENERATOR
SEND SECTOR
READ SGC STATUS
ALL CHARS RECEIVED
(SEND CRC)
START OF GAP 3 (4EH)
EMPTY OUT RECEIVE BUFFER

```

EOM2:
EOM3:
PX_IP
MOV

```

	JMP	RX_LP	SELECT REGISTER 3 OF CHANNEL B
	MOV	AL,AL	DISABLE RECEIVER
	OUT	AL,0	REST OF GAP 3 (4EH)
	MOV	DX,AL	
	OUT	AL,4EH	
	MOV	CX,40	
	CALL	GAP	
	MOV	AL,REG5	SELECT REGISTER 5 OF CHANNEL B
	OUT	DX,AL	DISABLE CRC GEN
	MOV	AL,B_TX_EN	
	OUT	DX,AL	
	POP	CX	
	MOV	AX,0000H	SET TO TO VARIABLES SEGMENT
	MOV	DS,AX	SAVE PORT POINTER TO TRACK COUNTER. PORT
	FUSH	DX	SET PORT POINTER TO TRACK COUNTER
	MOV	DX,IMPORT2	READ IN ACK COUNTER
	IN	AL,DX	RESTORE PORT POINTER
	POP	DX	AL BYTE PTR(TRACK_NO) COMPARE WITH PREVIOUS VALUE
	CHP	AL,BYTE PTR(TRACK_NO)	GOTO ABORT IF NOT EQUAL
	JNZ	ABORT	SET TO SIDE ZERO
	MOV	AH,0	READ SCC STATUS
	IN	AL,DX	CHECK CTS (SIDE)
	TEST	AL,00100000B	
	JZ	SIDO	SET TO SIDE 1
SIDO:	MOV	AH,1	AL BYTE PTR(SIDE_NO) COMPARE WITH PREVIOUS VALUE
	CHP	AH,BYTE PTR(SIDE_NO)	GOTO ABORT IF NOT EQUAL
	JNZ	ABORT	
ABORT	LOOP	SECT	SELECT REGISTER 5 OF CHANNEL B
	MOV	AL,REG5	
	OUT	DX,AL	DISABLE TXCLK
	MOV	AL,B_TX_EN	
	AND	AL,RTSOFF	
	OUT	DX,AL	
	RET		
SECT:	JMP	SECTOR	
GAP	PROC	NEAR	CHAR IN AL, COUNT IN CX
GAP_LP:	MOV	BI,AL	STORE CHARACTER
	IN	AL,DX	READ SCC STATUS
	TEST	AL,00000100B	TRANSMIT BUFFER EMPTY?
	JZ	GAP_LP	JUMP IF NOT
	MOV	DX,2B530BDREV	POINT TO REVERSED DATA PORT
	MOV	AL,BI	GET CHARACTER BACK
	OUT	DX,AL	SEND CHARACTER
	MOV	DX,2B5308C	RESTORE PORT POINTER
	LOOP	GAP_LP	
	RFT		
START_CRC	PROC	NEAR	STORE CHARACTER
STORCLP:	MOV	BI,AL	READ SCC STATUS
	IN	AL,DX	TRANSMIT BUFFER EMPTY?
	TEST	AL,00000100B	
	JZ	STORCLP	
	MOV	AL,REG5	
	OUT	DX,AL	SELECT REGISTER 5 OF CHANNEL B
	MOV	AL,B_TX_EN	

```

OR
OR
OUT
MOV
MOV
OUT
MOV
RET

SEND_SECT
RX_CHAR

SEND_CHAR

PROC
MOV
IN
TEST
JZ
CMP
JGE
MOV
:NSB
INC
MOV
IH
TEST
JZ
MOV
OUTSB
MOV
LOOP

AL,DIR
AL,IX_CRC_EN
DX,AL
DX,Z8530BDREV
AL,RL
DX,AL
DX,Z8530BC

KEEP DIR ACTIVE
ENABLE CRC GENERATOR
POINT TO REVERSED DATA PORT
GET CHARACTER BACK
SEND CHARACTER
RESTORE PORT POINTER

NEAR
BX,0
AL,DX
AL,0000001B
SEND_CHAR
BX,512
SEND_CHAR
DX,Z8530BDREV

SET CHAR READ COUNTER TO 0
READ SCC STATUS
CHARACTER READY ?
JUMP IF NOT READY
ALL CHARS RECEIVED ?
JUMP IF ALL RECEIVED
POINT TO REVERSED DATA PORT
READ CHARACTER INTO AL
INCREMENT RECEIVED CHARACTER COUNTER
POINT TO CHANNEL B CONTROL REGISTER
READ SCC STATUS
TRANSMIT BUFFER EMPTY
JUMP IF BUFFER NOT EMPTY
POINT TO REVERSED DATA PORT
SEND BYTE
POINT TO CHANNEL B CONTROL REGISTER

BX
DX,Z8530BC
AL,DX
AL,00000100B
RX_CHAR
DX,Z8530BDREV
DX,Z8530BC
RX_CHAR

```

APPENDIX Q TESTING THE FLOPPY DISK DRIVE EMULATOR

Testing a unit such as the floppy disk drive emulator requires a modular testing approach so that the number of variables in each test is minimised.

The unit can be divided into three modules for testing - the MFM decoder, the microprocessor control unit and the MFM encoder. A brief explanation of the testing of each of these units will be given.

Olivetti M24 Personal Computers provide the facility for the user to insert a Zilog 28530 Synchronous Communications Controller (SCC) on the motherboard. Only one channel of this dual channel device is hardwired to the RS 232 serial port with the other free for use. Using two P.C.'s, one of which should be an Olivetti to provide the above mentioned feature, the MFM decoder can be tested.

The MFM decoder has two inputs viz. the Write Enable and Write Data inputs that require connection to the floppy disk drive interface and two outputs providing a synchronous clock and data output. The decoder inputs are connected directly to the write enable and write data lines on the floppy disk drive interface. The outputs are connected to the RxD (Receive data) and one of the clock inputs on the SCC. A short assembler program was written for the Olivetti P.C. which initialises the SCC and then enters a polling routine to read in characters and display them on the screen. To ensure that no characters are missed, the characters are written directly to the screen memory. The synchronisation characters used are 4EH characters. Note the synchronisation characters must be written with their bit order reversed. The chip initialisation is the same as described in Appendix O. Using these synchronisation characters the SCC should begin assembling characters, two characters after the write enable signal becomes active. A disk is then formatted in the normal way on the P.C. to which the inputs of the decoder have been connected. The program on the

Olivetti P.C. should be started first so that the SCC is initialised and 'hunts' for synchronisation characters. When the format program is started the SCC will synchronise on the characters in gap 4 preceding the index address marker. See Figure P1. The characters being written to the disk being formatted are written to the screen of the Olivetti BUT their bit order is reversed i.e. instead of N's appearing (ASCII character corresponding to 4EH) we find r's (ASCII character corresponding to 72H). Beside testing the operation of the decoder, the format information written to the disk during formatting could be verified. The CRC characters were also of great use as will be described later.

The synchronisation characters were then changed to A1H and FBH respectively. These two characters are the characters immediately preceding the data written in a sector. (Again these two characters must be written with their bit order reversed). Using the same program as above on the Olivetti P.C. with the minor change in the synchronisation characters, PCTOOLS was run on the other P.C.. Again the Olivetti's program is started first and then individual sectors were written to the floppy disk drives. (Note in this test and the one described above a normal floppy disk is being written to. Which disk drive is used is of no importance since the Write Enable and Write Data signals connected to the decoder are common to all the floppy disk drives). This test was done to see if the choice of synchronisation characters would be suitable for the SCC receiver on the emulator. This choice of synchronisation characters requires no intervention from the CPU to separate the data. The data appearing on the Olivetti screen was compared with that being written using PCTOOLS. Using synchronisation characters of A1H and FBH the I.D. records of each sector could be checked during the formatting process. This proved to be very useful in the CRC checks as is explained later.

These tests verified the operation of the MFM decoder hardware as well as verifying some aspects of the software for the emulator. By using this

Olivetti P.C. should be started first so that the SCC is initialised and 'hunts' for synchronisation characters. When the format program is started the SCC will synchronise on the characters in gap 4 preceding the index address marker. See Figure P1. The characters being written to the disk being formatted are written to the screen of the Olivetti BUT their bit order is reversed i.e. instead of N's appearing (ASCII character corresponding to 4EH) we find r's (ASCII character corresponding to 72H). Beside testing the operation of the decoder, the format information written to the disk during formatting could be verified. The CRC characters were also of great use as will be described later.

The synchronisation characters were then changed to A1H and FBH respectively. These two characters are the characters immediately preceding the data written in a sector. (Again these two characters must be written with their bit order reversed). Using the same program as above on the Olivetti P.C. with the minor change in the synchronisation characters, PCTOOLS was run on the other P.C.. Again the Olivetti's program is started first and then individual sectors were written to the floppy disk drives. (Note in this test and the one described above a normal floppy disk is being written to. Which disk drive is used is of no importance since the Write Enable and Write Data signals connected to the decoder are common to all the floppy disk drives). This test was done to see if the choice of synchronisation characters would be suitable for the SCC receiver on the emulator. This choice of synchronisation characters requires no intervention from the CPU to separate the data. The data appearing on the Olivetti screen was compared with that being written using PCTOOLS. Using synchronisation characters of A1H and FBH the L.D. records of each sector could be checked during the formatting process. This proved to be very useful in the CRC checks as is explained later.

These tests verified the operation of the MFM decoder hardware as well as verifying some aspects of the software for the emulator. By using this

test configuration the area where hardware problems were most likely was restricted to the decoder.

Testing of the microprocessor control unit was done by testing sub-units within the control unit. The first sub-unit tested were the diagnostic L.E.D.'s since they couldn't fulfil their purpose until their functioning had been verified. The memory test described in Appendix O could then be performed. The input port that is used for the track counter was tested by applying various inputs to this port which was then output on the L.E.D.'s. The Olivetti P.C. was used in the testing of the control unit as well. Each of the SCC's on the control unit were tested by sending messages from the control unit to the P.C.. The P.C. was using the same program as described above.

At this stage the MFM decoder could be combined with the control unit. The one SCC's receiver was thus connected to the MFM decoder in the same way as it had been connected to the SCC in the Olivetti previously. The second SCC was connected up as a transmitter to the Olivetti P.C.. The software required for this test was a combination of the software used up to this stage. The program written for the Olivetti for the first formatting test was implemented on the control unit along with the program to transmit data to the Olivetti P.C.. This software was combined in such a way that a block of data was read into the control unit which was then transmitted to the Olivetti. The connections to the inputs of the MFM decoder were the same as in the original tests. The tests that had been previously performed on the MFM decoder were repeated. In this case though the data appearing on the Olivetti screen is not bit reversed since the bit order is restored by the control unit hardware. The original results obtained with the MFM decoder and Olivetti alone were now verified with the control unit included.

This confirmed the operation of the MFM decoder with the control unit.

Testing of the MFM encoder required the use of a logic analyser as well as the Olivetti P.C.. The SCC on the Olivetti provided the inputs to the MFM encoder. The data inputs to the MFM encoder as well as its output were monitored on the logic analyser. The encoding of all the bit sequences that when encoded produce all the possible MFM coding combinations were tested. These combinations are described in Appendix F. Sequences of zeros followed by the characters in the Index and I.D. address markers were used to verify the operation of the address mark generator. This required some control to be included from the software level. Recall that the address mark generator is controlled by the DTR line of the SCC. The waveforms recorded with the logic analyser corresponded with those of data being read off a typical floppy disk as required.

Having verified the encoding of the data only the CRC generation had to be verified. This proved to be rather difficult and time consuming. The main reason for this being that the manufacturers data either doesn't specify which CRC polynomial is used or what starting conditions are used or exactly what data it is calculated on. Some of the data that is available is in fact incorrect. In an Intel application note entitled 'An Intelligent Data Base System using the 8272' [9] the following statement is made in their discussion on the sector I.D. field. "The first byte of the field is the I.D. address mark. The second, third and fourth bytes are the cylinder, head and sector addresses, respectively, and the fifth byte is the sector length code. The CRC character is derived by the controller from the data in the first five bytes". As it turns out this is not the case. The Western Digital corporation have the FD1791/2 Floppy Disk Formatter / Controller that can be programmed to operate as a controller for the IBM System 34 Double Density formatting[10]. This format it should be recalled is the format used on the P.C.. The data sheet for the FD1791/2 states that the polynomial used is $x^{16} + x^{12} + x^5 + 1$. It also states that the CRC register is preset to ones before the data is shifted through it. This information was found to be correct. It also

states that the CRC includes all information starting with the address mark up to the CRC characters. In a figure in the data sheet the I.D. address mark is shown to be the FEH byte. In the data listed to format disks for the IBM format the data byte written to produce each of the AIH bytes required (which have missing clock pulses) states that the AIH byte is written in MFM and the CRC is preset. Comparing this information then with that from Intel it would appear that the CRC calculation commenced with the I.D. address mark in the case of sector I.D.'s and the data address mark in the case of sectors of data.

Armed with this information the SCC in the Olivetti was initialised to use the above mentioned CRC polynomial and initial conditions. The data in some of the sector I.D.'s that were read during the testing of the MFM decoder were sent through the SCC and the CRC generated. (Note the SCC was in loop-back mode so that the resulting message and CRC could be examined on the screen). The CRC characters generated did not agree with those read using the MFM decoder. A number of these sector I.D.'s were tested and none produced the required results. Finding patterns in the results obtained to try and determine the reason for the discrepancy is virtually impossible.

Some of the CRC's generated were verified by hand calculation. The results of the hand calculations corresponded with those of the SCC.

The first attempt to rectify the problem was to reverse the bits of each of the data bytes before calculating the CRC. This unfortunately just generated a third set of values. Working on the assumption that the polynomial and the initial conditions were correct, the only reason for an incorrect CRC is that the data on which it was calculated was incorrect. This assumption was made since the CRC polynomial used in the 8271 Single Density floppy disk controller [11] was the same and likewise the 82062 Winchester Disk controller [12] used this polynomial. An application note entitled 'Hard Disk Controller Design using the Intel 8089' [13]

states that this polynomial is used there as well. The polynomial therefore seemed a popular choice for this type of application.

The data passed through the generator was varied by adding and leaving out bytes, reversing the data etc. The solution was eventually found to be that the 3 A1H bytes preceding the sector I.D. address marker had to be included in the CRC calculation. The data must also be presented to the generator most significant bit first. The 3 A1H bytes preceding the data address marker were included in the CRC calculation for the data sector. Fortunately the resulting CRC corresponded with that of the floppy disk controller.

The final sub-unit that required testing was the track counter. This was done by connecting the STEP and DIR lines to the floppy disk interface. The control unit was programmed to read the track counter input port and put the value read out on the diagnostic L.E.D.'s. When a disk is formatted using MS-DOS 2.11's format program the track numbers are displayed on the screen as formatting takes place. The values on the L.E.D.'s were then compared with those on the screen.

Software based on the format information available was written to emulate a track of data i.e. the TRACK_EM routine with full size gaps. See Appendix P. The MEM decoder, encoder and controller unit were then interfaced to the floppy disk drive cable via the interface unit. The floppy disk drive emulator was then connected in place of one of the floppy disk drive units. Access to the emulator from DOS returned a 'NON DOS DISK ERROR'. This error indicated favourable performance by the emulator. The floppy disk controller was thus able to find the sector that it was looking for. When the sector was read and DOS realised that it did not correspond with the required DOS format the error was given.

Track 0 of a floppy disk contains various pieces of information for the operating system. This includes the bootstrap, file allocation table and the disk directory. The sectors assigned to these pieces of information are the following:-

Sector 1 - Boot record

Sectors 2 and 3 - First copy of the file allocation table

Sectors 4 and 5 - Second copy of the file allocation table

Sectors 6 to 12 - Directory entries

The sectors that the operating system reads when a directory of a disk is requested were ascertained by intercepting a BIOS interrupt call. The INT 13H routine in the BIOS is used for accessing the floppy disk controller from the higher levels of software. A short piece of assembler patch code was written that redirected this BIOS interrupt routine to a short routine that prints the parameters passed to this routine on the screen before proceeding with the rest of the interrupt routine. In this way the sectors required by the floppy disk controller can be seen. From this information the sectors required to provide a directory of a disk for example were determined. All these sectors lie on the same track viz. track 0. The information in these sectors was read from a physical floppy disk that had just been formatted. This information is inserted in track 0 in procedure TRACK0_INIT shown in the assembler listing at the end of this appendix.

The TRACK0_INIT procedure calls procedures FIRST, SECOND and SIXTH to generate the data required. FIRST fills the first and third sectors and fifth sectors with zeros. SECOND fills the second and fourth sectors which contain the File Allocation Table. SIXTH generates the data for each of the directory sectors.

When a directory was now requested the 'Files not found' message was returned by DOS which had immense significance. This indicates that the floppy disk controller has been able to extract data from the information

being generated by the floppy disk drive emulator. To test the ability of the emulator to store data written to it various write operations that require the use of track 0 alone were performed. This includes making directories (MKDIR) and removing directories (RMDIR). These operations require that the file allocation tables as well as the directory sectors are updated. Tests using PCTOOLS were also performed where a sector is read in from the emulator and the first and/or last characters say of the sector were changed. The sector is then written to the emulator and then re-read to ensure that the entire sector was being written correctly.

These test alone proved that the majority of the hardware was functioning as required and the software used to control it also appeared satisfactory.

The final test that was performed was the following :- The track cache was implemented in such a way that its contents were fixed i.e. it contained the first 21 tracks (i.e. 42 sides) of a floppy disk. Track 0 was again initialised in a similar way to that described above. It was now possible to emulate a floppy disk with approximately 180K of storage capacity. Now it is possible to copy large files onto the emulator and read them off again.

At this stage various experiments were performed in which the gap sizes were reduced. Again PCTOOLS was used to modify the beginning and end characters of the individual sectors. When the size of gap 2 is changed characters are lost in the sector being written. This gap was therefore left unchanged. The other gap sizes were reduced with no adverse effect.

A test was performed to observe the effects of simulated network operation on the disk drive emulation process. It should be noted that in the original emulator design had only one SCC. These devices have two channels

and it was originally designed for one channel to be used by the emulator while the other was for the network.

the network channel was initialised to accept an external transmit clock source and to transmit data in synchronous data format. The data that was to be transmitted was accessed from a single memory location using DMA. The external clock source was provided from a variable frequency oscillator. Files were read and written to the emulator as described above while at the same time the transmit clock frequency was increased. The results of this test were hardly encouraging. The required transmit clock frequency is between 1 and 1.5MHz but the emulation process was failing at frequencies as low as 20kHz!. Occasionally the emulation process was failing at lower frequencies. Closer examination of the problem with a logic analyser led to these conclusions :- The emulator software was based on polling while the network data was being supplied using DMA. DMA cycles can occur at anytime when the bus has not been locked by the central processing unit. It was possible therefore for a read or write to be made by the polling software on the SCC which would be followed or preceded immediately by a DMA access on the same device. The SCC has a certain access recovery time i.e. a minimum period is required between the accesses to the SCC. In the situation described where two accesses were made on the SCC one after another this access recovery time was not met. In a number of cases this corresponded with the failure of the emulation process. For this reason a second SCC was added for exclusive use by the network. When the tests were performed now the network transmit clock frequency could be increased to within the required range without any adverse effects on the emulation process.

The floppy disk drive emulator thus performed as required with file access times slightly shorter or the same as that from a physical floppy disk.

LIST OF REFERENCES

1. Haugdahl, J. Scott 'Local-area networks for the IBM PC' *BYTE* Vol. 9 No. 13 Dec. 1984 pp. 147 - 174
2. Abraham, R and Munro, R 'Microflopies battle for pre-eminence' *COMPUTER DESIGN* Vol. 22 No. 1 Jan. 1983 pp.119 - 126
3. INTEL '8272A Single/double density floppy disk controller' *MICROSYSTEMS COMPONENTS HANDBOOK* Volume 2 1984 Order No.230843 pp. 6-553 - 6-571
4. INTEL 'iAPX 86/88, 186/188 User's Manual Hardware Reference' 1985 Order No. 210912-001 pp. 2-1 - 2-82
5. Chaney, R and Johnson, B 'Maximising hard-disk performance' *BYTE* Vol. 9 No. 5 May 1984 pp. 307
6. McKeon, Brian 'An algorithm for disk caching with limited memory' *BYTE* Vol. 10 No. 9 Sept. 1985 pp. 129 - 138
7. 'IBM PC Technical Reference Manual' First Edition (Revised March 1983) Part No. 6936844
8. 'I.B.M. Disk Operating System ' First Edition (Revised May 1983) Version 2.00 Part No. 6183485
9. INTEL '82530 SCC Technical Manual' July 1984 Order No. 230925-001
10. ZILOG 'Z8530 Data Sheet' 1983/84 Data Book Order No.00-2034-03 pp. 409 - 429

11. INTEL AP-116 'An intelligent data base system using the 8272' *MICRO-SYSTEM COMPONENTS HANDBOOK* Volume 2 1984 Order No.230843 pp.6-421 - 6-454
12. WESTERN DIGITAL CORPORATION 'FD1791/2 Floppy Disk Formatter/Controller' Document No. #A0103
13. INTEL '8271/8271-6 Programmable floppy disk controller' *MICROSYSTEMS CONTROLLER HANDBOOK* Volume 2 1984 Order No. 230843 pp. 6-541
14. INTEL '82062 Winchester disk controller' *MICROSYSTEMS CONTROLLER HANDBOOK* Volume 2 1984 Order No.230843 pp. 6-574
15. INTEL AP-122 'Hard disk controller design using the 8089' *MICROSYSTEMS COMPONENTS HANDBOOK* Volume 2 1984 Order No.230843 pp. 3-89

BIBLIOGRAPHY

- Sidhu, G.S. and Andrews, R.F. 'Applebus PROTOCOL ARCHITECTURE' *Apple Computer Inc.* May 1 1984
- Worden, J 'Design Considerations For Dual-Density Diskette Controllers' *Computer Design* Vol. 17 No. 6 June 1978
- Bass, C. 'Local Area Networks - A Merger Of Computer And Communications Technologies' *Microprocessors and Microsystems* Vol.5 No.5 June 1981
- Hurwicz, M. 'MS-DOS 3.1 Makes It Easy To Use IBM PCs On A Network' *Data Communications* Vol. 14 No. 12 Nov. 1985
- Mier, E. 'The Evolution Of A Standard Ethernet' *BYTE* Vol. 9 No. 13 Dec. 1984
- Sahr, R 'Two Low-Speed Nets Race To Link Computers' *ElectronicsWeek* Vol. 57 No. 36 Dec. 17 1984
- Loyer, B.A. 'LANs: The Coming Revolution In Factory Control' *Machine Design* Vol. 56 No. 27 Nov. 22 1984
- Hoepfner, J.F. and Wall, L.H. 'Encoding / Decoding Techniques Double Floppy Disk Capacity' *Computer Design* Vol. 19 No. 2 Feb. 1980
- March, D. 'Floppy Disks' *Electronics & Wireless World* Vol. 91 No. 1587 Jan 1985

- Roskos, J.E. 'Writing Device Drivers For MS-DOS 2.0' *BYTE* Vol. 9 No. 2 Feb. 1984
- Linge, N. 'Emerging Local Area Network Technologies' *Microprocessors and Microsystems* Vol. 10 No. 1 Jan/Feb 1986
- Glinert-Cole, S. 'A Network For All Reasons' *PC TECH JOURNAL* Vol. 2 No. 12 Dec. 1984
- Schwaderer, W.D. 'CRC Calculations' *PC TECH JOURNAL* Vol. 3 No. 4 April 1985



Author De Souza E A

Name of thesis Edulan: a Local area network for an educational environment 1987

PUBLISHER:

University of the Witwatersrand, Johannesburg

©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.