N74-10142

TECHNICAL
RESEARCH
REPORT

User Microprogrammable Processors for High Data Rate Telemetry Preprocessing

James H. Pugsley
E. P. O' Grady

DEPARTMENT OF
ELECTRICAL ENGINEERING

UNIVERSITY OF MARYLAND

COLLEGE PARK, MARYLAND 20742

DEPARTMENT OF ELECTRICAL ENGINEERING UNIV. OF MARYLAND

July 1973

User Microprogrammable Processors for High Data Rate Telemetry Preprocessing

Final report prepared under Grant NGR 21-002-355 from the National Aeronautics and Space Administration

James H. Pugsley
E. P. O'Grady

The task of this project was to investigate the use of microprogrammable processors for the preprocessing of high data rate satellite telemetry. The major conclusions of the project are summarized below.

I) Existing commercially available user microprogrammable minicomputers were surveyed and an assessment made of their usefulness for such high data rate telemetry preprocessing. These processors offer a significant improvement in performance over non-microprogrammable processors, but come nowhere close to meeting the target data rate of 30 megabits per second. Writable control storage (i.e., the ability of the user to change the microprograms in control memory) increases the speed of telemetry preprocessing programs by a factor of from 5 to 15 over that for ordinary microprogrammable processors, since the algorithms needed are in general simple enough for the entire program to be executed from control memory without the need for main memory access.

II) Considerably higher data rates would be possible if the minicomputers used had a few special microinstructions available that were tailored to the bit manipulations needed in telemetry preprocessing. Examples of such microinstructions are end for end reversal of the bits in a word and various bitwise logical and shift operations, possibly controlled by a mask register.

III) Even with the special microinstructions indicated above the current execution speeds (c. 200 nsec per microinstruction) of computers do not allow preprocessing at 30 megabit data rates. To achieve such rates will require the use of multiple-computer configurations or special purpose computers. The use of large scale computers rather than minicomputers makes very little difference in the data rates that can be achieved since the large computers are superior to the minicomputers mostly in the areas of arithmetic accuracy and speed of arithmetic operations, neither of which constitute significant portions of the telemetry preprocessing problem. An investigation was begun of the problems in using a network of several minicomputers to achieve the target data rates. With a special purpose multiplexer on the front end and a matching demultiplexer at the outputs a parallel connection of minicomputers can theoretically achieve any desired data rate. In practice problems of control and system reliability appear to dictate a structure other than a simple parallel connection. It appears that a minicomputer network could in practice handle data rates of up to possibly 100 megabits per second, but this is a tentative conclusion and further studies of system configuration, control, and communications problems are needed.

Each of the areas mentioned above is discussed in the following pages, along with supporting studies made in the course of the project. Significant among the latter was a study of the use of simulation techniques in the design and evaluation of minicomputer systems. The results of this study, which compared FORTRAN, APL, and a special purpose simulation language CDL (1), indicate that for the class of systems under consideration for telemetry preprocessing APL is the best choice for the simulations. A brief report on this study is presented in Appendix A.

I. Evaluation of commercial microprogrammable minicomputers for telemetry preprocessing tasks.

To get some quantitative answers on what data rates were achievable certain of the data formats from OAO-A2 were selected as providing a representative mix of the formats likely to be encountered in high data rate observatóries. In particular we considered the DD (direct digital), SD (status data), CM (command memory), and ED (experimenter data) formats. Our assumption is that while the absolute data rates will change, these data formats will remain representative and that the relative data rates of these various data formats on OAO-A2 will still be correct for forthcoming high data rate satellites.

An investigation of commercial minicomputers indicated that the microinstruction sets of the available computers are similar enough so that there was little difference from one computer to another in the complexity of the programs for telemetry preprocessing tasks, as long as the computer had general purpose registers. Detailed consideration was given to the Microdata 800, the Hewlett Packard 2100, the Burroughs D-Machine, and the Standard Computer MLP-900. In addition more limited investigation was made (limited by the data available) of the Varian 73 and the Interdata 80. Detailed comparisons of three of the computers studied are given in Appendix B.

The computers surveyed were also very similar in the time required to execute a single microinstruction. In view of these facts all examples actually programmed were for the Microdata 800, for which software documentation was readily available to us.

Accordingly we investigated some of the algorithms for preprocessing of these telemetry formats and the implementation of these algorithms on available minicomputers. Many of the formats require primarily the separating of the incoming data stream words (which differ in length) into uniform length computer words. For example the ED format contains a mixture of 8-bit, 12-bit, and 25-bit words. Such separation into computer words is readily performed by shift operations in the computers investigated, and is not the most severe constraint on the data rate attainable. More severe constraints are imposed by formats such as the OAO-A2 DD format, and for this reason the DD format was chosen as typical of the high data rate pre-processing problems. While other formats were programmed during the investigation, the remainder of this report will use the DD format as an example.

In this format the telemetry data words are eight bits long, identical to the 8-bit word length of the Microdata 800. The data words represent light intensity values from the SAO (Smithsonian Astrophysical Observatory) experiment aboard the spacecraft. These values are packed 251 per telemetry frame, and 256 such frames make up a picture with a television-like scan. The major preprocessing needed on each data word in this format is to reverse the word end for end, so that the most significant bit becomes the least significant bit, and vice versa, and to complement the second, fourth, and sixth bits of the resulting word. (Clearly the complementation can be done before, after, or during the bit for bit reversal.) The optimum microprogram to perform these operations on a MICRODATA 800 requires 20 microinstructions including I/O. The microinstruction execution time for this machine is 220 nsec per micro-instruction, but a figure of 200 nsec will be used in timing computations as more representative of the state of the art in commercially available minicomputers.+ This gives a time of 4.0 μsec required to process each eight bit data word, corresponding to a data rate of 2 megabits per second.

+ The range of microinstruction execution times for the computers surveyed was 196-220 nsec.

The availability of writable control memory in the computer is critical to the data rate achievable. If the computer has to fetch instructions from main memory each instruction fetch will require on the order of 1 μsec and the time required to process each data word will be increased by a factor of about six. The program actually written for the OAO-A2 DD format on the MICRODATA 800 ran entirely in control storage and made no use whatever of the main memory of the computer. This type of operation is essential to the real time preprocessing of data at rates in the megabits per second range, and is achievable in practice because the telemetry preprocessing algorithms are rather simple in nature. The typical control store for a user microprogrammable minicomputer has from 256 to 2048 words (microinstructions) of control memory. Certainly at the upper end of this range enough control storage is available to contain microroutines for each of the formats for most satellites. In this sort of operation the main memory would be used chiefly for swapping microprograms when shifting from one satellite or format to another.

The relationship between data word length and computer word length has an obvious effect on the maximum processing rate, the worst situation being when the data words are one bit longer than the computer words. The microinstruction sets of the computers surveyed do not seem to be appreciably better or worse than the instruction set of a typical non-microprogrammed computer at handling this sort of problem. The availability of the "special" microinstructions discussed in section II of this report would be helpful in those cases where the data and computer word lengths are badly mismatched.

II. Microinstruction sets for telemetry preprocessing.

In writing microroutines to perform the desired format translations on the various OAO-A2 formats some processing steps or subalgorithms came up repeatedly, and among these were a few that were relatively time consuming. The designers of available minicomputers did not have in mind the type of bit manipulations needed in telemetry preprocessing when determining the microinstruction repertoire of the computer. Consequently the implementation (in hardware) of a very few "special" microinstructions in a minicomputer that was otherwise quite standard would significantly increase the data rate capabilities of the computer. Recommendations for such special microinstructions are discussed in this section. A table of existing microinstructions in commercial minicomputers is given in Appendix B.

One of the chief culprits in consuming time for format translation is the necessity in many formats for reversing the order of the bits in each data word. Such an operation is awkward on almost every computer, involving successive shift, test, and load instructions. We have yet to find a computer, whether microprogrammed or not, whose instruction set makes this transformation quick or simple. For example, of the 20 microinstructions needed for the OAO-A2 DD data word format translation, 15 are due to the need to reverse the bits in the word. The implementation in hardware of a single microinstruction to perform this operation would significantly reduce the time required and consequently raise the maximum data rate that can be accomodated in real time. If a single 200 nsec reversal instruction were available the time to process a DD format data word would drop to 6 x 200 nsec = 1.2 µsec, corresponding to a telemetry data rate of 6.67 megabits per second.

Next in order of importance are operations such as selectively complementing certain bits in each data word. In some existing minicomputers this requires a micro-instruction execution for each bit that must be complemented and a number of shift operations. In other microcomputers the microinstruction repertoire is rich enough to allow a single instruction to complement the desired bits. For this reason care should be taken in the selection of a minicomputer for telemetry preprocessing, to insure the existence of at least some general purpose registers in which to store the pattern of bits to be complemented, and the availability of logical operations such as a register to accumulator bitwise exclusive-OR operation in the micro-instruction set. Setting 1's in the mask register in the positions where the bits are to be complemented and then exclusive-Or'ing the mask register with the data word will produce the desired result in the accumulator. The MICRODATA 800 has such an instruction and the resulting microprogram to perform the desired format translation on each OAO-A2 DD format intensity value word has the form:

```
     START         INPUT TO AC
                   REVERSE WORD
                   EXOR MASK TO AC
                   OUTPUT FROM AC
                   INCREMENT COUNT AND SKIP IF 0
                   JUMP TO START.
```

Of course the MICRODATA does not have a single instruction to perform the REVERSE WORD operation. The register used to count the 251 data words in a frame must be set to -250 at the start of each frame, and other segments of microcode are needed to do this and to take care of the processing needed on the 32-bit frame sync pattern and the single 8-bit line number word that appear in each frame. The processing steps that must be performed on the frame sync and line number bits are less complex than those for the data. A typical format (DD) has four bytes of FSP to 252 bytes of data, so the influence of the FSP processing time will certainly not be major. It seems safe to assume that the FSP and bookkeeping portions of the preprocessing programs will not cause appreciable reductions in the maximum data rates attainable.

The gain in speed available from user microprogrammed computers is due to the fact that the programs are in faster memory. There are very few arithmetic operations required for telemetry preprocessing algorithms. Consequently the set of rather elementary operations available at the micro-instruction level is as well suited for the implementation of these algorithms as are the machine language instruction sets of conventional computers. As computer main memories get faster user microprogramming may lose some of its advantage, but at present it appears to offer the most promising path to high data rate telemetry preprocessing.

If some specifications can be imposed on the micro-instruction repertoire, it would be desirable to include some powerful shift operations, which could cut processing times for some formats. The available minicomputers have only single bit shift and single bit rotate instructions, whereas many larger computers have single-instruction shift-by-n operations available. It is certainly within the current state of the art to include shift and rotate by n microinstructions within the microinstruction repertoire of a minicomputer, with n being the value in one of the general registers. The obvious choice is to implement in hardware the set of microinstructions

SHIFT AC RIGHT BY (R)
SHIFT AC LEFT BY (R)
ROTATE AC RIGHT BY (R)
ROTATE AC LEFT BY (R)

where (R) denotes the contents of one of the addressable general registers. The logic design of a gate network to implement these four operations and the reversal of the bits in the word was carried out to insure that implementation of such instructions was feasible. The entire network required approximately 65 gates, and even with standard TTL logic the delays through the network were compatible with a 200 nsec microinstruction cycle time.

In summary, the use of a user microprogrammable minicomputer will allow real time preprocessing of telemetry data with data rates in the range of from one to possibly five or six megabits per second. The data rates at the upper end of this range are not achievable with commercially available minicomputers but would require the hardware implementation of some special microinstructions. Chief among the special instructions that would be valuable are the reversal of the order of the bits in the word and multibit shift and rotate instructions. Implementation of these instructions in a user microprogrammable minicomputer is feasible if they are included during the design phase, and should not increase the cost of the computer appreciably.

III. The use of multiple minicomputers to achieve high data rate processing.

The problem in attaining 30 to 100 megabit data rates with real time processing does not lie in the use of minicomputers. While typical large scale computers may have multibit shift and rotate instructions, most such computers are not user microprogrammable. The programs for telemetry preprocessing on such machines therefore must be stored in and executed from main memory. This actually results in longer execution times than can be obtained with a user microprogrammable minicomputer. The greater word length and fast arithmetic hardware of the large computer are of virtually no benefit in most telemetry preprocessing problems.

These observations lead directly to the consideration of a multiple-minicomputer system of some sort as a candidate in the high data rate situation. A variety of configurations are possible, and some of the salient features of several are discussed below.

The "classical" configurations are the pipeline and the fully parallel structures, representing opposite ends of the spectrum in interconnection topology. Considering first a pipeline connection of minicomputers, the structure would be that of Figure 1. With a 200 nsec microinstruction cycle

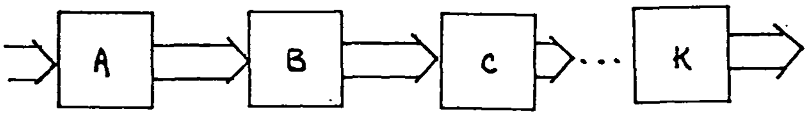

Figure 1. Pipeline configuration.

time for each of the processors this configuration is not attractive for achieving the data rates in question. At least an input instruction and an output instruction must be executed by each processor for each data word. No minicomputers appear to be available which are capable of simultaneous input and output, so at best two instruction cycles are required for each data word. This limit is achievable only if the computer has the ability to treat the input bus as though it were a register and combine the input of the data word with part of the processing in a single instruction. For most computers, and for some data formats no matter what computer is chosen, three instructions is the minimum value for a single processor. Even this assumes that the preprocessing algorithm can be broken down into single-instruction blocks. This results in very inefficient use of the processors, since each processor executes only a single computational instruction. It would be better to replace the minicomputers with special purpose hardware to perform the instruction in question if a pipeline structure is actually used. In any case three instruction cycles per processor yields data rates corresponding to 3 x 200 nsec per data word. For eight-bit data words this is a rate of only 13.3 megabits per second. In view of this figure no further consideration was given to pure pipeline organizations.

A purely parallel organization, as illustrated in Figure 2, could in theory provide any desired data rate.

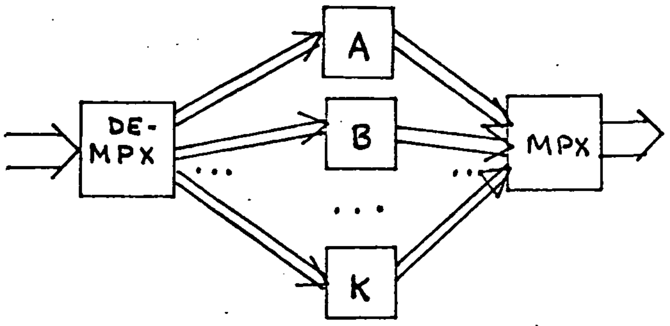

Figure 2. Parallel configuration.

As long as the demultiplexer at the front end can keep up with the incoming data it requires merely providing enough processors to handle the entire computation load in the available time. If, for example, processors similar to the MICRODATA 800 were used, with a single-processor data rate capability of 2 megabits per second, then it would require paralleling 15 such processors to achieve a 30 megabit data rate. The chief problems in such an interconnection of processors lie in the areas of timing and control. Some way of synchronizing the various processors must be used since interrupts and subroutine jumps in response to them are orders of magnitude too slow for the target data rates. Probably a single master clock for all processors will be required. The demultiplexer and multiplexer must be capable of operation at the data rate of the input telemetry stream. Unfortunately more needs to be done than just segment the incoming data stream into equal length words. The various telemtry formats used on a given spacecraft often have different word lengths and data rates. In addition there is sometimes a status word within the data to indicate which format the current frame is using. This will require that the front end demultiplexer have considerable computation and decision-making ability of its own; in fact a computer is needed here. The decision algorithms to identify the format of the incoming data are comparable in complexity to the format translation algorithms for the data words. If a computer could keep up with the decision-making for a 30 megabit input data rate, then a similar processor could handle the format translation and a two-processor pipeline could achieve the desired data rate. The problem is that processors with such speeds are not currently available, as pointed out in the earlier sections of this report. In addition, unless the control memory of each processor is large enough to hold all of the microprograms needed for all of the formats from a given satellite, it will be necessary to swap microprograms from main memory.

This certainly cannot be done within the available amount of time. The solution to these problems involves missing the first frame or two of data when a shift in format occurs, making use of the resulting time for the decision-making process and any needed swapping of microprograms. With this approach a minicomputer of the sort envisioned for the parallel processors can be used to control the demultiplexer.

With the parallel configuration of Figure 2, and ignoring the problems involved with the ends of a frame, data words are sent successively to different processors until the first processor has had time to complete processing of the previous word it received. This round robin distribution of data words is thus strongly dependent on the amount of time per data word required by each processor and on the number of processors. Adding another processor to such a system, or changing the microprograms, would require extensive changes in the program to control the demultiplexer.

A configuration which shows some promise from the viewpoint both of flexibility and of reliability is a distributed network illustrated in Figure 3. In such a

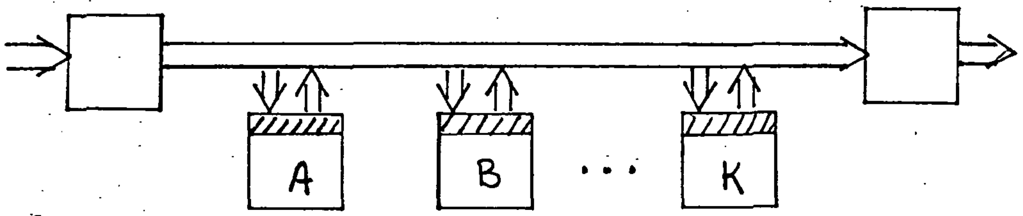

Figure 3. Distributed configuration.

network each computation task is not addressed to a particular processor but is put on a common communications link along with identifying information. Part of this identifying information indicates what processing needs to be done on the accompanying data. Hardware (possibly using associative memory techniques) in each processor checks to see if the processor is able (and free) to perform the required operations. The processor then either removes the data from the link and processes it, or ignores the message if it cannot perform the operation. In this latter case the message proceeds along the communications link until it encounters a processor than can accomodate it.

This configuration has the advantage that depending on the programming of the front end processor that is deciding what processing is required on each data word the system can behave like a parallel network, a pipeline network, or anything in between. The feasibility of such a system, as with any computer network, is extremely dependent on the choice of effective communications protocols. Some preliminary investigations have been made in this area, but it is not yet clear either what the protocols should be or what the details are of the optimum structure for the network. One possibility on structure is to form a "ring" of processors, as illustrated in Figure 4. This gives a system similar to the DCS system of Farber (2) which is valuable in telemetry preprocessing chiefly

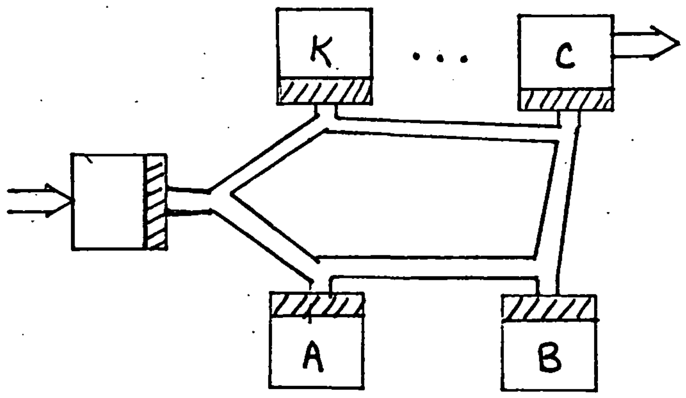

Figure 4. Ring configuration.

because it gives a more richly connected network, with resulting greater opportunities for continued operation in the face of failure of a processor.

Our preliminary studies indicate that the use of a network of minicomputers for telemetry preprocessing is promising. More work is needed on possible structures of the network, but the leading candidates at the present are the distributed and ring configurations. Many questions concerning the intercommunications between the processors remain to be answered, but the network approach seems to offer the best liklihood of achieving a general purpose telemetry preprocessor that will accomodate 30 to 100 megabit data rates in real time.

REFERENCES

1. Y. Chu, "An ALGOL-like Computer Design Language," CACM, vol. 8, October 1965, pp. 607-615.

2. D.J. Farber and K.C. Larson, "The System Architecture of the Distributed Computer System-The Communications System," presented at the Symposium on Computer Networks, The Polytechnic Institute of Brooklyn, April 1972.

APPENDIX A

SIMULATION OF SMALL MICROPROGRAMMED PROCESSORS

At the outset of the project it was anticipated that we would develop a microprogram simulator to check and compare different microprogram approaches. It is now out opinion that writing such a simulator would be a waste of effort. Any simulation program general enough to cope with even the spectrum of microprogrammed minicomputers now available would have to be as complex as existing languages. The power and flexibility of existing programs and languages, such as CDL[1] and APL , are at least the equal of any special simulation language that we could reasonably develop.

With these facts in mind we have attempted to determine what the efforts and costs for simulation of small processors typically are. To get comparisons we selected one particular microprogrammed minicomputer and set up three different simulations of this same machine. These simulations represent three rather different approaches to the problem of simulating a minicomputer. The three techniques compared were: 1) a simulation of the specific minicomputer written in FORTRAN; 2) a simulation of the minicomputer using CDL; and 3) anAPL simulation of the mini. Each of these approaches is discussed briefly below. The minicomputer chosen was the MICRODATA 800. Factors affecting this choice were that our investigations had shown it attractive for the telemetry format translation problem and the availability of a FORTRAN assembler and simulator for this machine.

The FORTRAN simulator used was written by (or at least for) the Microdata Corporation, and is supplied by them to users of the MICRODATA 800. It is written in ANSI FORTRAN and consists of approximately 5000 FORTRAN statements.

CDL[1,2] is a computer design and simulation language developed by Dr. Y. Chu at the University of Maryland. CDL compilers exist for several machines, including the Univac 1108. To simulate a minicomputer in CDL it is not necessary to write a complete simulation program, but only to describe the computer's architecture in CDL and specify the output desired from the simulator.

A simulation of the MICRODATA 800 in APL was developed. This approach, like the use of FORTRAN, requires writing a complete simulation program for each system to be simulated. The power and flexibility of APL makes the job much easier and faster in APL than in FORTRAN, however. Our experience indicates that it actually requires less effort to write a complete simulation program in APL than to prepare the necessary data to describe a minicomputer in CDL.

To provide a basis for comparison, two specific programs for the MICRODATA 800 were investigated. One was an 8 bit by 8 bit multiply routine, and the other was a program to translate an 8 bit word in OAO-A2 direct digital telemetry format. This format translation requires reversing the word (from lsb first to msb first) and complementing three specific bits within the word. Each of the two programs was run on each of the three simulators, and the average execution time results are given in Table II. All of the computer runs were made on the University's Univac 1108.

Table I gives comparative data and estimates on the simulation programs themeselves.

	FORTRAN	CDL	APL
Number of statements or cards	5000	450Δ	80
Time to produce simulation	5 man-months	3 man-weeks	2 1/2 man-weeks

Table I. Simulator Characteristics.

Δ In this case the 450 represents the number of cards needed to describe the MICRODATA architecture in CDL, not the size of theCDL program itself.

	MICRODATA 800	Simulators		
		FORTRAN	CDL	APL
Multiply routine (11 instructions)	15 μsec	0.71 sec	101 sec	18 sec
format translation program (18 instr.)	21 μsec	1.04 sec	232 sec	21.1 sec

Table II. Execution Times.

The figures given in the tables above represent preliminary and in some cases estimated data. The "time to produce simulation" figure in Table I for CDL represents the actual time required to prepare the input and get the simulation running correctly, and does not reflect any of the development time for CDL itself. The 5 man-month figure for the FORTRAN simulator is an estimate based on the length and complexity of the program. All of these figures in Table I assume that the person doing the work is familiar with both the language in question and with the structure of the minicomputer to be simulated.

The execution time results of Table II were somewhat surprising. Our expectation had been that CDL would be faster in execution than APL but slower than FORTRAN. On the basis of our experimental data, we would conclude that CDL is <u>not</u> an appropriate language for this sort of simulation. The choice then is between FORTRAN and APL. For a fixed computer architecture, where the problem is program development, the execution time advantage of the FORTRAN simulation would be likely to more than make up for the longer program development time. On the other hand, for investigating computer structures and architecture, the flexibility of APL and the ease of changing parts of the computer structure in the simulation would probably outweigh the approximately 20/1 execution time penalty relative to FORTRAN. We feel that the results obtained in this study are typical, in that similar results would be obtained in the simulation of any computer-like system of comparable size and complexity to a microprogrammed minicomputer.

References

1. Y. Chu, "An ALGOL-like Computer Design Language," CACM, vol. 8, October 1965, pp.607-15.

2. CDL-Users' Manual for the UNIVAC-1108, Computer Science Center, University of Maryland, August, 1070.

Appendix B

EXAMPLES OF MICROPROGRAMMED PROCESSORS

Three microprogrammed processors are described in this section. The Microdata MICRO 800[1] which is used in the examples in this report is a byte-oriented minicomputer which does not employ writable control store. The Hewlett-Packard HP 2100A[2] and the Burroughs Interpreter (D-Machine)[3] are increasingly sophisticated 16-bit processors which employ writable control store.* The following sections briefly describe the three processors. The microprogrammed arithmetic and logic operations of the three machines are compared in Table B-1.

MICRO 800. The MICRO 800 is an 8-bit, byte-oriented minicomputer with main memory cycle time of 1.1 microseconds and microinstruction cycle time of 220 nanoseconds. It contains 15 general purpose registers (file registers) plus a number of internal processor registers. The control store consists of up to four modules of 256-word by 16-bit read only memory.

Three microinstruction formats, reminiscent of the machine language instructions found in most small computers, are used. They are illustrated in Fig. B-1. A four-bit op code in Bits 15-12 is used to distinguish various microinstructions. Op codes 1 to 6 (hexadecimal) designate literal commands in which a literal (i.e., a constant) stored in Bits 7-0 of the command is loaded into various processor registers, used as a bit configuration of data value in comparisons (tests) with file registers, or added to the contents of a file register. Bits 11-8 designate the register used in the literal command. Op codes 7 to F designate operate commands which control the flow of data in or out and through the machine and perform arithmetic and logic operations. The arithmetic and logic operations consist of the following: Add, Subtract, Or, Exclusive Or, And, and Shift. Op code 0 designates an execute command which provides a means of modifying a microinstruction before execution.

*Other computers employing writable control store include: Interdata Model 80, Nanodata Corporation Model QM-1, Varian 73, Burroughs B1700, and Standard Computer MLP-900 (formerly IC-9000). In addition, a number of models of the IBM 360 and 370 computer families contain writable control store which cannot normally altered by the user.

Table B-2: Arithmetic, Logic, and Shift Operations Available

	MICRO 800	HP 2100A	D-Machine
Arithmetic			
$X + 1$	x	x	x
$X + Y$	x	x	x
$X + Y + 1$	x	x	x
$X - Y$ (2's C)	x	x	x
$X - Y$ (1's C)	x	x	x
$X - 1$			x
$X + (XvY)$			x
$X + (XY)$			x
Logic			
$\overline{X}$	x	x	x
$\overline{Y}$	x	x	x
XY	x	x	x
$X\overline{Y}$	x		x
$\overline{X}Y$			x
$\overline{X}\overline{Y}$		x	x
$Xv\underline{Y}$	x	x	x
$Xv\overline{Y}$	x		x
$\overline{X}vY$			x
$\overline{X}v\overline{Y}$			x
$X\overline{Y}v\overline{X}Y$	x	x	x
$XYv\overline{X}\overline{Y}$			x
Shift			
Left 1	x	x	x
Right 1	x	x	x
Left 4		x	x
Arbitrary			x

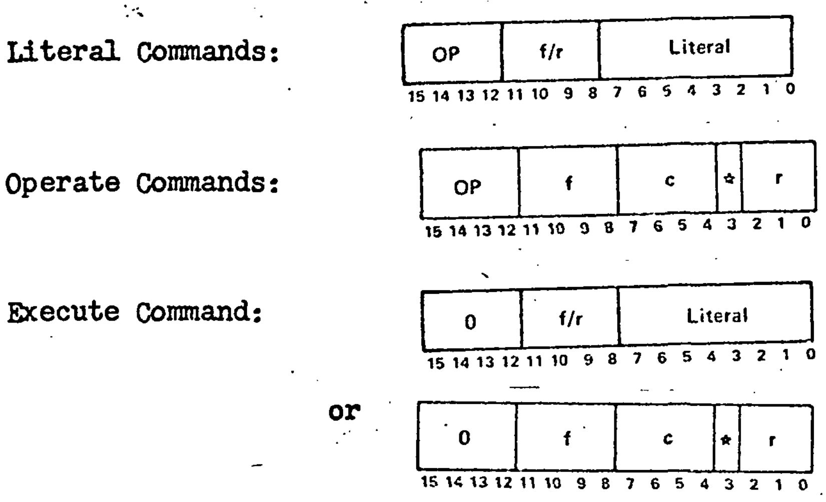

Figure B-1. MICRO 800 microinstruction formats. The notation is as follows.

OP--op code
f --file register designator
r --control register designator
c --control option bits
* --inhibits transfer of result to f
0 --op code of 0000

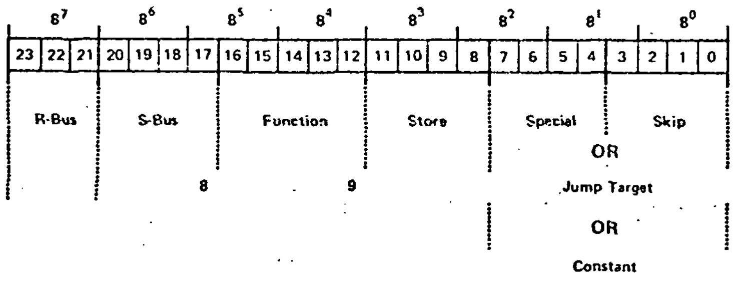

Figure B-2. HP 2100A microinstruction format.

HP 2100A. The HP 2100A is a 16-bit-word-length computer with main memory cycle time of 980 nanoseconds and microinstruction cycle time of 196 nanoseconds. It contains 8 hardware registers (four of which are scratch pad) plus a programm counter. It is not a totally microprogrammed computer as some of the controls for input/output operations and for skip instructions are hardware generated.

The control store consists of up to four modules of 256-word by 24-bit memory. The modules can include a combination of read-only memory and writable control store. Writable control store modules are installed in standard input/output slots and thus can be modified dynamically using standard input/output operations. Microinstructions implementing the machine operations considered to be the basic HP 2100A instruction set are normally stored in module 0. Modules 1, 2, and 3 are available for user-specified micprograms.

The microinstruction format for the HP 2100A consists of six fields. It is illustrated in Fig. B-2. Typical functions specified by each field are as follows.

R-Bus Field--specifies a register as one of the inputs to the arithmetic and logic unit (ALU).

S-Bus Field--specifies a register as one of the inputs to the ALU.

Function Field--specifies ALU function or other control function.

Store Field--specifies a register as destination of the ALU output.

Special Field--specifies a main memory read or write cycle.

Skip Field--specifies a condition for a possible microinstruction skip decision.

Burroughs Interpreter. The Burroughs Interpreter, also referred to as the D-Machine, is a 16-bit-word-length, general purpose, micprogrammable processor designed for use as a hardware building block in larger multiprocessing systems. The processor contains 3 general purpose registers and a number of internal registers. The architecture of the D-Machine is determined by its microprogrammed instruction set.

The control store is separated into two parts called the microprogram memory (MPM) and the nanomemory (NM). The MPM contains 16-bit microinstructions in blocks of 64 words, expandable to 4096 words. The NM con-

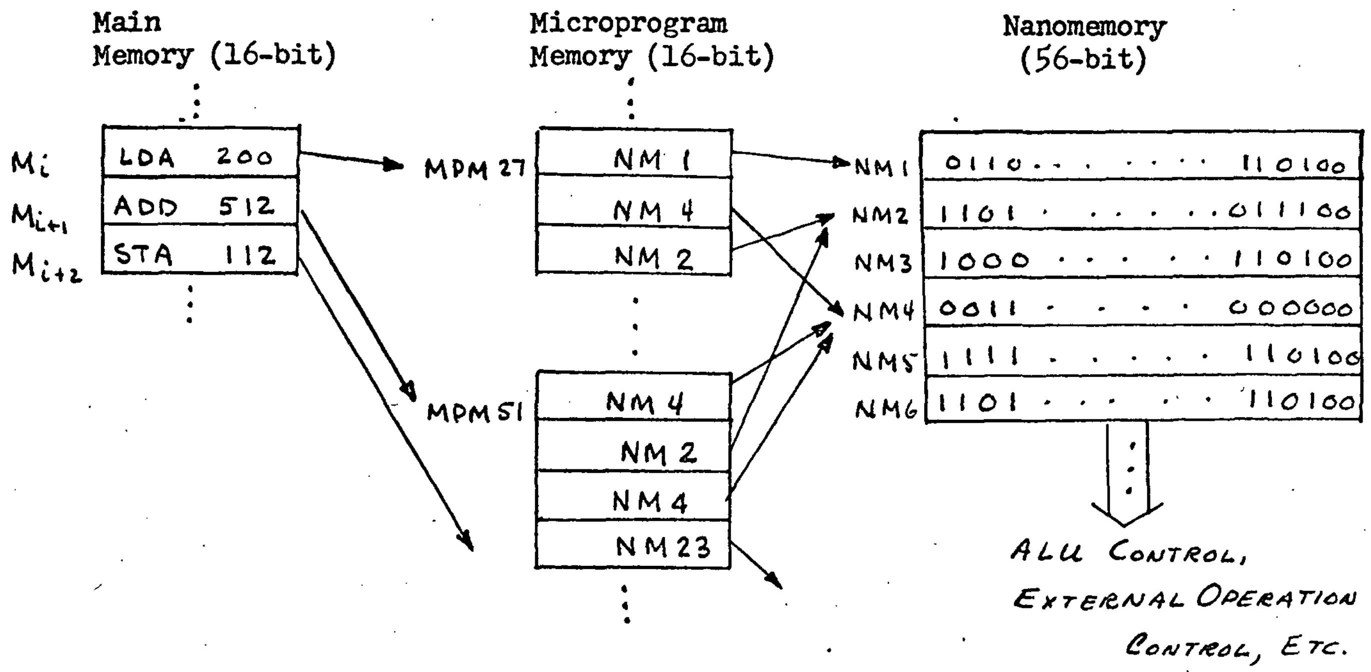

Figure B-3. Relationship among Main Memory, Microprogram Memory (MPM), and Nanomemory in D-Machine.

Table B-2: Nanoinstruction Fields in D-Machine

Nano-Bits	Function
1-4	Select conditions
5	Selects true or complement of condition.
6	Specifies conditional or unconditional LU operation.
7	Specifies conditional or unconditional external operation (memory or DDP).
8-10	Specify set/reset of condition.
11-16	Microprogram address controls (wait, skip, step, etc.).
17-26	Selects A, B, and Z.
27	Carry control
28-31	Select Boolean and basic arithmetic operations.
32-33	Select shift operation.
34-36	Select inputs to A registers.
37-40	Select inputs to B register.
41	Enables input to MIR.
42	Enables input to AMPCR.
43-48	Enable and select input to address registers and counter (MAR, BR1, BR2, CTR).
49-50	Select SAR preset.
51-54	Select external operations (read, write, lock, etc.).
55-56	Not assigned.

tains 56-bit nanoinstructions in blocks of 64 words, expandable to 4096 words. Figure B-3 illustrates the relationships among main memory, MPM, and NM. If instructions stored in main memory are referred to as machine instructions, each type of machine instruction provides an entry address into MPM where a sequence of microinstructions for executing the machine instruction is stored. Most of the microinstructions call for execution of a nanoinstruction in NM. Use of two levels of memory (MPM and NM) in the control store provides the powerful microoperation combinations associated with a long (56-bit) microinstruction word while minimizing the total number of memory bits required. An effective microinstruction cycle time of about 200 microseconds is possible using fast semiconductor memory.

Microinstructions are of two types. Type I instructions contain an address which specifies a location in NM from which a nanoinstruction is fetched and executed. Type I instructions are executed in two phases (Phase 1 and Phase 3), each requiring a single clock cycle. When two Type I instructions are executed in sequence, execution of Phase 3 of one instruction and Phase 1 of the following instruction is overlapped. Type II instructions contain literal data which are transferred to internal registers. Type II instructions are executed in a single clock cycle.

Of primary interest in this study are the operations which can be controlled by execution of a 56-bit nanoinstruction. Table B-2 illustrates the fields present in each nanoinstruction. The fields can be classified into four groups according to the time interval during which they exercise control. The time intervals and nanoinstruction bits for each group are as follows.

1. During Phase 1. Bits 1-5 specify a condition which is to be tested to determine the course of subsequent operations. Bits 6 and 7 specify whether the subsequent operation is to be an ALU operation or an external operation such as a main memory read or write.

2. At the end of Phase 1. Bits 11-16 specify how the next MPM address is to be determined. If an external operation has been specified, Bits 8-10 specify a condition which is to be altered and Bits 51-54 initiate a specific external operation.

3. During Phase 3. An ALU operation is executed. Bits 17-19 specify one input source to the ALU. Bits 20-26 specify a second input source to the ALU. Bit 27 allows or inhibits carry propogation between bytes. Bits 28-31 specify the arithmetic or logical operation to be executed by the ALU. Bits 32-33 specify the type of shift operation, if any, to be performed on the output of the ALU. It should be noted that the result of an ALU operation controls some of the processor conditions which are tested by the Phase 1 portion of the next instruction.

4. At the end of Phase 3. The result of an ALU operation is transferred to a number of destination registers. The destination registers and the types of transfer are specified by the fields in Bits 34-50 of the nano-instruction.

References

1. Microprogramming Handbook, 2nd Edition, Microdata Corporation, 1972.
2. Microprogramming Guide for Hewlett-Packard Model 2100 Computer, Hewlett-Packard Company, 1972.
3. Davis, R. L., and S. Zucker, "Structure of a Multiprocessor Using Microprogrammable Building Blocks", National Aerospace Electronics Conference, 1971.