FOREWORD

This study was initiated by the Simulation Techniques Branch, Behavioral Sciences Laboratory, 6570th Aerospace Medical Research Laboratories, Aerospace Medical Division, Wright-Patterson Air Force Base, Ohio. The research was conducted by IIT Research Institute of Chicago 16, Illinois, under Contract AF 33(657)-11007. Mr. Don Gum, Simulation Techniques Branch, served as contract monitor. The work was performed in support of Project No. 6114, "Simulation Techniques for Aerospace Crew Training," and Task No. 611408, "Simulation Computers." The effort covered in this report is the two month study phase started 1 May 1963, and ended 30 June 1963.

The IIT Research Institute personnel contributing to this report include K. W. Andresen, who also served as principal investigator, and Duncan Ewing. In addition, Eugen F. Uretz contributed to the study phase effort.

This report is a revision of IITRI progress report H6003-2.

ABSTRACT

In the study phase of the project to provide a general purpose laboratory facility for use in research in training simulation techniques, digital computer systems and interface equipments were evaluated for the application. Criteria for the system evaluation were obtained from previous studies, involving the F100A aircraft and EROS vehicle flight simulations using the UDOFT computer facility.

Requirements for the computer hinge on a real time operating capability which stresses high computation rates. Significant characteristics include:

1. An operating rate in excess of 75,000 instructions per second on flight simulation problems.
2. A memory capacity of at least 8,000 words,
3. A computer word length of at least 24 bits, and
4. At least three index registers.

The result of the study phase is a recommendation of the Packard Bell 440 as the central computer of the simulation system; and as an alternative, a recommendation of the faster SDS 9300 computer is made provided its higher cost and later delivery time are acceptable.

The recommended interface equipment will include a multiplexed analog-to-digital conversion subsystem capable of digitizing 32 input channels to 12 bits at a rate in excess of 35,000 conversions per second, a dual resolution digital-to-analog conversion system supplying 64 analog outputs to analog equipment, 72 sense inputs, 72 control outputs, and a digital interval timer.

PUBLICATION REVIEW

This technical documentary report is approved.

Walter F. Grether

WALTER F. GRETHER
Technical Director
Behavioral Sciences Laboratory

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF ILLUSTRATIONS AND TABLES

LIST OF ILLUSTRATIONS AND TABLES (Cont.)

EXPLANATION OF TERMS

AD.... An abbreviation for Analog-to-Digital.

BCD.... An abbreviation for Binary Coded Decimal, a system for representing alphanumeric characters as six digit binary numbers.

Channel.... A group of wires used to control and transmit data between the computer main frame and the Input-Output system.

Commutator.... A multi-position electronic switch which scans a set of lines. In the PB440 the commutator scans four interrupt lines.

Control Sequence.... A micro-sequence which obtains each program-level instruction, does any operations common to most (or all) microroutines, and transfers control of the microroutine appropriate to define each program-level instruction.

DA.... An abbreviation for Digital-to-Analog.

DAP.... An abbreviation for DDP-24 Assembly Program.

Hybrid Computation.... A computation employing both digital and analog computing elements. Such a system may be referred to as a hybrid system.

Instruction Sequence.... A series of instructions.

Instruction Set.... The control sequence and all the microroutines in the PB440 computer.

Interrupt Line.... A toggle in the PB440 which contains (as a voltage level) the "ready" status of the I/O device connected to the channel. The programmer may choose to have this "ready" status interrupt the program automatically or he has the option of testing the state of the toggle and then servicing the device.

Interrupt Mask.... A toggle in the PB440 which permits or prevents automatic program interrupt.

Interrupt Sequence.... The sequence which services an input or output device which has caused a program interrupt.

Literal.... The actual appearance of an operand in the modifier field of a micro-step, or in the address field of a command.

Logic Level.... In the PB440, the level at which micros are executed as opposed to program level where instructions are executed.

(Macro) Instruction.... A program-level instruction.

Micro-Pair.... Two micro-steps within the same memory cell. A micro-pair consists of a left-hand and a right-hand micro-step.

Micro-Sequence.... A series of micro-steps.

Micro-Step.... A basic (logic level) operation, which consists of six bits of operation code and either a six-bit modifier field or two three-bit register designators.

Microroutine.... A micro-sequence which defines the operation of a program-level instruction.

Mnemonic.... Alphanumeric characters which symbolically designate operation codes and registers.

Program.... A complete list of instructions used to control or direct the solution to a problem.

Program Interrupt.... The action in the PB440 by which the normal sequence of micro-step execution is altered by placing a micro-pair (from one of the first four locations of fast memory) into the logic level command ("E") register.

Subroutine.... An instruction sequence which is a component part of, and is called by a complete program.

Toggle.... An addressable one-bit register, or console switch.

RESEARCH AND DEVELOPMENT FOR A REAL TIME TRAINING SIMULATION RESEARCH SYSTEM

SECTION I

INTRODUCTION

A. BACKGROUND INFORMATION

The work in the field of flight simulation through the use of digital computers was initiated in 1950 by the Moore School of Electrical Engineering of the University of Pennsylvania under the sponsorship of the U.S. Naval Training Devices Center (USNTDC) (University of Pennsylvania, ref. 2 and 3). One of the results of the mathematical study was the logical design of a digital simulation computer, UDOFT, which was designed to solve the equations of flight in real-time. The remaining design and development work on the UDOFT system was completed by Sylvania Electric Products, Inc. under a joint Air Force-Navy contract (Sylvania Electronic Systems, ref. 8 and 9).

The experience gained by employing UDOFT in simulating aerospace vehicle flights has established the usefulness and advantages of digital simulation over analog simulation. In addition, studies by Mark E. Connelly, Electronic Systems Laboratory, MIT, have shown that relatively small general purpose digital computers are capable of solving problems as complex as the F100 simulation in real time and at a cost competitive with analog computer equipment (see Krasny, ref. 5).

Further advantages of the digital computer over the analog flight trainer computer include more flexibility in changing the mathematical model as aircraft manufacturer's data changes and greater adaptability to different aircraft types permitting the system to be employed in many different simulations. Thus, as techniques for digital simulation have evolved and computer design advances have lowered the cost per computation of digital computers, the trend toward digital simulation in operational flight trainers (OFTs) has been firmly established.

B. THE REAL TIME SIMULATION RESEARCH PROBLEM

The Simulation Techniques Branch of the Aerospace Medical Research Laboratories conducts applied research on engineering techniques to be used in the development of full mission trainers, mission element trainers, and other training devices required to support advanced weapon systems. This includes research on visual simulation, computer techniques, mathematical models and programming techniques.

To deal with these problem areas, a laboratory facility for research into these and other related areas is being developed for the Laboratory. The facility potentially could employ flight trainers in its program.

A flight trainer consists of an airplane cockpit and an instructor's console connected with a computer. As the pilot goes through the motions of flying, the signals are fed from the cockpit controls to the computer. On the basis of the position of the controls and the history of the flights, the computer determines the aircraft's rate of climb, velocity, altitude, etc., and feeds the computed values to the pilot's and the instructor's console instrument dials and indicator lights. At the same time, the instructor can arbitrarily introduce various circumstances, such as heavy icing, rough air, or failure of hydraulic systems, through the instructor's console; and the computer program changes the parameters accordingly. Thus, flight conditions can be simulated and the pilot's reactions monitored.

C. SCOPE OF THE RESEARCH AND DEVELOPMENT PROGRAM

The research and development program, which generated this report, is to determine the feasibility and detailed characteristics, develop and design, fabricate and install a real time training simulation research system. The work is divided into three phases as follows.

Phase I. Study. Work under this phase is to be completed in a two-month period and is to determine the feasibility of, and result in a description of, a simulation research system having these capability and functional characteristics:

1. A digital operand representation of not less than 22 binary bits.

2. Internal random access memory of not less than 8,000 words.

3. One index register.
4. Paper tape input/output.
5. Analog to digital converter providing at least 32 input channels.
6. Digital to analog conversion for at least 64 output channels.
7. Logical outputs for 64 lines.
8. Logical inputs for 64 sense switches.
9. A real time clock.
10. External interrupt provisions.
11. An assembly program capable of generating machine language code from symbolic program input.
12. A typical instruction execution rate of at least 75,000 per second.

Phase II. Development. Work under this phase consists of complete designs for the simulation research system, fabrication and installation of the system.

Phase III. Operation. Work under this phase consists of supplying initial operating, programming, and maintenance services and assistance to the government for a period of six months.

This report documents the work done in Phase I.

D. ORGANIZATION OF THIS REPORT

The following sections of this report explain the reasoning leading to the recommended real time training simulation research system and provide details of the system. Inasmuch as micro-programming on the Packard Bell 440 computer is not widely known, it was thought advisable to include a good deal of pertinent detail in appendices. Appendix I covers micro-coding on the PB440 in detail. Appendix II covers the Systems Command Set, a command set available for use with the PB440 which appears most suitable for use in real time simulation computation. Material included in Appendix II describes micro-coded routines in sufficient detail to aid in understanding and applying the material covered in Appendix I. Appendix III contains background information on complement arithmetic in relation to the PB440. Appendix IV contains comparison programs in various computer "languages." Appendix V provides functional descriptions of the two alternative computer systems recommended, the PB440 and the SDS 9300.

SECTION II

COMPUTER SYSTEM EVALUATION

A. PRELIMINARY SELECTION

The list of commercially available digital computer systems is a long one, covering a broad range of operating rates, word sizes, memory sizes and prices. The first step in evaluating potential systems for use in the Real Time Training Simulation Research System was to examine the requirements of the application and, keeping in mind the resource limits, narrow the field of computers to be considered.

The major requirements of the application which were employed to narrow the field were:

1. Types of computations normally encountered in scientific and engineering problems;

2. Operating rates high enough to simulate, in real time, modern aerospace vehicle dynamics and to compute flight paths.

The real time simulation requirement places primary emphasis on high operating rates of the digital computer system due to the sequential nature of digital computation. Since memory access time limits digital computation rates, only computers with very short access times are feasible, leading to the elimination of slow access drum or disc memory computers from consideration. Furthermore, magnetic core memory computers having effective memory cycle times greater than six microseconds were also eliminated from consideration.

Scientific and engineering computations in general are performed more efficiently on binary word-organized computers rather than on BCD or character-organized computers. To demonstrate this, assume that a computer system were required to deal with numbers modulo 10^8. Such numbers coded into four bit BCD representation would require 32 bits whereas a binary word of 27 bits is sufficient to code a number of even a large modulus ($2^{27} = 134, 217, 728$). This more efficient use of bits afforded by the binary word-organized computer results in less hardware and thus less cost, and provides faster operating rates for those operations whose rates are bit dependent such as multiplies, divides, and shifts. Thus the BCD, or business-application, oriented computers were not considered in the evaluation as being non-competitive in operating rate with binary scientific oriented computers. (See Grabbe etal, Ref. 21)

Miscellaneous factors which further narrowed the field were:

1. The computer would be domestically manufactured to avoid communication and maintenance problems with the manufacturer;

2. The electronics would be transistorized for reliability, low power consumption, and economy.

3. Overall system cost to be less than $300,000.

The field of computers to be evaluated was thus narrowed to include:

1. ASI 210

2. Beckman 420

3. CDC 924

4. DDP24

5. PB440

6. PDP 1

7. PDP 6

8. SDS 9300

9. Sylvania M64.

The further reduction of this list of computers was done with a detailed application in mind. The application selected for this purpose was the F100A simulation problem, an ideal application in that much information is available from the UDOFT simulation of the F100A and that work continuing at MIT in operational flight trainers is based in large part upon the F100A simulation problem.

Prior to comparing operating rates of these computers, the word size requirement was considered. Experience in simulating an orbiting vehicle on UDOFT has demonstrated that the integration routines employed would have benefited by the use of longer word lengths than the 20 bits plus sign available. This was particularly true when integrating the orbital angular velocity to obtain distance traveled where the time increment employed was the conventional 50 millisecond solution interval. It was found that the least significant bits of the increment disappeared when the integrand was accumulated, creating rather large errors in the problem results. Since the trend in simulation will be toward higher iteration rates to cope with higher frequencies involved in reentry and docking problems, the necessity for longer word lengths to maintain precision in integration will correspondingly increase. This reasoning has led to a requirement of considering computers with word lengths of at least 24 bits, which eliminated the ASI 210 (21 bits), Beckman 420 (18 bits), and PDP 1 (18 bits) computers from further consideration.

In evaluating the operating rates of various computers in light of the requirements of the F100A simulation program, an expression for a figure of merit, adapted from Connelly (ref. 1), is used. This figure of merit is essentially a rough measure of the number of computer instructions that may be performed in the solution interval. This interval is usually determined by requirements in the specific simulation problem and in most simulations is based upon the highest frequencies expected in the solution. The instruction periods are estimated to be twice the memory cycle time for all but the multiply and divide orders. Multiply and divide orders are limited by the arithmetic unit rather than the memory access, requiring much longer to execute. The duration of the multiply is taken as typical of these extended arithmetic sequence orders since divides are used less often and can be relatively rare in simulation computations. The expression for the figure of merit NR may be taken as

$$NR = \frac{h}{AT + K(MT - AT)}$$

where h is the solution cycle time, taken to be the 50,000 microseconds used in the F100A simulation; K is the ratio of long computation time instructions (multiply and divide orders) to the total instructions in the F100A simulation and is taken to be 0.10; AT is the execution time of the memory access limited instructions (all but multiply and divide); and MT is the execution time of the multiply order. Table I gives NR for various computers. Note that AT and MT are given in microseconds.

TABLE I

FIGURE OF MERIT

COMPUTER	AT	MT	NR
CDC 924	9.3	37.5	4,120
DDP24	10	31	4,130
PB440 - SCS	10	40	3,840
PDP 6	8	15	5,750
SDS 9300 (4K mem)	3.5	8.75	12,400
SDS 9300 (8K mem)	2.8	7.9	15,100
Syl. M64	8	70	3,520
UDOFT	5	10	9,100

Connelly has further defined a computability index which includes a term that is a diminishing function of memory size to give weight to the trade off between operating rate and memory size. This index is given as

$$IC = NR \left[1 - \exp(-S/NR)\right]$$

where NR is the figure of merit defined above; and S is the number of words of magnetic core memory.

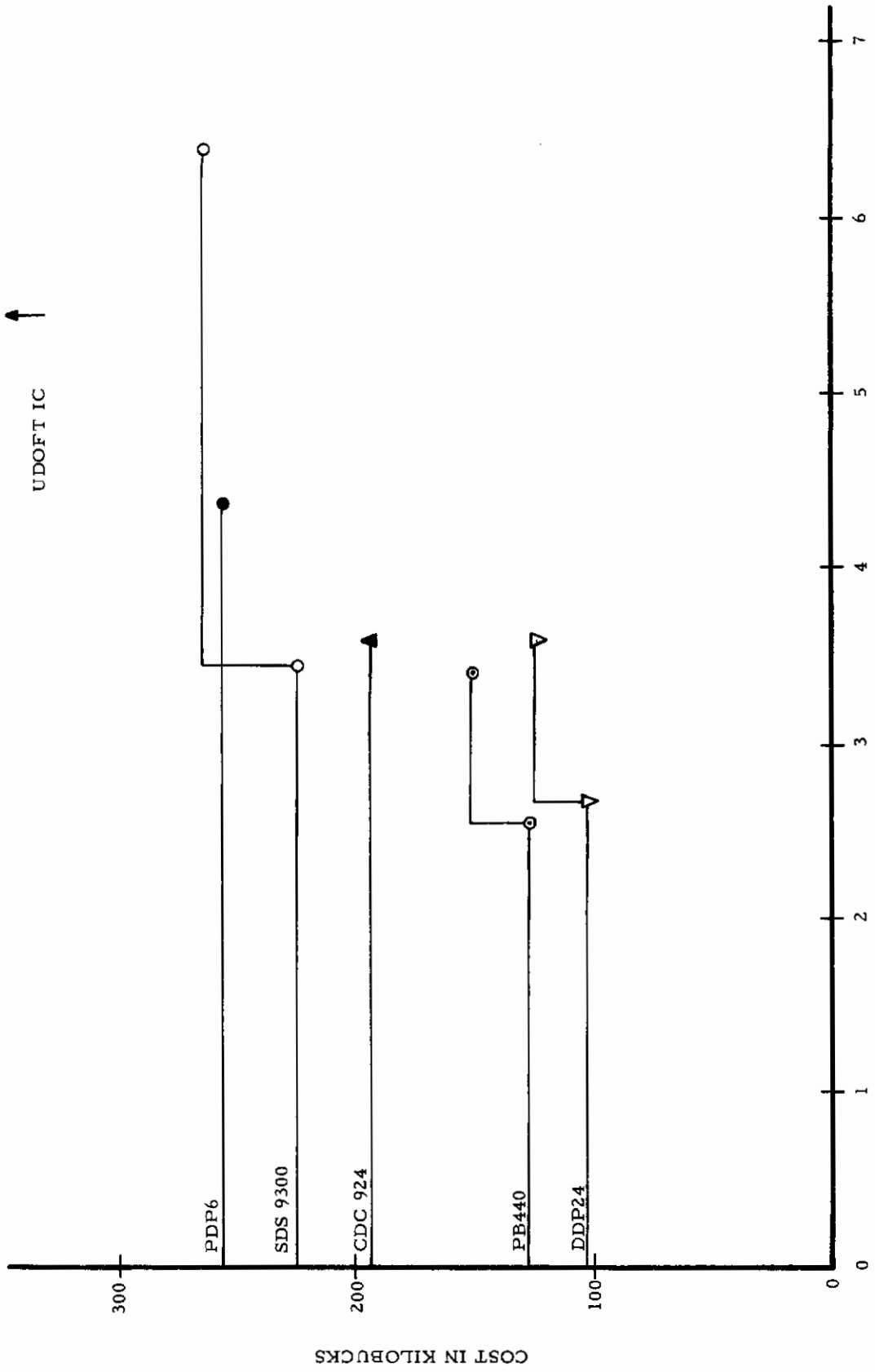


Figure 1 Computability Index vs Cost

Since IC is a function of memory size and the cost of memory is known for the various computers, the plot of IC versus computer cost is now possible. This is given in figure 1 for various computer systems. The cost figures shown are for a basic computer with at least 4096 words of storage, 3 index registers, paper tape input/output, and typewriter input/output. The step function is a result of adding modular units of memory to the computer, usually in banks of 4096 words. In the cases of the CDC 924 and the PDP6, however, the modules are available only in 8192 word sizes.

It is seen that the PB440 and the DDP24 are the most attractive equipments from a cost standpoint, and for this reason they were singled out as the two computers to be extensively evaluated in the study phase. The CDC 924 and Sylvania M64 are priced significantly higher; enough to exceed the cost limit when the analog input/output subsystem is incorporated.

The PDP 6 and SDS 9300 computers are also priced significantly higher, but have figures of merit comparable to or exceeding that of UDOFT and thus attract attention to their important capabilities in a real time application. The PDP 6 has a delivery scheduled for July 1964 at the earliest and so cannot be further considered. The characteristics of the PDP 6 are worthwhile pointing out in passing, however.

The PDP 6 word length is 36 bits, and memory of 4 microsecond cycle time and 2 microsecond access is supplied in 8, 192 word modules. When two 8, 192 word modules are in the system, memory overlap is feasible. The instruction repertoire includes several half word (18 bit) arithmetic operations and flexible half word transfers between the accumulators and memory. Floating point instructions are included which operate with 36 bit operands: 1 sign bit, 8 exponent bits, and 27 fraction bits. A 16 word fast memory module, with 0.5 microsecond cycle time, is used for accumulators, index registers and short program loops. The first 8 locations of fast memory may be used as either 8 accumulators or 15 index registers or some combination of the two. The PDP 6 with 16K memory and floating point is quite competitive with the SDS 9300 based on the preliminary manufacturer's data. The attractiveness of the PDP 6 for simulation work is in its combination of high operating rate, floating point instructions, and 16K memory. The floating point instructions take a load off the programmer in scaling and the 16K memory permits extensive storage of aerodynamic function tables. It is unfortunate that delivery schedules prohibit further evaluation of the PDP 6 computer.

Following the preliminary selection, the computers remaining for detailed study were the DDP24, the PB440 and the SDS 9300.

B. COMPARATIVE EVALUATION OF DDP24 AND PB440

The DDP24 computer is of conventional organization and is readily understood by those familiar with computers currently in use such as the 704, 7090, 1105, 2000, 1604, etc. The PB440 however is somewhat different in that the instruction execution is not completely wired in but instead is determined to a large extent by "micro-programs" stored in a special memory and executed as conventional stored programs. The micro-programs and the associated micro-program memory are rightfully to be considered a portion of the control section of the computer. But where the control section of conventional computers is hardware, the control section of the PB440 is a marriage of hardware and software. This concept of hardware-software instruction execution is also known as "stored logic" in contemporary computer terminology.

An advantage of the PB440 micro-programming capability is that problem-oriented instruction repertoires may be devised and inserted into the PB440 without the necessity of disturbing so much as a wire. This advantage comes at the cost of some additional hardware in the control section and of considerable programming effort if a unique problem-oriented instruction repertoire and assembly routine is to be developed which optimally utilizes the PB440 micro-programming features. This last disadvantage is somewhat eased since the computer manufacturer plans to provide several user-oriented instruction repertoires in his software package.

The PB440 and DDP24 computers are both scheduled to come equipped with assembly routines which appear to be complete in all respects. The DAP language for the DDP24 will provide the usual program listings with elementary diagnostics, and relocatable object programs with automatic subroutine linkage. The DAP system is supported by service routines such as dumps, traces, and updates. The Packard-Bell symbolic assembly routine for the Systems Command Set has the same basic capabilities, and sufficient attention seems to have been given to the peculiarities of designing an assembler to permit the user to expand the instruction repertoire for his own applications. However, at the micro level, the unusual "stored-logic" feature of the PB440 can be expected to raise a number of considerations which will be novel for most programmers. Several of these are discussed below.

An immediate instance of the peculiarities of "micro-programming" is the following: it is by no means obvious, after a reading of the micro-instr ction set given in Appendix I, how the programmer can make a simple random memory acce say to location 1000 in main memory. The "LDM R1 R2" instruction loads into register R2 the contents of the memory location whose address is in register R1, but how is the proper address to be formed in R1? Loading the address from memory is not a solution, since one is again faced with the same problem: its address must previously be located in a register. The following two micro-pairs, however, will do the job.

```
LDI   P D      X→D, P+1→P
LDM   D D      (1000)→D

      0000
      1000     Operand X
```

Here the second micro-pair is the required address, stored in the instruction sequence, but not executed as an instruction.

In addition to curious techniques of this nature, the micro-programmer will have to become conversant with different forms of number representation. Most machines have their method of representing negative numbers built in, but the PB440, having essentially only one arithmetic operation, addition, places the burden of devising a method for handling signed arithmetic on the user. The machine was apparently designed with sign-magnitude numbers in mind, but the experience of Packard-Bell programmers so far indicates that 1's or 2's complement arithmetic can generally be handled faster. The choice between the latter two has apparently been resolved in favor of 2's complement, and the Systems Command Set is designed to operate with numbers in this form. However, investigation has shown that 1's complement can be made a little faster in some operations, although it is probably a little more awkward for the programmer. In any case, programmers desiring to extract all possible benefits from the flexibility of the PB440 will want to be familiar with both forms, and a short discussion of each is included in Appendix II.

Another interesting aspect of programming on the micro-level lies in the mechanics of various systems of interpreting macro-instructions. It is Packard Bell's intention at the present time to provide several alternative control sequences with its Systems Command Set, one of the several sets of instruction repertoire Packard Bell will provide in the software package, and to allow the programmer to specify before assembly which control sequence he will use. Examples of these control sequences, which differ in timing characteristics and address-modifying capability, are given in Appendix II. However, much could undoubtedly be done to improve this system; for example, microutines are given in Appendix II which could be used to alter the control sequence from one form to another under program control. This would save time by using the faster, non-indexing sequence in parts of the program which did not use indexing. Another microutine is given, Index Next Instruction, which could be used to index a single address more quickly than the control sequence could be changed. The same technique could be used for indirect addressing. Another very interesting possibility would be to eliminate the central control sequence altogether, and add two micro-steps to the end of each microutine whose function would be to interpret the next macro-instruction. The operation of returning to the main sequence, which now terminates every microutine, would then be eliminated. Indexing could be handled in a manner similar to the INI instruction above. These and similar possibilities will no doubt occupy much of the attention of PB440 users in the near future.

From the above, one may gather that the PB440 poses unique problems to the programmer. However, these problems usually represent opportunities for optimization which are not normally available in other machines. Although the techniques involved in micro-programming are novel, there is certainly nothing intrinsically difficult about them, and it is felt on the whole that the added complications are compensated by the possibility of writing programs which operate at faster rates. Beginning with the Systems Command Set, which is comparable in ease of programming to the DDP24 DAP assembly system, a further expenditure of programming effort will permit PB440 programs to have faster operating rates. And it seems, as the programmer expends more effort in the PB440 program, that he can increasingly obtain higher operating rates until he reaches the limit of either fast memory available for microutines or approaches a limit of about twice the DDP24 operating rate. This concept is qualitatively indicated in figure 2 below.

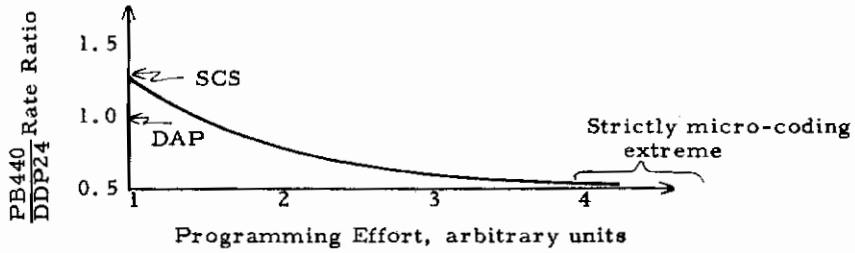


Figure 2 DDP24 - PB440 RATE RATIO

The DDP24 does not permit this type of program optimization since its instruction code is wired in. Only by hardware modification can the DDP24 be optimized for an application.

The vital consideration in the selection of a computer for flight simulation is operating rate. One way to estimate this figure is to select a representative set of operations, weigh each one according to its frequency in the type of problem to be solved, multiply each by its execution time for the given computer, and sum the results. This process was carried out for the PB440, the DDP24, and the SDS 9300, using instruction frequencies based on the F100-A simulation written for UDOFT. The results are shown in table II. The times in this table show that only the SDS 9300 can be expected to surpass the performance of the UDOFT computer.

For the PB440 Systems Command Set times, a quantity x must be added corresponding to the control time needed to interpret each macro-instruction. The non-indexing interpretive control sequence provided in the Systems Command Set requires a minimum of 4 microseconds; 5 microseconds on certain instructions when the instruction and operand are in the same memory bank. Also available in the Systems Command Set are interpretive control sequences having indexing capability but which require from 6 to 11 microseconds for execution.

It is desirable to have x as short as possible and yet retain the indexing capability where required. One technique for decreasing x which was investigated is to employ the indexing control sequence only in that portion of the program requiring this feature. It was found that more then 85 percent of the instructions could be carried out using the 4 microsecond sequence; the remaining 15 percent using the 7 microsecond sequence. Employing the Enable Index Registers and Disable Index Registers instructions, shown in detail in Appendix IV, the average x for an F100 simulation program is about 4.45 microseconds. Allowing additional time for possible memory overlap problems, we may take the average control time to be 5 microseconds. With these assumptions, the F100 simulation iteration period on the PB440 is 17 percent longer than on the DDP24.

TABLE II

F100 SIMULATION AVERAGE INSTRUCTION TIMES

Operation	Relative Frequency in F100 Simulation (%)	UDOFT Time	DDP-24 Time	PB440 Time (SCS)	SDS 9300 (4K mem)	SDS 9300 (8K mem)
Store	21.5	5	10	3 + x	5.25	4.38
Uncond. Jump	13	10	5	2 + x	1.75	1.75
Clear Add	18	5	10	4 + x	5.25	4.38
Multiply	9.5	10	31	40 + x	8.75	7.9
Shift 4	8	5	9	11 + x	3.50	2.62
Subtract	5.5	5	10	7 + x	3.50	2.62
Add	12.5	5	10	3 + x	3.50	2.62
Jump On	5	5	6*	5 + x	8.75	7.0
Clear Sub	2.5	5	15*	5 + x	7.00	6.1
Sub Mag	2	5	10	9 + x	7.00	6.1
Store Address	1		10	7 + x	3.50	2.62
Divide	.75	105	33	68 + x	17.5	16.6
Add Mag	.5	5	10	9 + x	7.00	6.1
Jump on 0	.25	5	5	6 + x	8.75	7.0
Average Time		6.83	11.35	8.26 + x	5.0	4.1

Notes: 1. Times given in microseconds.
2. * indicates two operations required.
3. "x" is control sequence time; average value 5.

The add instruction time given in Table II for the PB440 takes advantage of a special situation in the SCS system in that it is located immediately preceeding the control sequence. Thus it does not require the unconditional branch to the control sequence at its termination, thereby saving two microseconds. Table I of course does not reflect this time savings since it is not representative of typical SCS instruction execution times.

The savings in time afforded by elimination of the return branches to the PB440 control sequence for every SCS command could, however, be realized by inserting a control sequence at the end of every SCS microutine. It is estimated that such a procedure would save about one microsecond per SCS command execution, but only at a cost of storing two additional micropair per command microutine.

The PB440 may also be programmed in a conventional manner without the use of the Systems Command Set and the macro instruction interpretive method by coding exclusively at the micro level. The elimination of the control sequencing time and return branches afforded in this method would effectively reduce the average instruction execution time for the PB440 given in Table II by six microseconds. Thus one may predict that PB440 routines, wholly contained in the fast memory and coded at the micro level, would execute in only 64 percent of the time required for the equivalent execution in the DDP24. That this enhanced PB440 operation could actually be achieved is not claimed here in view of the small fast memory size (512 micro-instructions). However this is useful in that a rough measure of the upper bound on program execution rate in the PB440 in comparison with the DDP24 is delineated.

TABLE III

DDP24 - PB440 RATE RATIOS

Routine	Length	DDP-24 time	SCS time	SCS with MAD time	Micro-Routine time	SDS 9300
Level-Select and Slope	12	1.0	1.43		.71	
Mod-Gurk Integration	25	1.0	1.18	.92	.60	.28
Trapezoidal Integration	33	1.0	1.29		.49	
Direction Cosines	173	1.0	1.27	1.17		.32

In the above table the "SCS with MAD" columns show another feature of the PB440 system, namely, the possibility of adding to the Systems Command Set list macro-instructions which are specifically designed for the calculation at hand. The MAD command, a feature built into the hardware of the UDOFT computer, adds the previous contents of the accumulator to a newly-formed product, and is seen to have an appreciable effect on the times of two of the above routines.

From the above data one can conclude that:

1) Straightforward use of the Systems Command Set leaves the PB440 at a disadvantage of perhaps 20% in operating speed compared with the DDP24.

2) Careful use of the Systems Command Set, with problem-oriented instructions, arrangement of instructions and data as far as possible in different memory banks and optimal handling of control sequences, should give the PB440 operating speeds approximately equal to those of the DDP24.

3) Substantial savings, on the order of 50%, may be achieved by the PB440 whenever whole subroutines can be micro-coded in fast memory.

C. INDEX REGISTER REQUIREMENT

Although UDOFT functioned successfully with only one index register, it is felt that it would be advantageous to have more, for the following reasons:

1. Present day computers have developed other uses for index registers beyond their original purpose of address modification. In particular, they are also used as iteration counters and for the temporary storage of addresses. Clearly the simultaneous use of several of these features is hampered if there is only one index register.

2. Experience with UDOFT has shown occasionally the desirability of more elaborate address-modification facilities than the one index register available. One case in point was the EROS simulation where multivariate polynomials were employed to represent approximations to aerodynamic functions. It was found that on a digital computer with full indexing capability (at least three index registers) one program could have been written to evaluate any of the monovariate or bivariate polynomials required, replacing the three that were used.

The basic DDP24 comes equipped with one index register, and two more are available as an option. The DDP24 has operation codes intended for using the index registers for address storage, iteration counting, and also shift counting for the manipulation of floating point numbers. Address modification with the DDP24 is especially economical as it adds no time whatever to the execution of instructions. For example, a trapezoidal integration subroutine which required 450 microseconds using only one index register was programmed for three index registers to give a running time of 279 microseconds.

The PB440 does not have index registers in the normal sense, but it does have seven memory locations which can be accessed by three bits of an instruction word. To the macro-level programmer the PB440 appears to have seven index registers and a wide variety of operations utilizing them. It is to be noted that, although the PB440 has seven index registers compared with three for the DDP24, the speed of indexing operations is slower on the PB440 due to the additional memory access being required to fetch the index operand.

D. WORD LENGTH REQUIREMENT

Experience with the EROS simulation integrations on UDOFT indicates that an operand word length of 21 bits is not sufficient when dealing with real time orbiting trajectories using a 50 millisecond computation interval. The case in point resulted from the integration of an orbiting angular rate of 0.001 radians per second at the solution rate of 20 iterations per second. If the integrand word is scaled to accommodate 2π radians, it is seen that the radian change in the solution interval is on the order of 2^{-17} of the full scale value. Assuming a 0.1% error limit is desired, a minimum of an additional 10 bits of significance should be carried in the integration. The resulting word length requirement is seen to be 27 bits as a minimum. It was estimated that 26 bits plus sign should have been available for the EROS simulation.

In a study of digital simulation computer requirements for future aerospace vehicles, Perry (ref. 6) evaluated word length requirements for both fixed and floating point simulation computations. The fixed point operand length requirement was based on an integration similar to the EROS problem, this example involving the change in altitude of 10 feet per minute of an aircraft at an altitude of 100,000 feet. It was found that the word length must be at least 24 bits for the altitude to change at all in the 50 millisecond interval. Additional bits would have to be carried above this minimum requirement to preserve accuracy in the computation.

In addition to these instances, it seems likely that other cases will arise in aerospace simulation which require more than 24 bits of precision. However, if a computer word length must be long enough for all cases, a price is paid both in slower computation rates of multiplies, divides, and shifts (more bits imply more additions in multiply and divide), and for expanded memory and operand register hardware. An alternative approach to take, instead of expanding the word length of the computer, is to use double precision arithmetic routines. However, if a substantial portion of the computation is to be converted to double precision arithmetic, a significant lowering of the computation rate results; enough to prevent the computer from maintaining real time operation. Thus it appears that the optimum operand word length for simulation work would permit sufficient precision for integration yet requires few double precision arithmetic computations. A more suitable solution to the problem of specifying a word length would be to employ a computer with a variable word length or several word-lengths from which to choose to optimize any given problem segment.

Because much of the power of the modern general purpose digital computer resides in its ability to store both operands and instructions in the same memory, thereby permitting programmed modification of the running program and time sharing of common transmission paths, instruction word length requirement must also be considered in the word length specification. The minimum instruction word length for flight simulation is one which provides at least:

1. A single address field with sufficient bits to code the 8,192 storage addresses in the computer, (13 bits),

2. An operation code field sufficient to code at least 32 commands permitting an instruction list of sufficient versatility for simulation computation, (5 bits), and,

3. An index field sufficient to identify uniquely at least 3 index registers, (2 bits).

The resulting minimum instruction word length is 20 bits.

The word length consideration is further restricted by available binary computers which invariably employ word lengths evenly divisible by 6, resulting in 12, 18, 24, 30, 36, 48, etc., bit word systems. Once the instruction word minimum length of 20 bits is accepted, the possible word lengths to consider narrows to 24, 30, 36, 48 and over. Available computers with word lengths of 48 bits or more are outside the cost range of the project, or are lacking in other characteristics, and so are excluded from further consideration. Of the remaining candidates, 24 bit word computers are the most economical due to cost being an increasing function of the bit length. From UDOFT experience, it appears that 21 bit operand word lengths were sufficient for all but a few exceptional computations. By specifying a minimum of 24 bits, the amount of double precision computations required can be held to an acceptable level, and at the same time an economical 24 bit computer can be considered for simulation computation.

The PB440 computer meets the word length requirement of at least 24 bits as do the DDP24 and SDS 9300. When more than 24 bits of precision are required, the PB440 offers significant operating advantages over the DDP24 for two reasons:

1. Partial word operations are easily performed and,

2. Floating point operations are speeded by micro-steps which can manipulate independently the sign, exponent, and fraction fields.

The operational ability of the PB440 through micro-coding enables it to perform partial- and multiple-precision arithmetic very conveniently. Not only is double precision arithmetic done faster than on the DDP24, but the multiply or divide instructions may be micro-coded for any number of bits. For instance instead of settling for either a 24 bit or a 48 bit multiply as with the DDP24, the PB440 programmer may utilize a 32 bit multiply if he needs it, thus realizing a savings in time over the 48 bit multiply while maintaining an adequate precision. Comparative rates for multiple precision arithmetic, both fixed and floating point, are given in Table IV.

TABLE IV

DOUBLE PRECISION TIMES

Instruction	DDP24	PB440	SDS 9300 (8K mem)
Add Double Precision Fixed	55	24	3.5
Add Double Precision Floating	181	48	14-21*
Multiply Double Precision Fixed	263	97	
Multiply Double Precision Floating	371	122	12.25*

NOTE: Numbers are times in microseconds.
* with optional floating point hardware.

Floating point operations on the PB440 are facilitated by:

1. the micro-steps which can operate on the fraction fields independently,
2. the exponent register N,
3. the carry toggle from bit 9,
4. four full word arithmetic registers, and
5. suitable test micro-steps.

The partial word capability of the PB440 is also of importance for sub-multiple precision operations. Cases in the simulation area where say 12 or 14 bit operand arithmetic is advantageous are:

1. Scaling and limit testing operands from the AD converter,
2. Limit testing operands to be sent to the DA converter, and
3. Evaluation of aerodynamic functions, using low precision stored tabular data.

The SDS 9300 computer as well as the PB440 has capabilities in the multiple precision area, with its Twin Multiply operating on 12 bit operands, and several double precision fixed point instructions including load, store, add, and subtract and the optional floating point instructions. In the variable precision area, the SDS 9300 is not as versatile as the PB440, but is sufficiently fast to not require this degree of versatility. The SDS 9300, if employed in the EROS simulation, would certainly reduce many of the programmer's problems with scaling of wide ranging variables and with precision in integrations through the use of high-speed double-precision arithmetic operations.

SECTION III

REAL TIME INPUT/OUTPUT SYSTEM

The system containing both analog and discrete inputs and outputs together with the digital clock is relatively independent of the computer system selected. However, some details of the system will be affected by the computer selection. These are:

1. Digital signal characteristics including: logic levels, rise and fall times, impedances, and pulse timing;
2. Command word control bit function assignment;
3. Operand number system employed, i.e., one's or two's complement or sign and magnitude binary representation;
4. Interrupt processing, and
5. Control logic.

The Real Time I/O controller operation will depend to a significant extent upon the computer selected. In the PB440 system, it will stand as a fairly independent control unit, attached to the input and output data buses. In the SDS 9300 system, however, some I/O controller functions may already be provided for in the automatic buffered input/output channel. This channel is a communication path for direct transfer of I/O data to and from memory and comes as part of the basic SDS 9300.

A. PROGRAM CONTROL OF THE REAL TIME I/O SYSTEM

The communication paths between the computer and the I/O controller are shown in Figure 3. The output operand bus conducts both command and data words to the I/O controller from the computer. Separate control lines from the computer inform the I/O controller: (1) when the operand is on the output bus and (2) the type of operand, whether command or data.

The input operand bus conducts input data and I/O controller status information to the computer from the I/O controller. Identification of the data word is known to the programmer since he commands all I/O operations through the use of the input/output instructions in the computer instruction repertoire. Since the programmer controls the I/O, the responding data will be well defined as to its source, whether it be a given digitized analog input, a clock readout, or a controller status word, etc.

Additional control lines carry interrupt signals to the computer and interchange time synchronization between the computer and the I/O controller.

Command options for the Real Time I/O controller provide the following control of I/O:

1. Selection of input type
   a. Analog input via AD converter and multiplexer
   b. Discrete input
   c. Remaining clock interval
   d. Status word
2. Selection of output type
   a. Analog output via DA converter
   b. Discrete output
   c. Interval time
3. Selection of channel
   a. DA conversion, 1 of 64
   b. AD conversion, 1 of 32
4. Clock on/off control
5. Calibration check control

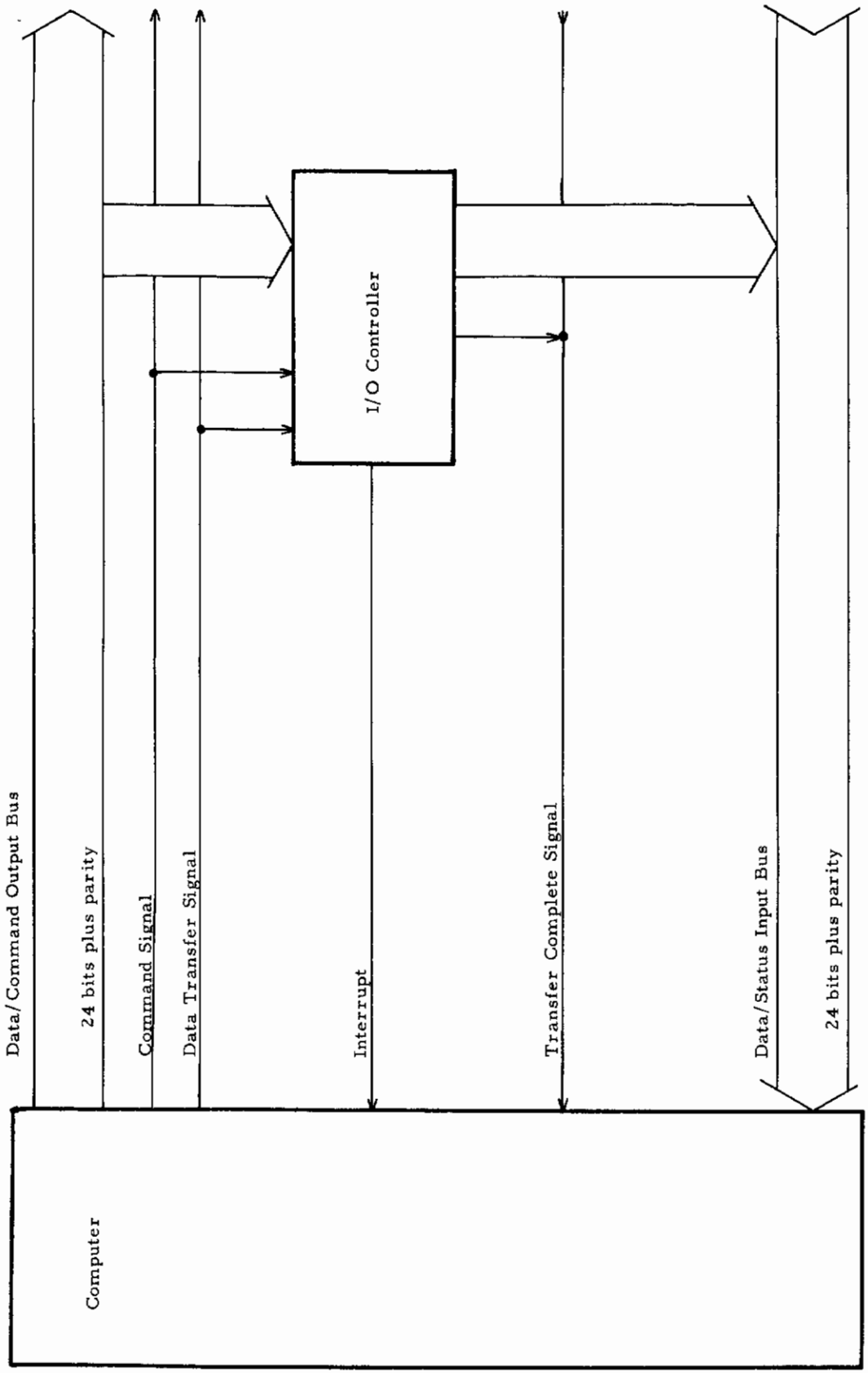


Figure 3 Input/Output Communication Paths

B. ANALOG INPUT

The analog input subsystem will employ a high speed multiplexer-AD converter combination capable of digitizing 32 channels of bipolar analog signals to 11 bit-plus-sign binary digital representation. The input signal levels are to be less than or equal to ±10 volts full scale with a source impedance of less than 1000 ohms. A high conversion rate is desired to minimize time skew errors between input channel samples without going to the expense of simultaneous sample-and-hold circuits and at the same time maintaining a conversion accuracy of ±0.05% of full scale input.

Final selection of the multiplexer and AD converter await the detailed design phase. However the favored components are the 845A multiplexer and 834 AD converter, both manufactured by Texas Instruments.

The 845A multiplexer specifications are:

Input levels:	±10v max.
Overvoltage Tolerance:	±50v
Accuracy:	±0.02% full scale
Linearity:	less than ±0.01% of full scale departure from best straight line.
Settling Time:	less than 8 microseconds (with 834 AD converter)
Offset:	less than ±0.25mv at 25°C ±5°C
Source Impedance:	Recommended maximum is 1000 ohms
Maximum Sampling Rate:	50,000 channels per second
Maximum Number of Channels:	160
Channel Selection:	Addressable, binary

The 834 AD Converter is a successive bit approximation converter whose specifications are:

Conversion period:	22.5 microseconds
Accuracy:	±0.05% of full scale ±1/2 least significant bit
Input Impedance:	100,000 ohms shunted with 50 picofarads
Sample Period:	3.0 microseconds
Automatic zero stabilization period:	1.5 microseconds
Sample time uncertainty: (Aperture time)	0.1 microsecond
Time per bit decision:	1.5 microseconds (1.2 microseconds special)

One of the features of the 834 converter is that sample-and-hold circuitry is incorporated which reduces the sample time uncertainty to 100 nanoseconds. The combination of the 845A multiplexer and 834 AD converter permit a conversion rate in excess of 35,000 channels per second to be realized in the input/output system. The combination permits the multiplexer settling time to overlap the sample time such that only 27.5 microseconds are required for both multiplexing and converting.

C. ANALOG OUTPUT

The analog output will take the form of 64 DA converters rather than one or two DA converters time shared via a demultiplexer and 64 sample and hold circuits. The individual DA converters offer the operational advantage of a time-independent zero order memory for each output operand permitting indefinite retention of the analog voltage without requiring updating and, in addition, are less expensive than the corresponding sample and hold circuits.

Two levels of resolution are planned; a large number of lower resolution converters with about ±0.1% full scale accuracy and resolution of at least 9 bits plus sign, together with a few high accuracy, high resolution converters with at least ±0.05% accuracy and resolution of 11 to 13 bits plus sign. The high accuracy channels are required for driving high resolution indicators contemplated for future aerospace vehicles, for communicating with analog computation elements, and for providing precision analog signals for calibration of the analog input/output channels. The 64 DA converter channels are to be composed of 8 fourteen bit units and 56 ten bit units.

The use of high resolution, high accuracy DA converters in the general purpose simulation system stems from three anticipated requirements:

1. Providing high resolution analog signals to extended scale length indicators contemplated for use in future manned aerospace vehicle cockpits,

2. Providing an accurate voltage reference source for calibrating the analog portion of the simulation facility, and

3. Providing accurate analog voltages to state-of-the-art analog computation components.

The first requirement hinges on the desire to display a quantity in analog form (i.e., the displacement of a pointer along a scale) with sufficient realism so the pilot in the simulator is not aware of the digital nature of the signal driving source. A simple experiment employing a DA converter and multimeter has shown that discrete meter movements of as small as 0.003 inch can be detected if the movement takes place in a short period of time and if the observer is intent upon detecting such movement. A lower resolution limit of 0.01 inch per quantum step appears to be practical. With this resolution goal applied to the 180 inch scale length tape indicator contemplated for future aerospace vehicles, it is found that a resolution of 18,000 to 1 is necessary. The current state of the art is the 16,384 to 1 resolution obtainable with 14 bit DA converters. Thus it appears that 14 bit converters are desirable in the research system to accommodate future developments in indicators and be able to simulate continuously varying functions.

A second justification for 14 bit DA converters stems from a system maintenance consideration of having a precision voltage source available for calibration purposes. Such a converter could supply a voltage accurate to $\pm 0.01\%$ of the full scale analog voltage range, which is more accurate than the remaining components of the system and which may be used as an easily programmed source of variable calibration voltages.

The third requirement for 14 bit DA converters is to match the accuracy of analog computing components which may be employed in a hybrid simulation facility. The accuracy of the 14 bit DA converter is equivalent to that of such state-of-the-art analog computing components as operational amplifiers.

The requirement for the low resolution DA converters is based upon their most likely use as panel indicator driving sources. Typical cockpit panel meter scale lengths are approximately 10 inches in length. This is coupled with the desire to present continuously varying functions rather than the discrete quantizations arising from the digital computation. This implies the incremental change in a meter indication due to the smallest quantum change in the digital source value be small on the order of 0.01 inch to be on the threshold of detection of the quantum change. The trade-off between cost and resolution is important here since the large number of DA converters required represents a major portion of the input/output system. A ten-bit resolution is recommended as the minimum to insure a 0.01 inch maximum scale increment on a 10 inch panel meter scale. The cost curve steepens considerably above 10 bits as shown in Figure 4.

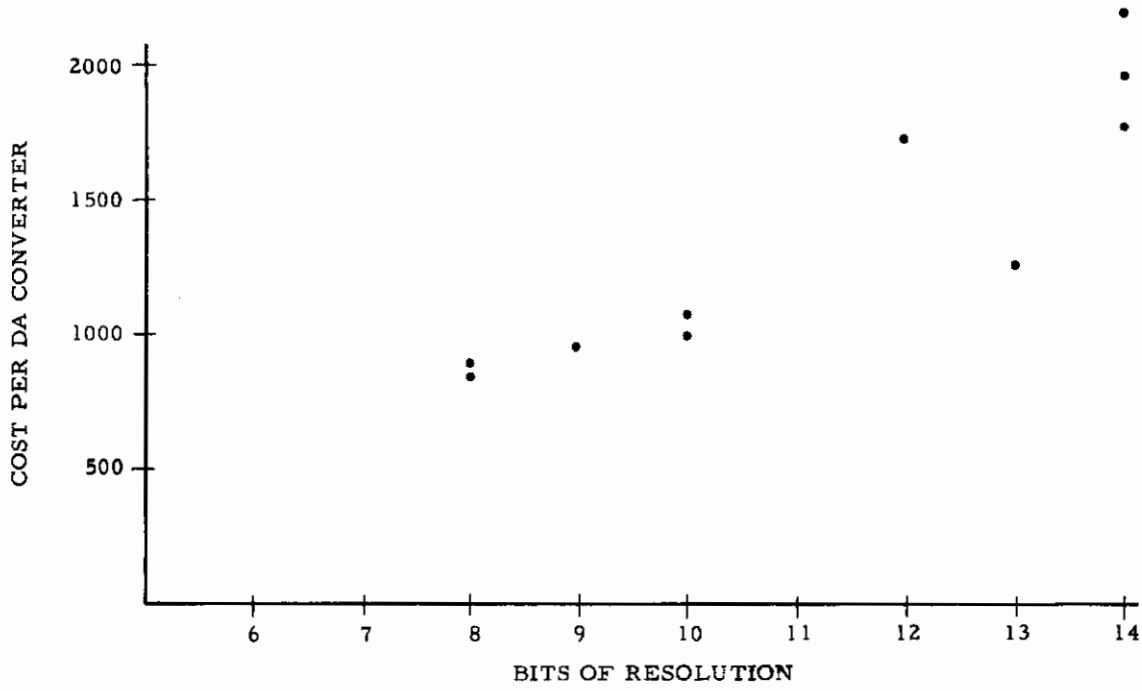


Figure 4 DA Converter Cost vs Resolution

D. DISCRETE INPUTS AND OUTPUTS

The discrete inputs or sense lines will be divided into 3 groups of 24 lines each, totaling 72 inputs. All lines of a given group will be sampled simultaneously and the 24 sensed conditions will be processed into the computer in parallel. The determination of the status of a given sense line in the group will be done in the computer by logical operations, shifting, and testing. The discrete input subsystem is characterized by simplicity, economy, and high input rate capability.

The discrete outputs will be handled in a manner similar to the discrete inputs in that the outputs are divided into 3 groups of 24 lines. Each output line will contain a flip-flop to retain its true or false condition indefinitely if desired. Each group of 24 flip-flops is updated in one parallel 24 bit transfer from the computer. A 'zero' in a given bit position will clear the corresponding flip-flop whereas a 'one' will set the flip-flop. The drive capability to be provided by the output lines will be determined in the detailed design phase.

E. DIGITAL CLOCK

The digital clock is in the form of a program setable interval timer. Referring to Figure 5 it is seen that two flip-flop registers, a clock pulse source, data transfer gates, and suitable control functions constitute the interval timer.

The desired interval is loaded into the Time Interval Register (TIR) by the program using the command and data word output facility of the computer. First a command word is sent to the I/O controller to alert it to the fact that the following data output word is the new operand to be inserted into the TIR. Then the data word is transmitted to the TIR. Suitable lockout circuitry is provided to prevent the TIR from accepting the new information during that short period when the content of TIR is required for resetting the Clock Counter Register.

The CCR is a decrementing counter which initially is loaded from the TIR. Periodic clock pulses issuing from the pulse source decrement the count in CCR one count per pulse until zero is reached. Upon reaching the count of zero, a pulse is issued: (1) to the computer to cause an automatic interrupt to occur and (2) to transfer the content of TIR to CCR, permitting the next interval to begin timing out automatically.

The pulse source is a crystal controlled oscillator, producing the clock pulse at a 10kc rate (100 microsecond period). The frequency stability and accuracy of the clock pulse will be better than 0.01%. Extremely high accuracy timing pulse sources are available but are not felt warranted in the application since in flight simulation, time errors of less than 1% are difficult to detect by the man in the loop unless he has an independent clock of greater accuracy to check against. In the event the digital computer is used with analog integrators, a 0.01% time base appears adequate in comparison with typical integrator errors which are greater than 0.05%.

The TIR and CCR both have 14 bit stages, permitting intervals to be set by the programmer in the range from 100 microseconds to 1.6 seconds in increments of 100 microseconds. Interruptions more often than every 100 microseconds definitely waste computer time by using an excessive proportion of the time to process interrupts, thus the lower range on the interval is adequately low. The upper limit may not be long enough for some applications, however the programmer has the choice of using multiple interrupt periods to time an interval of arbitrary duration simply by counting interrupts in his interrupt processing routine.

Read out of the current content of CCR is a facility provided to permit the programmer to ascertain the time remaining in the interval. To request the content of CCR, the appropriate command word is sent to the I/O controller requesting the operand currently in the CCR. Providing that the decrementing pulse is not then currently causing CCR to change, the content of CCR is gated onto the computer data input bus. Otherwise, a short delay will occur, permitting the flip-flops in CCR to stabilize, to prevent erroneous data readout.

The added control facility of turning the clock on and off is provided to the programmer. This is accomplished by sending a command word to the I/O controller causing a toggle (flip-flop) to switch to the proper state enabling or inhibiting the clock pulses as desired. This facility is inexpensive to provide and enables removal of the periodic interrupts when they are not desired, say for non-simulation programs such as assemblies or for terminal operations in simulation runs.

One feature of the interval timer that is an advantage over the type of clock used in UDOFT is that, when enabled, the timer continues to run and does not stop when the interval is over. Thus time cannot be lost by failure to interrogate the clock sufficiently often, as was the case with UDOFT.

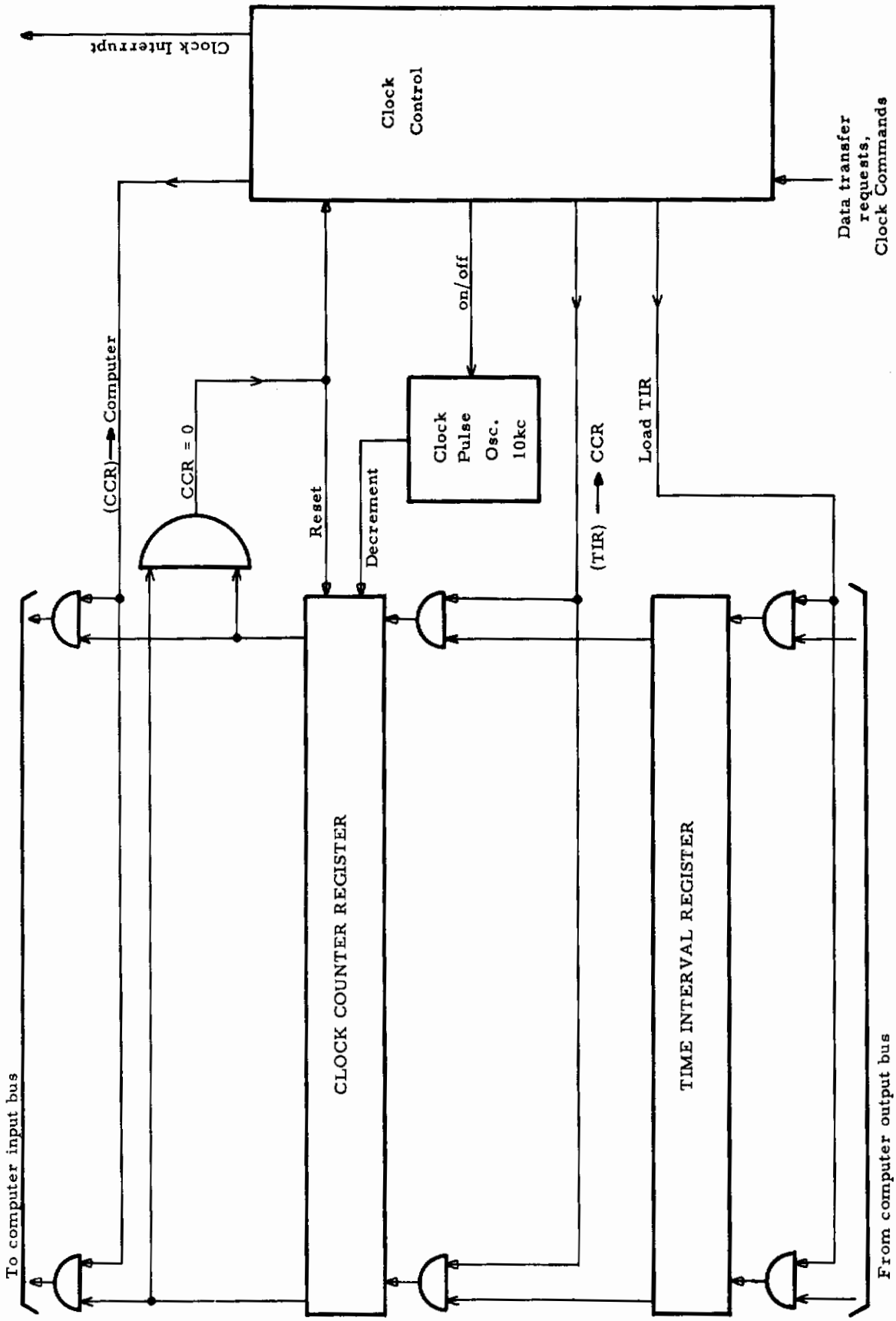


Figure 5 Digital Clock Interval Timer

SECTION IV

SUMMARY AND CONCLUSIONS

A. SUMMARY

The Study of Digital Computers for a Real Time Training Simulation Research System has surveyed the available and soon-to-be available general purpose digital computers together with analog input/output equipments for implementing a proposed simulation research laboratory facility. The objective of the study was to select the digital computer to be employed in the laboratory and to define the input/output system.

1. Computer Study

The computer study reviewed the computer models available and matched their characteristics against the minimum specifications required for digital control of flight simulators. The computers selected for close scutiny fell into two catagories:

1. Those meeting the cost objectives and minimum operating rate requirement,

2. Those exceeding the cost objectives, but possessing a significant operating rate capability in excess of the minimum requirement, to provide an excellent laboratory tool which is capable of simulating aerospace vehicle performances which surpass that of the manned supersonic fighter of the F100A class.

There were two computers which fell into catagory 1 which were examined closely; the Computer Control Corporation's DDP24 and the Packard Bell PB440. The one computer which fell into catagory 2, which was considered in the study is the Scientific Data System's 9300.

In the case of all three computers, the prototype of the model did not exist as a completed system in the period of the study. It is felt that this situation is typical of any digital computer evaluation study due to continual lowering of the cost per computation as each new model is marketed. As each of the several manufacturer's strive to better their competitors, newer, faster, and more economical computers will appear. This places a burden on such an evaluation as was conducted in this program, namely that delivery date schedules are risked in an effort to select the most modern equipment.

2. Input/Output System

The Input/Output System definition consisted of matching the commercially available digital, AD and DA converter equipment to the equipment budget based on a requirement for the system to handle a simulation of at least the size of theF100A simulation. The system requirements were conventional in light of typical hybrid computation facilities except perhaps in the number (64) of DA converters required. The DA converter subsystem was the major cost item of the Input/Output System and attention was given to means for reducing the cost in this area. The technique of employing a single time-shared DA converter with multiple sample-and-hold circuits was examined and found to offer no economical benefit. This approach was abandoned in favor of multiple DA converters with the attendant operational advantage of long term memory in each channel, not requiring continual updating as the sample-and-hold technique does. A subgrouping of the DA converters into low and high resolution catagories was found to offer considerable reduction in equipment cost yet retain a high accuracy capability for a reasonable number of channels. The high accuracy channels were found desirable for three reasons:

1. To provide realistic simulated analog variable presentation to extended scale length indicators;

2. To permit a hybrid computation to employ the system equipment with minimum degradation in the accuracy of the signal being converted from the digital to the analog domain, as in contemporary practice; and

3. To provide accurate sources of calibration voltages for checkout of the system.

The remaining Input/Output System components consist of a 32 channel multiplexer and AD converter, a discrete input capable of sensing the binary state of as many as 72 variables, 72 discrete, or boolean, outputs, and a program-controlled interval timer capable of interrupting the computer.

B. CONCLUSIONS

1. Computer Study

Of the two lower cost computers, the PB440 was found to have a narrow edge over the DDP24 for the intended simulation application. The advantages of the PB440 over the DDP24 are:

1. A higher computational rate potential which is all-important in real time simulation,
2. Built-in floating point instruction provisions,
3. A versatile and easily changeable instruction repertoire which permits the user to more fully optimize his program, and
4. A variable precision arithmetic capability due to micro-programming control of the length of multiplies and divides.

A further advantage of the PB440 in the simulation laboratory application is its relatively greater capability of simulating other digital computers with an appropriate inclusion of the simulated computer's repertoire. This advantage is important in that the laboratory may be called upon to evaluate other digital computers, by simulation, when such computers are employed in aerospace vehicle simulation.

The advantages of the DDP24 over the PB440 are:

1. A lower cost,
2. More certain delivery since a predessor model, the DDP19, has previously been developed and delivered,
3. Ease of programming and software documentation due to a straightforward conventional machine organization and one fixed instruction repertoire.

The SDS 9300 computer was found to be about three times faster than the lower cost computers in the F100A simulation as programmed for UDOFT. In future aerospace simulations where floating point instructions may be employed more extensively, the 9300 may be five to eight times faster than the PB440 or DDP24 computers. The high operating rate of the 9300 would be very advantageous in the proposed laboratory facility since it could cope with a wide variety of aerospace vehicle simulations.

2. Input/Output System

The 32 channels of AD conversion may be handled by a multiplexer and single AD converter. The Texas Instrument multiplexer and AD converter are suitable for the simulation facility by providing a high digitizing rate to reduce time skew between channels yet retaining an adequate accuracy.

The 64 DA conversion channels may be suitably implemented with 64 conventional DA converters. The cost of this number of DA converters dictates that a large portion of them be of a resolution and accuracy which permits costs to be kept down yet provide sufficient capability in the OFT simulation application. The DA converter configuration to meet the objectives is one which contains a few, say eight, 14 bit 0.01% units and a larger portion, or fifty-six 10 bit 0.1% units.

Discrete inputs and outputs total 72 each and are easily implemented in the system. The number of discrete inputs and outputs appears sufficiently high to accommodate simulations of the F100A scale and even allow for more complex system simulations. The figure of 72 is selected to fit the 24 bit computer word in three groupings and provide discrete input and outputs in excess of the number used in the UDOFT F100A simulation facility.

The interval timer may take the form of a program setable time interval register, a counter, and a crystal oscillator. The timer detailed in the body of the report will provide interrupts at intervals of from 100 microseconds to 1.6 seconds under program control and to an accuracy of 0.01%. The computer may access the counter register for time information between intervals, thus the interval timer may serve as a real time system clock device.

C. RECOMMENDATION OF COMPUTER SYSTEM

Inasmuch as a very capable, high speed computer will be available in the near future, namely the SDS 9300, although at a cost exceeding the initial objectives, it is felt that alternative recommendations are advisable; one for a lower cost computer and the second for the higher cost computer.

The PB440 computer system manufactured by Packard Bell Computer Corporation is alternative system A. The basic PB440 computer includes 4096 words of main memory and 256 words of logic memory.

The manufacturer's optional features which are recommended for inclusion in the PB440 configuration are:

1. Paper Tape Punch
2. Paper Tape Reader
3. Input/Output Typewriter
4. One additional 4096 word main memory module
5. One additional 256 word logic memory module
6. Spare parts kit.

The SDS 9300 computer system, manufactured by Scientific Data Systems, Inc., is recommended as alternative system B. The basic 9300 includes 4096 words of memory, paper tape punch, paper tape reader, input/output typewriter, and three index registers. Optional equipment recommended for the 9300 computer consists of one additional 4096 word memory module and a spare parts kit.

A summary-comparison of the alternative systems is given below:

TABLE V SYSTEM CHARACTERISTICS

Characteristic	System A PB440	System B SDS 9300
1. Paper Tape Reader	500 cps bidirectional	300 cps
2. Paper Tape Punch	110 cps	60 cps
3. In/Out Typewriter	15 cps Selectric	15 cps
4. Memory size	8,192 words	8,192 words
5. Memory access time	2 μSec	0.7 μSec
6. Memory cycle time	5 μSec	1.75 μSec
7. Number of Index Registers	7	3
8. Delivery	Jan. 1964 (fairly certain)	April 1964 (tentative earliest)
9. Cost, including spares	$190,000	$275,000
10. Average instruction execution time (F100 simulation)	13.26 μSec	4.1 μSec

Considering the two alternatives, we want to point out that the SDS 9300 system has many more unknowns in it than the PB440. An important item for consideration is that the SDS 9300 delivery date could be delayed due to problems in their 1.75 microsecond memory development and the initial production schedule of only 1 SDS 9300 per month cannot provide rapid recovery from delays.

The higher cost of the SDS 9300 is a definite hindrance in its selection. The possibility of reducing the requirements on memory capacity and input/output capability in order to meet the cost objectives exists. By reducing the memory capacity to 4096 words and eliminating 56 of the 64 converter channels, a system incorporating the SDS 9300 could be provided at no cost differential over the proposed PB440 system. This reduced capability system could not then be

utilized for any significant flight simulation computation due: (1) to the lack of memory storage for instructions and operands, and (2) the serious lack of analog outputs to the cockpit displays. We recommend that such an emasculated system not be considered. It is to be noted that the extra cost floating point instruction option is not included in the recommended SDS 9300 system. We feel that the additional $45,000. for the floating point hardware places the system cost at a level beyond considering in view of possible funding problems, however the floating point option would probably pay off in the long term benefits gained by ease of programming and faster operation in those routines requiring floating point arithmetic.

REFERENCES

1. Connelly, Mark E., "Real-Time Analog-Digital Computation," I.R.E. Transactions on Electronic Computers, p. 31, 1962.

2. Design of Digital Flight Trainers, Moore School Rpt. 54-09, University of Pennsylvania, Philadelphia, September 1953.

3. Digital Operational Flight Trainers, Moore School Rpt. 54-05, University of Pennsylvania, Philadelphia, December 1953.

4. Feasibility of Actuating Trainers by Digital Computers, Moore School Rpt. 53-06, University of Pennsylvania, Philadelphia, September 1962.

5. Krasny, Louis M., The Functional Design of a Special-Purpose Digital Computer for Real-Time Flight Simulation, Technical Documentary Report No. MRL-TDR-62-39, Behavioral Sciences Laboratory, 6570th Aerospace Medical Research Laboratories, Wright-Patterson Air Force Base, Ohio, April 1962.

6. Perry, Edward L., Submicrosecond Simulation Computer Study Program, Parts 1 and 2, Technical Documentary Report No. MRL-TDR-62-27, Behavioral Sciences Laboratory, 6570th Aerospace Medical Research Laboratories, Wright-Patterson Air Force Base, Ohio, May 1962 and October 1962.

7. Programming Manual for the UDOFT Computer, Sylvania Electronic Systems, Needham, Mass., August 1959.

8. UDOFT Simulation Program, Final Report FR 77-IN, Sylvania Electronic Systems, Needham, Mass., May 1960.

9. UDOFT Summary Report, Sylvania Electric Products, Inc., Needham, Mass., December 1962.

10. DDP24: General Purpose Digital Computer, Computer Control Company, Inc., Framingham, Mass., 1963.

11. DDP24: Reference Manual, Computer Control Company, Inc., Framingham, Mass., 1963.

12. DDP24: DAP Manual, Computer Control Company, Inc., Framingham, Mass., 1963.

13. DDP24: FORTRAN II Manual, Computer Control Company, Inc., Framingham, Mass., 1963.

14. Authorized Federal Supply Schedule Price List, Control Data Corporation, Minneapolis, Minn., 1962.

15. Control Data 924 Computer Reference Manual, Control Data Corporation, Minneapolis, Minn., November 1962.

16. An Introduction to the PB440 General Command Set, SP-159, Packard Bell Computer, Los Angeles, Calif., April 1963.

17. An Introduction to the PB440 FORTRAN Programming System, SP-158, Packard Bell Computer, Los Angeles, Calif., April 1963.

18. An Introduction to PB440 Microprogramming, Packard Bell Computer, Los Angeles, Calif., May 1963.

19. SDS DES-1 Preliminary Specification, Scientific Data Systems, Inc., Santa Monica, Calif.

20. SDS 9300 Technical Introduction, Scientific Data Systems, Santa Monica, Calif., July 1, 1963.

21. Grabbe, Ramo, Wooldridge, Handbook of Automation Computation and Control, Wiley, 1959 Vol. 2, pp. 12-18.

APPENDICES

APPENDIX I

PB440 MICRO-INSTRUCTION REPERTOIRE

The information in this appendix has been taken from the printer's proof of the Packard Bell document entitled "An Introduction to the PB440 Computer." The information, although altered in minor details of order and format from the PB document, contains the essential detailed information to code microroutines for the PB440.

A. THE PB440 MICRO-INSTRUCTION SEQUENCE

The registers available in the PB440 Computer are shown in Figure I-1. For the moment, we need consider only the registers labeled "P" and "E".

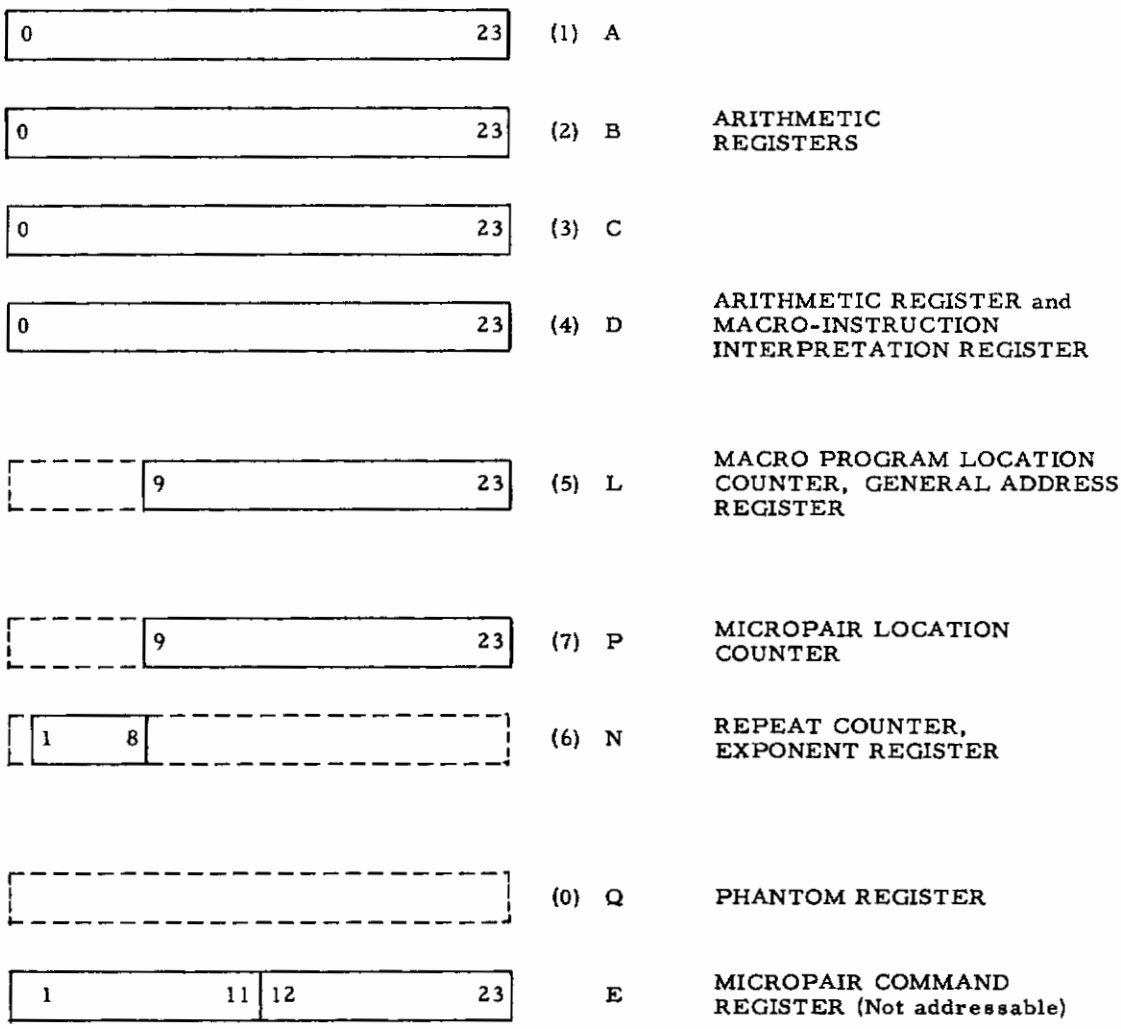


Figure I-1 PB440 Registers

The individual micro-steps are stored in memory in pairs (see Figure I-2) and are executed in sequence from left to right. The "P" register functions as a micro-pair location counter, and normally contains the address of the next micro-pair to be executed. It is automatically incremented by 1 each time a micro-pair is obtained from memory for execution. Micro-steps are stored in pairs since the two 12-bit micro-steps fit into the 24 bit word. The left micro is executed first.

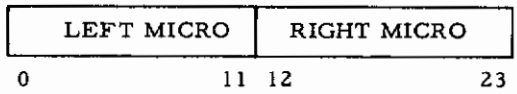


Figure I-2 Micro-pair storage

"E" register holds the pair of micro-steps being executed, and may be thought of as the micro-level command register. The following diagrams illustrate the sequence in which operations are performed in the PB440. Each numbered column refers to one machine cycle, which requires one microsecond. The labelled lines refer to aspects of the operation; when the line is up, that particular operation is execution.

1. The Normal Sequence

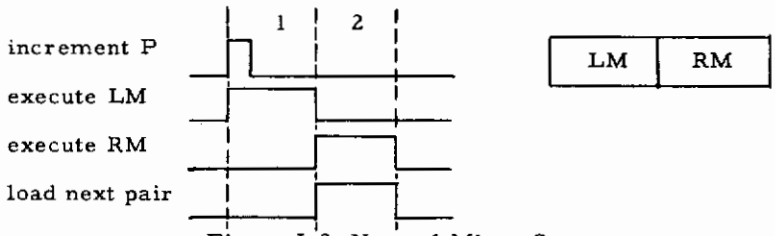


Figure I-3 Normal Micro Sequence

This is the sequence followed most of the time when micro-pairs are being executed out of fast memory. At the beginning of the first cycle the conditions are that

1) The micro-pair whose left half is denoted LM and whose right half is denoted RM is in the E register.

2) The P register contains the address of the location of this micro-pair in memory.

At this point the following operations take place;

Cycle 1

1) The P register is incremented by one. This operation is drawn as requiring less than one machine cycle. It is to be understood as logically preceding the other operations which occur in this cycle.

2) The micro-step indicated as LM is executed. At the same time, the right hand micro-step is examined to see whether it involves a memory access or a change in the P register. If it does not, the operations proceed as shown in the normal sequence. If it does, the sequence is not normal, and is discussed later.

Cycle 2

1) The micro-step indicated as RM is executed.

2) The contents of the memory location whose address is in the P register are fetched from memory and at the end of cycle 2 are inserted into the E register.

It may now be verified that conditions are exactly the same as they were at the beginning of cycle 1, except that the micro-pair concerned is the one following the original one in memory. Hence the cycle begins again. Normal timing for this execution sequence, assuming the next pair is located in fast memory, is 2 clock pulses. To a first approximation one can say that execution proceeds at a rate of 1 micro-step per clock pulse.

The above sequence may be altered somewhat depending on the nature of the operations LM and RM. In particular it is changed when these operations cause

1) branching,

2) testing,

3) main memory accesses, or

4) multi-step operations.

2. Branching Sequences

It is seen that in the above sequence the P register is used to locate the next word to be placed in the E register. Hence alteration of the normal one-after-the-other flow of instructions is accomplished by changing the contents of the P register. When the right-hand micro-step changes the content of the "P" register, however, the proper micro-pair cannot be obtained from memory until this operation is completed. This condition requires that the computer wait for one additional clock pulse, so that 3 clock pulses are required to execute the micro-pair.

In particular, suppose the micro-step LM in the preceeding discussion had been "CLP 000 001." Then during part 2 of cycle 1 the contents of the P register are replaced by the address 7001_8 (the first location in fast memory). And hence in part 2 of cycle 2 the word loaded into the E register is not the word following the instruction presently being executed, but rather the first word in fast memory, thus causing a program branch to this word.

However, suppose the "CLP 000 001" instruction had been in the right-hand half of the word. In this case the examination of the right micro-step described in part 2 of cycle 1 above would have caused the loading of the next micro-pair to be held up until after part 1 of cycle 2 had been completed. The sequence is shown in figure I-4 below.

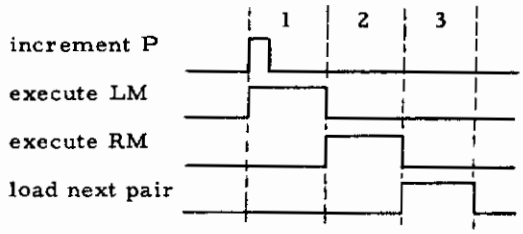


Figure I-4 Micro Branching Sequence

Cycle 1

1) The P register is incremented by 1.

2) Operation LM is performed. At the same time, since the right hand micro-step alters the P register, provision is made to hold up the load-next-pair operation until RM has been executed.

Cycle 2

1) Operation RM "CLP 000 001" in this case, is performed, i.e., the address of the first location in fast memory is loaded into the P register.

Cycle 3

1) The word at the location specified by the P register is loaded into the E register.

Thus it is apparent that when the right hand member of a micro-pair changes the P register the micro-pair effectively requires 3 microseconds to execute rather than 2.

3. Testing Sequences

Other alterations in the normal internal sequence appear with the use of the test micro-steps. These micro-steps all function in the following manner. Their titles describe a certain condition, i.e., Test Non-zero, or Test Mode True. If the condition described in the title is in fact the case, i.e., the specified bits are non-zero, or the mode bits are in the true state, then the condition is said to be met, and succeeding instructions are executed as written. If the condition described is not the case, then

1) if the test is a left micro-step, the associated right micro-step is not executed, and the next micro-step to be executed is the left half of the succeeding micro-pair.

2) if the test is a right micro-step, the P register is incremented before the next micro-pair is extracted from memory. Normally this means the next micro-pair is skipped.

The above operations are shown in detail below.

1. Left micro-step, condition met.

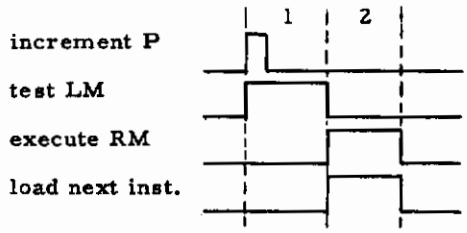


Figure I-5 Left Micro Test-Skip

Figure I-5 shows the execution sequence of a micro-pair whose left half is a test micro when the condition is met. This is seen to be exactly the same as a normal execution sequence.

2. Left micro-step, condition not met.

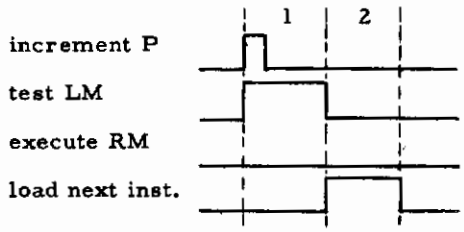


Figure I-6 Left Micro Test-Non-Skip

Figure I-6 shows the execution sequence of a micro-pair whose left half is a test micro when the condition is not met. This is seen to be exactly the same as a normal execution sequence, except that the execution of the micro-step RM is deleted.

3. Right Micro-step, condition met.

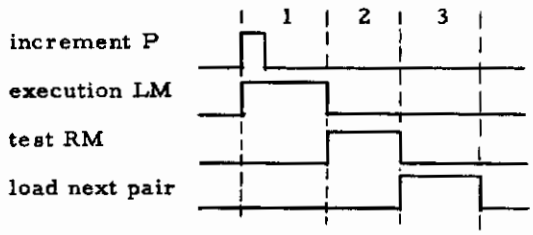


Figure I-7 Right Micro Test-Skip

Figure I-7 shows the execution sequence of a micro-pair whose right half is a test micro whose condition is met. This is logically equivalent to a normal execution sequence, but requires one microsecond longer, since the next pair cannot be loaded into the E register until the outcome of the test is determined. Hence the RM micro-step execution and the loading of the next pair cannot take place simultaneously as in the normal sequence.

4. Right Micro-step, condition not met.

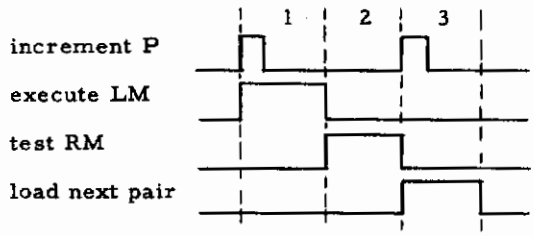


Figure I-8 Right Micro Test-Non-Skip

Figure I-8 shows the execution sequence of a micro-pair whose right half is a test micro whose condition is not met. It is seen to be the same as 3 above, except that the P register is incremented in cycle 3 as well as in cycle 1. This has the effect of causing the next micro-pair to be skipped, since it is the one which follows which is loaded into the E register.

Thus it is seen that in general a test micro requires 1 microsecond when placed in the left half of a micro-pair, and 2 when placed in the right half. Hence, other things being equal, it is best to keep tests in the left half of a word wherever possible.

One restriction is inherent in the micro-pair structure. Since the "P" register contains the location of each pair, it is possible to transfer control only to a left-half micro-step. While this restriction might indicate that many "no-op" micro-steps would be required, programmer ingenuity can keep them to a minimum.

4. Memory Access Sequences

The normal execution sequence is also altered when the loading of the next micro-pair is delayed by the right micro-step making a memory access. This is due to the memory-bus structure which allows either fast or main memory to be referenced during any clock pulse, but not both at the same time. Therefore, if the right-hand micro-step is one which references either main or fast memory, the next micro-pair cannot be obtained at the same time, and the computer must wait for 1 additional clock pulse after the memory access. There is no compounding of waiting periods for both changing "P" and accessing memory in a right-hand micro-step, however. The memory access sequence by a right micro-step is illustrated below.

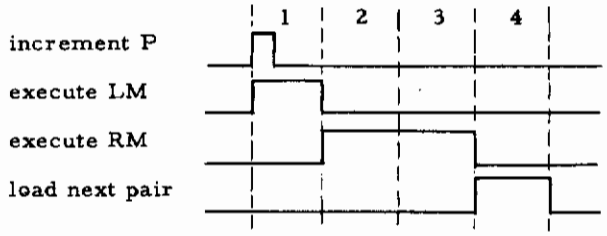


Figure I-9 Memory Access Sequence

Figure I-9 shows the execution sequence of a micro-pair whose right micro-step accesses main memory. Cycles 2 and 3 are used to execute the memory access, giving a normal access time of 2 microseconds. However, since the next pair could not be extracted from memory during this time without a memory bus conflict, an extra cycle is needed to load the next instruction. Thus any memory access requires 1 more microsecond than usual when executed from the right half of a micro-pair.

5. Multi-step Sequences

Several micro-steps, such as MPS, DVS, SDL, etc. require several machine cycles for their execution, the number depending usually on the contents of the N register. In executing these instructions the control sequence is simply altered by expanding the appropriate execution cycle to the necessary number of cycles.

B. THE REGISTERS

The working registers are shown in diagram form in Figure I-1. All of the registers shown are directly addressable with the exception of E, the micro-pair command register. The dotted lines indicate portions of a register which are not actually present in hardware; reference to such a region has the same effect as if the register contained all zeroes in those bit positions. The Q register is all empty; it may be considered as a register permanently containing zeroes.

The working registers consist of the A, B, C, and D registers of 24 bits each, the N register consisting of 8 bits, the L register (15 bits in length), three carry toggles, 6 "program flags" (which are addressable 1 bit registers), and a parity toggle. Not shown in Figure I-1 are those elements which are available on the computer console (6 sense-switch toggles, address switches representing a 15 bit address, and another set of switches representing a full 24 bit word). These elements can be referenced by the programmer (as can various input-output elements and devices) but are not considered in detail in this discussion of the registers.

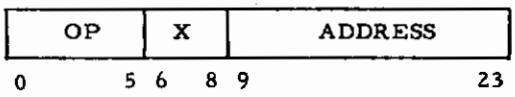
In general, it is expected that most arithmetic and logical operations will be performed utilizing the A, B, C, and D registers. It is expected that the L register will be used as a macro-instruction location counter, though it should be emphasized that the design in no way restricts the use of any register (except, of course, for "P" and "E") to any particular function. Several micro-step operations reference specific registers, but, for the most part, the programmer is free to decide the function of the registers for himself.

The carry toggles are set by certain micro-steps automatically, and are designed to indicate carry-out from bit positions 0, 1, and 9. That is, a carry toggle is made true (or "1") if carry-out occurs from its bit position, or made false if no carry-out occurs. They may be set and tested individually as well. The 6 program flags may be set and tested at the discretion of the programmer, and their content, as well as the content of any other register, may be displayed on the console. The parity toggle contains the parity of the last executed memory access micro-step, and it may be referenced by the programmer.

The N register serves as a repeat counter for those micro-steps which are of a repetitive nature, such as multiplication, division, and shifting operations. It may also be used for other purposes when desired.

Figure II-1 (Appendix II) shows various word formats in diagram form. Various fields in the word have been labeled so that they may be referred to during the description of the individual micro-steps. It will be noted that the N register is located so that it can hold the "exponent" field of a floating-point format word. Also, the "fraction" portion of a floating-point format word consists of 15 bits and can also contain an address for reference to the memory system of the computer.

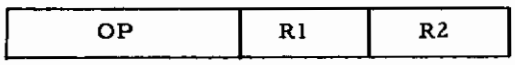
Several micro-steps reference only the D register which can be used to good effect as a macro-instruction command register. One of the possible macro-instruction formats used in conjunction with the D register is that employed in the Systems Command Set and is shown below:



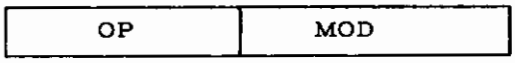
Any register may be shifted, but the N register should be shifted only after careful consideration of the net effect.

C. THE MICRO-STEPS

Figure I-10 shows two possible formats for a micro-step. While many micro-steps utilize both the R1 and R2 fields to designate one of the registers, not all of them do so, and these 3 bit fields may have special meaning for certain of the micro-steps. In some cases the full 6 bit field, exclusive of the operation field, is used as a unit and reference to the register (or registers) is implied by the operation field.



Some micro-steps employ the right-hand 6-bit field as two 3-bit designators which may be used to address registers.



Other micro-steps employ the 6-bit field as a modulo 64 modifier.

Figure I-10 Micro-step Formats

In general, where the R1 and R2 fields are used to designate two registers, both fields may designate the same register. In some cases it is not meaningful to do this since no change results, but it is always permitted. For convenience, the individual micro-steps have been grouped in the descriptions below.

1. Memory Access Operations

Memory accesses may refer to either main memory or to fast memory. "Load" micro-steps retrieve a full 24 bit word from memory, and "Store" operations place the full 24 bit word into the designated memory cell. Where a register is less than 24 bits in length the "missing" portion will always store zeroes into memory in those bit positions.

When main memory is referenced for a load operation, 2 clock pulses are required to read the content of the memory cell into the designated register. Following the main memory access, 3 clock pulses are required to establish (or re-establish) the content of the memory cell. This time may be used for useful computing, and is referred to as "shadow time". Should a reference to main memory be made during this three clock-pulse interval, the computer will wait until the previous cycle is completed.

When main memory is referenced for a store operation, the computer is immediately ready for useful computing. However the full memory cycle time must elapse before another memory reference can be made. Therefore since the store operation requires a single clock pulse, the "shadow time" is 4 clock pulses, and if another main memory reference is made during this time the computer will wait until the previous cycle is completed. It is the programmer's responsibility to utilize the "shadow time" effectively if he can. Since fast memory read-out is non-destructive, no "shadow time" results from a fast memory access. The amount of time lost in storing into fast memory (no computing is possible until such an operation is completed) is 8 microseconds. This operation should probably be avoided if maximum speed is desired.

Each individual module of main memory has its own read-write circuitry, so that reference to module 2 may, if desired, be made during the "shadow time" resulting from a reference to module 1, with no penalty in execution speed.

In the micro-steps described below, the R1 field designates a register which contains an address. The low order 15 bits of this register (bit positions 9-23) are used as the address; the high-order bits are ignored. The carry toggles are not affected unless specifically stated in the description. The parity toggle is set to 1 if the number of 1's in the word is even, or to 0 if the number of 1's is odd.

LDM R1 R2 (VI)	Load from Memory	The content of the memory cell, designated by the address in register R1, replaces the content of the register designated by R2.
STM R1 R2 (VII)	Store into Memory	The content of the register designated by R2 is stored into the memory cell designated by the address in R1. The content of R1 and R2 remains unchanged.
LDI R1 R2 (VI)	Load (from memory) and Increment	The content of the addressed memory cell replaces the content of the register designated by R2. Simultaneously, the address in register R1 is incremented by 1 in bit position 23. This latter operation does not change the setting of the carry toggles. When R1 and R2 designate the same register the content of the addressed memory cell and the incremented value of the designated register are combined (logical or), and the logical sum replaces the content of that register.
STI R1 R2 (VII)	Store and Increment	The content of the register designated by R2 is stored into the addressed memory cell. Simultaneously, the address in R1 is incremented by 1 in bit position 23. This latter operation does not change the setting of the carry toggles. If R1 and R2 designate the same register, the memory cell will contain its own address and R1 will contain the logical sum of the address and its incremented value at the end of the operation.

It is clear that the above micro-steps can refer to any memory cell in the computer, but an address for that cell must be present in one of the registers. (Designating Q or N by the R1 field can refer only to main memory location 00000. This should probably be avoided.) Since many common operations involve the requirement that the content of some register be stored temporarily, a group of 8 cells in main memory has been designated as "working storage". These may be referred to directly, without the requirement that an address be provided. The following two micro-steps reference these cells. The R1 field designates which one of the 8 cells is to be used, and does not designate a register. (See Figure I-11)

LDW R1 R2 or LDW n R2 (VI)	Load from Working Storage	The content of the working storage cell designated by R1 replaces the content of the register designated by R2. The content of the designated memory cell remains unchanged.
STW R1 R2 or STW n R2 (VII)	Store into Working Storage	The content of the register designated by R2 is stored into the designated working storage cell. The content of the register designated by R2 remains unchanged.

2. LDS and STS -- Load Special and Store Special

In addition to the above, two special micro-steps have been provided which reference certain fixed locations in memory. In these micro-steps the R1 field is used as a control field to designate which one of eight possible load or store operations is desired. In all of the operations, the effective address of the cell is constructed from a designated field of the D register.

The first portion of main memory is divided into blocks of 64 words each, numbered, for reference, blocks 0-3. Seven of the 8 possible configurations of R1 reference this portion of memory while the eighth configuration references fast memory according to the following scheme:

R1	Portion of D Used	Memory Block Referenced
0	Bits 6-8 (mode-field)	Block 1, cells 00-07 (Locations 64-71)
1	Bits 18-23	Block 1, cells 00-63 (Locations 64-127)
2	Bits 18-23	Block 2, cells 00-63 (Locations 128-191)
3	Bits 18-23	Block 3, cells 00-63 (Locations 192-225)
4	Bits 0-5	Block 0, cells 00-63 (Locations 00-63)
5	Bits 0-5	Block 1, cells 00-63 (Locations 64-127)
6	Bits 0-5	Block 2, cells 00-63 (Locations 128-191)
7	Bits 0-5	Fast memory cells 28864 to 28926 (last 64 cells in 1st module of fast memory)

The portions of main memory referenced by R1 configurations 0-6 may be visualized by referring to the representation of memory module 1 in figure I-11.

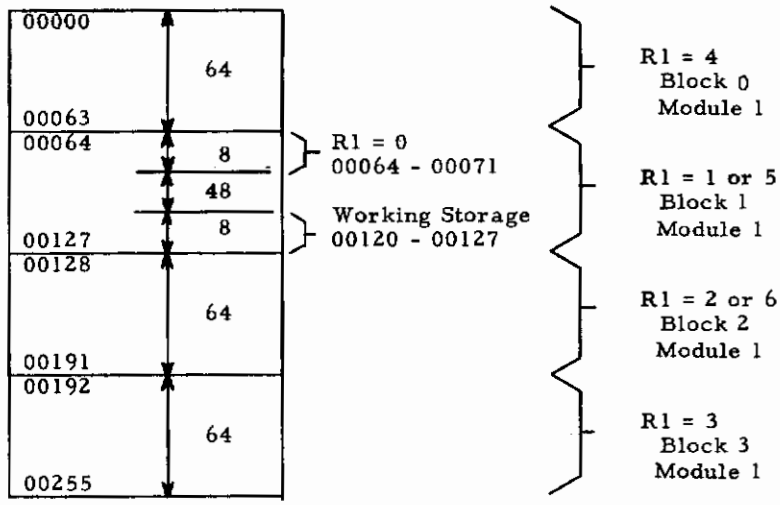


Figure I-11 Main Memory Special Addressing

The portion of fast memory referenced by R1 configuration 7 may be visualized by the representation of memory module 7 in figure I-12.

Note: 28927 is last address in first fast memory module.

Figure I-12 Fast Memory Special Addressing

LDS R1 R2 (VI)	Load Special	An address is constructed from the content of portions of the D register as designated by R1. The content of the addressed cell replaces the content of the register designated by R2.
STS R1 R2 (VII)	Store Special	An address is constructed from the content of portions of the D register, as designated by R1. The content of the register designated by R2 is stored in the addressed memory cell. The content of the register designated by R2 remains unchanged.

Several useful operations can be performed simply and easily by means of these micro-steps. For example, any arbitrary 6-bit character code can be converted into any other code by having available a table of the replacement code, arranged in the proper order, in memory. When one character of the alien code is obtained it is placed in the D register, say in bit positions 18-23. A single micro-step, e.g., "LDS 3 D" will replace the character with one chosen from the table in memory block 3. Another extremely important operation involves branching to one of 64 possible microutines on the basis of a macro-instruction operation code. If this code is located in, say, positions 0-5 of the D register, and a table of microutine starting locations is located in memory block 0, then execution of the micro-step "LDS 4 P" will cause the desired branch of control. If a faster branch is desired, in which fast memory is used for the "jump table" (located in the last 64 words of the first 256-word block of fast memory) the micro-step "LDS 7 P" will effect the control transfer.

It should be clear that memory location assignments for macro-instructions must take into account those main memory locations actually used by the instruction set, including working storage. It is expected that normal operation will involve using the first 64 words of main memory (Block 0) as a "jump table" for rapid macro-instruction interpretation, and that Block 1 will be used for index registers and, possibly, Block 1, 2, or 3 to replace the character set obtained from input devices with one more amenable to programming use.

3. Logical Word Operations

A rather complete set of operations is provided for the manipulation of a 24-bit word, called a logic word. In all of the following operations all 24 bit positions take part in the operation; where registers of less than 24 bits in length are involved, the "missing" portions take place as if they contained zeroes. In all of these micro-steps R1 and R2 designate registers, and in all cases the result of the operation appears in the register designated by R2. The carry toggles are not affected unless specifically stated in the description.

CPL R1 R2 (I)	Copy Logical	The content of the register designated by R1 replaces the content of the register designated R2. The content of register R1 remains unchanged. If both R1 and R2 designate the same register, no change is observed.
CCL R1 R2 (I)	Copy Complement Logical	The content of the register designated by R1 is complemented (1's complement) and this result replaces the content of the register designated by R2. The content of register R1 remains unchanged unless it is the same register that is designated by R2.
CIL R1 R2 (I)	Copy Increment Logical	A 1 at bit position 23 is added to the content of the register designated by R1, and this result replaces the content of the register R2. All three carry toggles are set by this operation; they will be set to 0 if no carry-out passes their bit position, and will be set to 1 if such a carry is present. Register R1 remains unchanged unless it is also designated by R2.

CDL R1 R2 (I)	Copy and Decrement Logical	An internally generated word, consisting of 24 1's, is added to the content of the register designated by R1. This result replaces the content of register R2. The net effect is the same as if a 1 at bit position 23 had been subtracted from the content of R1. The carry toggles are not set by this operation, and register R1 remains unchanged unless it is also designated by R2.
EXC R1 R2 (I)	Exchange	The contents of the two registers designated are exchanged. It doesn't matter which of the two is designated by R1 and which by R2; the result is the same. If R1 and R2 designate the same register, no change is observed.
AND R1 R2 or EXT R1 R2 (I)	And Extract	The logical product of the contents of the two registers replaces the content of the register designated by R2. The result has a 1 only in those bit positions where both words contained a 1. The content of R1 is unchanged.
LOR R1 R2 (I)	Logical Or	The logical sum of the contents of the two registers replaces the content of the register designated by R2. The result has a 1 in any bit position where either of the words contained a 1. The content of register R1 is unchanged.
XOR R1 R2 (I)	Exclusive Or	The logical operation "exclusive or" is performed on the two designated words, and the result replaces the content of the register designated by R2. The result has a 1 in any bit position where one word had a 1 and the other did not; in bit positions where both words contained a 1, or both contained 0, the result is 0. Note that if R1 and R2 designate the same register, the result is an all-zero register.
ADL R1 R2 (I)	Add Logical	The arithmetic sum of the two designated registers replaces the content of the register designated by R2. All three carry toggles are set by this operation. Register R1 remains unchanged unless it is the same register designated by R2. In this case the result is identical with that obtained by shifting the content of the register left 1 position.
ALC R1 R2 (I)	Add Logical for Carry	The three carry toggles are set by this operation, based on whether or not a carry-out would occur at their bit position if the contents of the two designated registers were added together arithmetically. The contents of the registers designated by R1 and R2 remain unchanged.
SLS R1 R2 or SL6 R1 R2 (I)	Shift Left Six	The content of the register designated by R1 is shifted left 6 bit positions, end-around, and the result replaces the content of the register designated by R2. The content of register R1 remains unchanged unless it is also designated by R2.

4. Partial-Word Operations

Many of the micro-steps reference certain portions of the 24 bit word, and operate only on that portion. The remainder of the word remains unchanged by the operation. As with the logic word format micro-steps, R1 designates one of the addressable registers and R2 designates the other, with the result appearing in R2. The carry toggles are not affected unless the description so states.

a. Sign Bit Operations

CPS R1 R2 (I)	Copy Sign	The sign position (bit 0) of the register designated by R1 replaces the sign position of the register designated by R2. The remainder of register R2 remains unchanged. Register R1 is not affected by this operation. If R1 and R2 designate the same register no change is observed.
CCS R1 R2 (I)	Copy Complement Sign	The sign position of register R1 is copied into the sign position of R2 in complemented form. If the sign of register R1 were positive (contained a zero) then the sign position of register R2 will be negative (will contain a 1) after the operation. If R1 and R2 designate the same register, the net effect is to change the sign of that register.

ADS R1 R2 (I)	Add Signs	The sign position of register R1 is added to the sign position of register R2, the result replacing the sign of register R2. Carry toggle 0 will be set by this operation. It will be on (will contain a 1) if the sign of both designated registers contained a 1, and will contain a zero (be set off) otherwise. The sign of register R1 is unchanged unless it is also designated by R2.

These three micro-steps give the programmer complete control over the sign position of registers "A", "B", "C", and "D", the only registers whose sign position is actually present. Reference may be made to the other registers, particularly in the R1 field of the micro-step, where this might be useful. Remembering that any "missing" portion of a register may be thought of as containing zeroes, the micro-step "CPA N A" will set the sign of the "A" register positive. Similarly, "CCS L B" will set the sign of "B" negative. The sign position of a register may be "preserved" (in carry toggle 0) by the micro-step "ADS C C". The result of this operation will leave the sign of the "C" register zero, and carry toggle 0 will be on only if the original content of the sign position was a 1 (was negative).

It should also be noted that "ADS A D" will leave the sign of the "D" register in a state which indicates whether it was the same as the sign of the "A" register or not. If the signs of "A" and "D" were both positive, or were both negative, the D register sign will be positive following the operation. It will be negative otherwise. This operation corresponds to the logical operation of "exclusive or", applied only to the sign positions of the two registers. The micro-step "CCS B B" will, or course, reverse the sign of the "B" register.

b. Magnitude Operations

CPM R1 R2 (I)	Copy Magnitude	The magnitude field (bits 1-23) of the register designed by R1 replaces the same field of the register designated by R2. The sign position of the R2 register remains unchanged. If both R1 and R2 designate the same register, no change is observed.
CCM R1 R2 (I)	Copy Complement Magnitude	The magnitude field of the register designated by R1 is complemented (1's complement) and this result replaces the same field of the register designated by R2.
ADM R1 R2 (I)	Add Magnitude	The magnitude field of the register designated by R1 is added to the magnitude field of the register designated by R2. This result replaces the magnitude field of the R2 register. The sign position of the R2 register is not affected. Carry toggles 1 and 9 are both set by this operation, but carry toggle 0 is not affected If R1 and R2 designate the same register, the result will be the same as a magnitude left-shift of 1, with carry toggle 1 indicating the value of the bit "shifted off".
AMK R1 R2 (I)	Add Magnitude with Carry-in	The magnitude field of the register designated by R1 is added to the magnitude field of the register designated by R2, and at the same time, the content of carry toggle 1 is added into bit position 23 of the sum. Carry toggles 1 and 9 are set by this operation, but carry toggle 0 is not affected.

The micro-steps described above allow the computer to perform arithmetic computations using data words arranged in a "sign-and-magnitude" format, providing the sign of the sum is determined separately and is inserted into the sign position of the proper register. Double-precision addition in this format is particularly easy providing the format does not require that the sign position of the least significant word be used. For example, if the "A" and "B" registers contain one double-precision data word and the "C" and "D" registers contain the other, the micro-sequence

ADM D B	D + B → B
AMK C A	CT_1 + C + A → A

places the double-precision sum in the A and B registers. In the case of differing signs, the programmer must expand this sequence to perform the complement operation on the proper words, and to establish the sign of the result.

c. Floating Point Format Operations

Certain of the micro-operations have been chosen to allow rapid manipulation of double-precision data words in a particular format; this format is shown in Figure II-1 (Appendix II) and is designated the "floating-point format". It should not be assumed that the format shown is

the only possible one which can be implemented on the computer; it is one whose execution times will be particularly short. The micro-steps described below reference the "exponent" and "fraction" fields of the floating-point format. It should be noted that the "fraction"field of 15 bits is also a convenient field for an address in a macro-instruction word, and the micro-steps dealing with this field can also be used to single out the address field when it corresponds.

CPX R1 R2 (I)	Copy Exponent	The exponent field of the register designated by R1 replaces the exponent field of the register designated by R2. Bit 0 and the bit field 9-23 of the R2 register are not affected by this operation. The exponent field in any register may be set to zero selectively if the R1 field references a register whose exponent field is not present (is permanently zero).
CCX R1 R2 (I)	Copy Complement Exponent	The exponent field of the register designated by R1 is complemented (1's complement) and the result replaces the exponent field of the register designated by R2. If R1 references a register whose exponent field is permanently zero, and R2 designates a full 24-bit register, the exponent field in the latter register is set to all 1's.
CIX R1 R2 (I)	Copy and Increment Exponent	A 1 at bit position 8 is added to the exponent field of the register designated by R1, and the result replaces the exponent field of the register designated by R2. Carry toggle 1 is set by this operation, and will contain a 1 only if the original exponent field of the R1 register contained all 1's. Carry toggles 0 and 9 are not affected by this operation.
ADX R1 R2 (I)	Add Exponents	The exponent field of the register designated by R1 is added to the exponent field of the register designated by R2, and the sum replaces the exponent field of the R2 register. Carry toggle 1 is set by this operation; the other carry toggles are not affected.
CPF R1 R2 (I)	Copy Fraction	The fraction field (bits 9-23) of the register designated by R1 replaces the fraction field of the register designated by R2. Bits 0-8 of the R2 register are not affected by this operation. The fraction field in any register may be set to zero if the R1 field references a register whose fraction field is not present.
CCF R1 R2 (I)	Copy Complement Fraction	The fraction field of the register designated by R1 is complemented (1's complement) and the result replaces the fraction field of the register designated by R2. If R1 references a register whose fraction field is permanently zero, the fraction field in the register referenced by R2 is set to all 1's.
ADF R1 R2 (I)	Add Fraction	The fraction field (bits 9-23) of the register designated by R1 is added to the fraction field of the register designated by R2, and the result replaces the fraction field of the R2 register. Carry toggle 9 is set by this operation; the other carry toggles are not affected.
AFK R1 R2 (I)	Add Fraction with Carry-in	The fraction field of the register designated by R1 is added to the fraction field of the register designated by R2, and, at the same time, the content of carry toggle 1 is added into bit position 23 of the sum. Carry toggle 9 is set by this operation; the other carry toggles remain unchanged. The resulting sum replaces the fraction field of the R2 register; bit positions 0-8 of the R2 register remain unchanged.

An additional micro-step, which might be included in this group, is the one designated "SFR -- Shift Floating Right". Since it is somewhat specialized, and involves the use of the N register for repeat-counting, its description will be deferred until the discussion of the other shift operations.

d. CLN and CLD -- Copy Literal to N and Copy Literal to D

Several "copy" operations of a specialized nature are possible utilizing micro-steps of this type. Some of these are described under "Transfer of Control", since this is the effect obtained. Another two micro-steps provide the same function, but reference different registers.

The term "Literal" is used in these micro-steps to designate that the operand is contained in the modifier field of the micro-step.

CLN M or LRC M (I)	Copy Literal to N Load Repeat Counter	The Modifier field of this micro-step, treated as a single 6-bit unit, is copied into bit positions 3-8 of the N register. Bit positions 1 and 2 of the N register are set to zero by this operation. This micro-step provides a simple way of initializing the "N" register for repeated operations such as multiplication, division, and the various shifting micro-steps.
CLD M (I)	Copy Literal to D	The modifier field of this micro-step, treated as a single 6-bit unit, is copied into bit positions 18-23 of the "D" register. Bit positions 0-17 are set to zero by this operation.

e. CFS and CTS -- Copy from Special and Copy to Special

This pair of micro-steps is provided primarily for the purpose of program interrupt operation. They allow the programmer to record into the designated register the contents of the various toggles for storage into main memory, and provide the facility of retrieving this information after the interrupt operation is complete and restoring the original state of the toggles. Included as well is the facility for setting special (optional) registers connected with high-speed input-output devices. It is also possible to set an information pattern into any class of toggle, or into all three classes together.

In these micro-steps, the R1 field designates the class, or classes, of device which is to be referenced, according to the following table:

R1	Items Referenced
4	All Carry Toggles
5	All Interrupt Masks
6	All Program Flags
7	All Toggles (Items 4, 5, and 6 Above)

The R2 field is used to designate a register which either receives the information from the various items, or provides the information for setting the items referenced by the R1 field. By convention, the various classes of toggles are copied to or from the following register bit positions:

1. Carry toggles 0, 1, and 9 correspond to register bit positions 9-11.
2. The interrupt masks correspond to register bit positions 14-17.
3. The program flags correspond to register bit positions 18-23.

CFS R1 R2 (I)	Copy from Special	The information contained in the item, or items, designated by R1 is copied into the register designated by R2. If the information does not occupy the complete register, the "unused" portions of the register are set to zero. Where toggles are designated by R1, if the toggle is in the on, or true, state, the register bit position corresponding will be set to 1. If the toggle is off (false state) the bit position will be set to zero.
CTS R1 R2 (I)	Copy to Special	The information contained in the register designated by R2 is copied into the item, or items, designated by R1, where such an operation is possible.

5. Shifting Operations

All shifting operations are performed in a repeated fashion (with the exception of the "SLS -- Shift Left 6" micro-step) and utilize the "N" register for counting purposes. Execution proceeds at a rate of 1 clock pulse per bit shifted (a shift of either zero or 1 position requires 1 clock pulse). For the direct-shift micro-steps, the programmer must initialize the "N" register prior to execution of the shift micro-step. A count may be copied into the "N" register from some other register; it may be loaded from memory, or may be inserted by means of the "LRC -- Load Repeat Count" micro-step. Execution of the direct-shift micro-steps will repeat until the "N" register is "counted down" to zero. The carry toggles are not affected by these micro-steps.

a. SSL and SDL -- Shift Single Length and Shift Double Length

These two micro-steps utilize the R1 field to designate the type of shifting operation desired. To simplify discussion, the three bit positions of the R1 ield are called bits A, B, and C. In these two micro-steps they are given the following meanings:

Bit	Value	Type of Shift
A	0	Shift Left
A	1	Shift Right
B	0	Closed (End-Around) Shift
B	1	Open Shift
C	0	Magnitude Shift (sign position does not take part in the operation)
C	1	Logical Shift (sign position is shifted)

When an "open" shift is indicated, the vacated positions of the register are set to zero, and the bits "shifted off" are lost. The "N" register will always contain zeroes after the execution of a direct-shift operation.

SSL R1 R2 (IV A) — Shift Single Length

The content of the "N" register is examined. If it is initially zero, no operation takes place. If it is non-zero, the content of the register designated by R2 is shifted one bit position in the manner specified by R1, and the content of the "N" register is decremented by 1. This process is repeated until the "N" register is reduced to zero.

SDL R1 R2 (IV A) — Shift Double Length

The coupled contents of the "A" and "B" registers, treated as a single double-length register, are shifted in the manner designated by R1. The R2 field of this micro-step is not used. The "N" register is decremented by 1 for each bit position shifted. The operation is repeated until the "N" register is reduced to zero.

b. SFR -- Shift Floating Right

This micro-step is provided to facilitate certain floating-point operations:

SFR R1 R2 (IV A) — Shift Floating Right

The contents of the fraction field of the "A" register and the magnitude field of the "B" register are treated as a single, extended-length register and are shifted right until the content of the "N" register is reduced to zero. Zeroes fill the vacated portion of the "A" register, and bits shifted beyond position 23 of the "B" register are lost. The R1 and R2 fields are not used. The sign and exponent fields of the "A" register and the sign position of the "B" register are unaffected.

c. SLC -- Shift Left and Count

Although designed primarily as an aid to manipulations involving normalized floating-point numbers, this micro-step can be utilized for various types of logical operations as well. The magnitude field of the "B" register, and designated portions of the "A" register, are treated as a single extended-length register for this operation. The R1 field of the micro-step serves two functions: It designates which portions of the "A" register are to be involved in the shifting operation, and also indicates which bit position, or positions, are to be examined for termination of the shifting process. The R2 field is not used.

The shifting operation is always a left shift, and is repeated until one of the bit positions designated for examination contains a 1. There are circumstances in which this operation will not terminate, and it is the programmer's responsibility to see that they do not occur. For each bit position shifted the "N" register is decremented by 1, but the "N" register is not examined by the process; it terminates only when one of the designated bit positions of the "A" register is non-zero.

The effect obtained from the various possible configurations of the R1 field can best be seen by reference to a table. The letters "S", "X", and "F" are used to designate the sign field, the exponent field, and the fraction field respectively. Where more than one letter appears it indicates that the designated fields take part in the shift; absence of a letter or letters indicates that the fields do not take part in the operation. The R1 field is shown in binary format.

R1	Portion of "A" Shifted	Bit Positions of "A" Examined
000	F	None. This will not terminate.
001	F	Position 9 only
010	XF	Position 1 only
011	XF	Positions 1 and 9
100	SXF	Position 0 only
101	SXF	Positions 0 and 9
110	SXF	Positions 0 and 1
111	SXF	Positions 0, 1 and 9

SLC R1 R2 — Shift Left and Count
or
FLC R1 R2 (IV B) — Float Left and Count

The coupled contents of the magnitude field of the "B" register and designated portions of the "A" register are treated as a single extended-length register and are shifted left until one (or more) of the bit positions of "A" designated for examination contains a 1. The content of "N" is decremented by 1 for each bit position shifted. The sign position of "B" and portions of "A" not designated for shifting remain unchanged. Vacated portions of "B" are set to zero.

6. Multiplication and Division

Two micro-step operations are provided which can greatly simplify the arithmetic operations of multiplication and division; in many cases they are sufficient in themselves to provide this function. They are both repeated micro-steps, and the programmer must specify, by establishing the proper count in the "N" register, how many times they are to be repeated. The multiplication operation (MPS -- Multiply Step) requires 1 clock time per step of its execution, while divide step (DVS) requires 2. The operation of MPS is perhaps best explained by means of a flow chart (Figure I-13). The conditions which must be established by the programmer prior to the execution of MPS are as follows:

1. The multiplier is loaded into the "B" register.
2. The multiplicand is loaded into the register designated by R1 (normally the "C" or "D" register).
3. The repeat count is loaded into the "N" register.
4. If the result is to be a simple double-length product the "A" register must be set to zero. If it is not zero, its content will be added into the least significant bit positions of the double-length product.

Assuming the "N" register contains "23" at the start of the operation, indicating that the multiply step operation is to be executed 23 times, the results will be as follows:

1. The "A" register will contain the most significant bits of the product in bit positions 1-23. Bit position 0 of the "A" register does not take part in the operation and remains unaffected.
2. The "B" register will contain the least significant half of the double-length product in bit positions 1-23. Bit position 0 of the "B" register does not take part in the operation and is unaffected.
3. The "N" register will contain all zeroes.
4. The contents of all the other registers (except, perhaps, "P") remain unaltered.

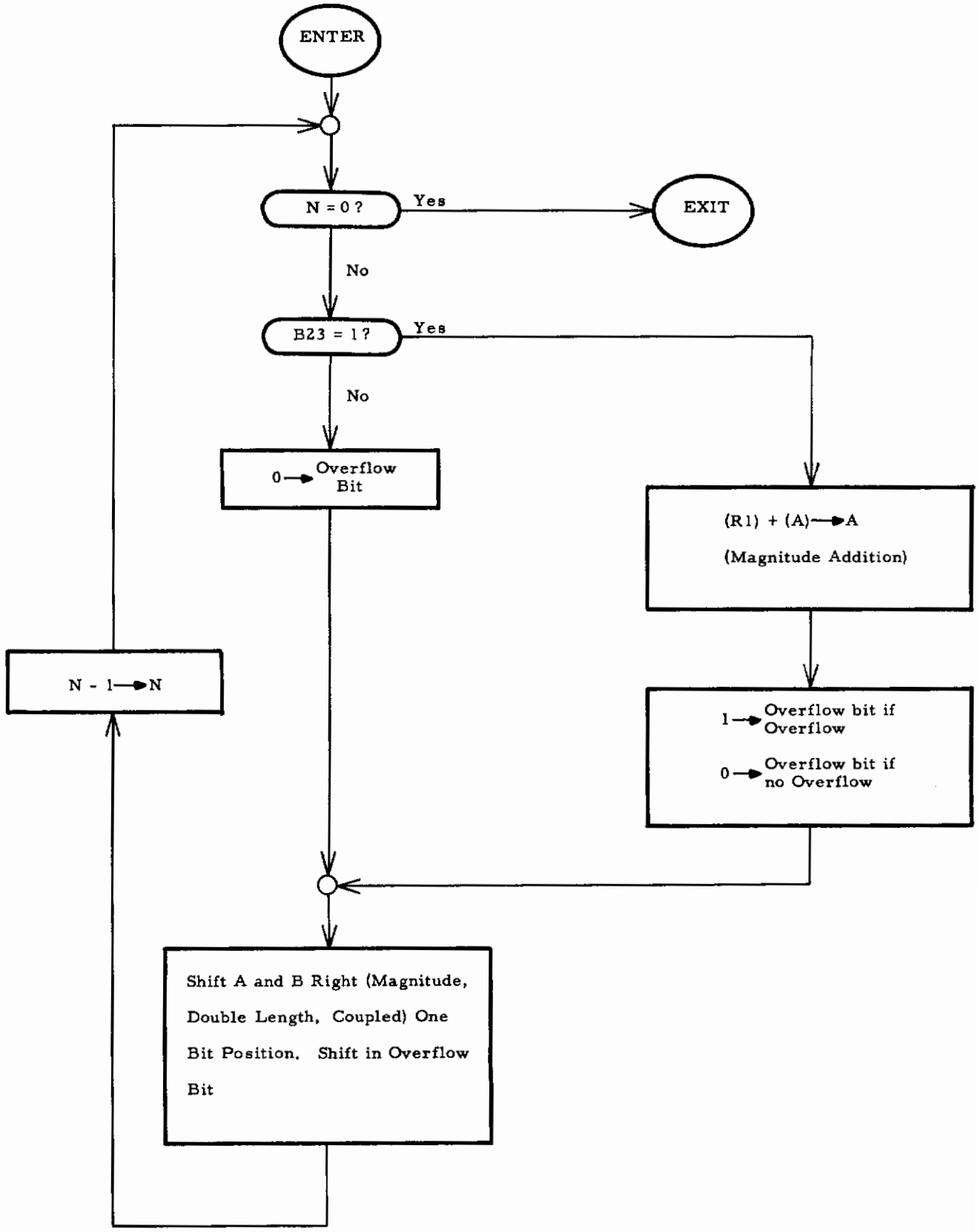


Figure I-13 Multiply Step Operation

This multiplication algorithm has many advantages which may not be obvious from casual inspection. First of all, the sign of the product can be established by adding sign bits of the initial operands using the single micro-step "ADS". For integer multiplication (where the least significant bit is located in bit position 23) the multiplicand is loaded into the "B" register, "A" is set to zero (if no addition is to take place) and the "MPS" micro-step is executed 23 times. The integer product will appear in the "B" register, and overflow will be indicated by a non-zero "A" register. In this case, the sign of the product would be established in the "B" register, either before or after the operation.

Again, dealing with integers, a repeated multiply step operation can give the result B = (BC) + (A), where the letters designate the registers involved, and the "=" sign indicates that the result appears in "B". As before, a non-zero "A" register indicates overflow.

The divide-step algorithm, shown in flow chart form in Figure I-14, has similar advantages. Prior to execution of the DVS operation the programmer must establish the following conditions:

1. The most significant half of the dividend must be in the "A" register in bit positions 1-23. The sign bit of "A" is unimportant, but will be destroyed in the process.

2. The least significant half of the double-length dividend must be in the "B" register, bit positions 1-23. The sign position of the "B" register does not take part in the operation, and may contain the pre-established sign of the quotient if desired.

3. The divisor, in 2's complement form, must be in bit positions 1-23 of the "C" or "D" register. The sign position of that register must contain a 1.

4. The repeat count must be in the "N" register.

Assuming the "N" register contains "23" at the start of the operation, indicating that the divide step operation is to be executed 23 times, the results will be as follows:

1. The quotient will appear in the "B" register in bits 1-23.

2. The remainder will appear in the "A" register, in bit positions 1-23. The sign of the "A" register will, in general, not mean anything; this can best be seen from the flow chart.

3. The "N" register will contain all zeroes.

4. The other registers (except, perhaps, "P") will be unchanged.

No automatic provision is made to detect a divide-overflow condition. This problem is left as an exercise for the programmer.

While it is, in theory, possible to designate some register other than "C" or "D" in using both MPS and DVS, this should be done with caution; in particular, designating "N" as the multiplicand will result in the multiplicand being decremented by one after each iteration. The two micro-steps use the same format as the others, except that the R2 field must designate the "A" register. Neither micro-step affects the carry toggles.

MPS R1 A (IV A)	Multiply Step	The content of the register designated by R1 is multiplied by the content of the "B" register, and the double-length product appears in the magnitude fields of "A" and "B". The "N" register must initially contain the number of multiplication iterations desired. The sign positions of the "A" and "B" registers are not affected. The prior content of the "A" register is added into the least significant bit positions of the double-length product. The "N" register will contain all zeroes.
DVS R1 A (V)	Divide Step	The content of the coupled "A" and "B" registers, treated as containing (in their magnitude fields) a double-length dividend, is divided by the content of the register designated by R1. This latter register must contain the divisor in 2's complement form, and the "N" register must initially contain the number of divide iterations desired. Following execution, the quotient replaces the magnitude field of the "B" register, and the remainder replaces the magnitude field of the "A" register. The sign of the

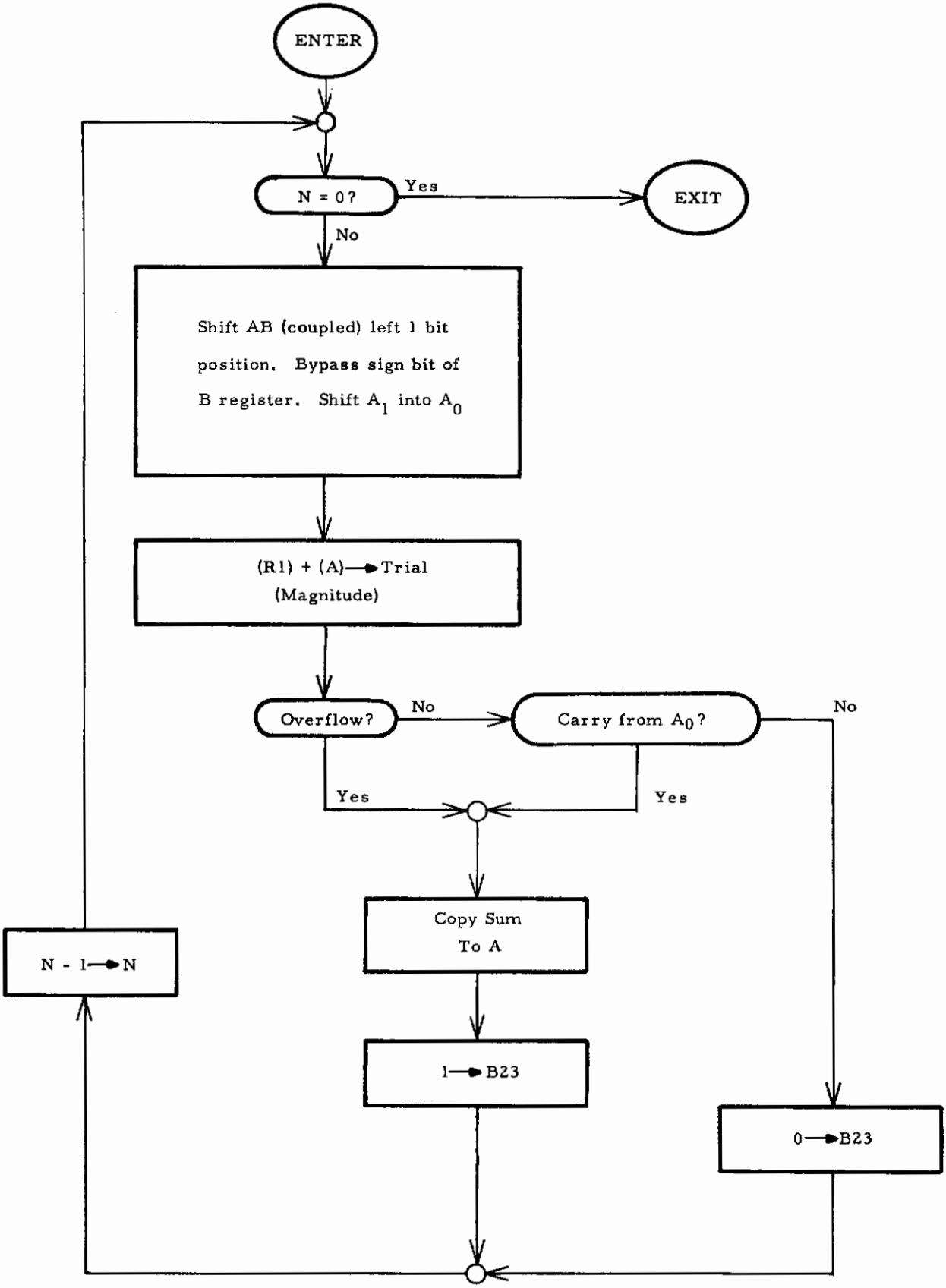


Figure I-14 Divide Step Operation

"B" register is unaffected, but the sign of the "A" register is destroyed. The "N" register will contain all zeroes.

7. Transfer of Control

The "P" register contains the address in which the next micro-pair to be executed is located. If the content of the "P" register is altered during the execution of any micro-pair, the new pair will be obtained from the (altered) location indicated by the "P" register. Three special micro-steps are provided to facilitate this process, although it may also be accomplished by other means. For example, if a transfer to some specified location is desired, it can be accomplished by the micro-step "LDM P P". In this case the transfer address must be located in the memory cell following the one in which the LDM micro-step is located. Transfer to any portion of memory can be effected in this manner. If return-transfer is desired (that is, if one wants to return to this sequence after transfer to another, as in a subroutine call) the micro-pair

LDI	P	D	(P)→D, P + 1→P
EXC	P	D	P⇄D

will cause transfer to the desired location in memory, with the proper return location in the "D" register. In general, any alteration in the "P" register content will cause a transfer of control unconditionally. One interesting point in this regard is the following: Recalling that the "N" register and the "P" register fields do not overlap, the micro-step

AMK	N	P	CT_1 + P→P

will cause the following micro-pair to be skipped (not executed) if carry toggle 1 contained a 1; the pair will be executed if it contained a zero.

The following three micro-steps utilize the modifier field as a unit (see Figure I-10); it is treated as a single 6-bit literal field. Carry toggles are not affected by these micro-steps.

CLP M (I)	Copy Literal to P	The content of the M field replaces bit positions 18-23 of the "P" register. The rest of the "P" register is set to 0 except for bit positions 9-11, which are set to 1. This results in an address which refers to one of the first 64 words of fast memory.
ALP M or FTR M (II)	Add Literal to P Forward Transfer Relative	The content of the M field (from this micro-step) is added into bit positions 18-23 of the "P" register. The new micro-pair to be executed will be obtained from the resulting address. Remember that "P" contains, normally, the address of the next micro-pair to be executed.
ACP M or BTR M (II)	Add Complement (of Literal) to P Backward Transfer Relative	A number is added to the content of the "P" register consisting of all 1's in bit positions 9-17, and the literal field M in positions 18-23. The M field should be in 1's complement form to effect a transfer of control to an earlier part of the sequence. Remember that "P" normally contains the address of the next micro-pair to be executed; if the M field is composed of all 1's (the complement of zero) a dynamic halt will result due to looping back to the current micro-pair.

The last two micro-steps provide simple means for a relative transfer of control for looping and for other operations, while the first of the trio provides a method of control transfer to a group of fixed locations in fast memory. Since each command set requires a control sequence, this could be located in this region for simple return-transfer from the many different microroutines.

8. Test Operations

Several micro-steps are provided for testing bits within registers and most of the hardware toggles. In all instances, the address field of the micro-step specifies the item (bit or toggle) being tested. The response is a conditional prompt "skip" of the next micro-step or micro-pair. If the conditional-test micro-step is the left-half micro-step, the right-half micro-step will be conditionally skipped; if it is in the right-half, the next complete micro-pair will be conditionally skipped.

When a left-half test micro-step is executed, its execution time, as usual, requires one clock pulse. If the condition is met (i.e., the test is "successful") the right-half micro-step will be executed as written and its timing will also be normal. If, however, the condition was not met, the right-half micro-step is not executed, and the next micro-pair is inserted into the "E" register during the next clock pulse.

Execution of a right-half test micro-step precludes obtaining the next micro-pair until the result of the test is known. Thus, if the test is successful, the total elapsed time is two clock pulses. If the test is unsuccessful, the computer increments "P" and places the word addressed by "P" into the "E" register during the same clock pulse. Therefore, a total elapsed time of two clock pulses is required for the execution of a right-half test micro-step whether the condition is successful or not.

a. TZO and TNZ -- Test for Zero and Test for Non-Zero

These two micro-steps can address any one of the eight working registers shown in Figure I-1. The R1 field of the micro-step is used to designate the register, while the R2 field is used to designate the specific portion, or portions, of the register to be tested. The high order bit of the R2 field designates the sign position of the register (bit position 0), the next (or center) bit designates the exponent field (bit positions 1-8) and the least significant bit designates the fraction field (bit positions 9-23). If it is desired to test a portion of a register, the corresponding bit must be a 1; if a bit in R2 is a 0, the corresponding portion of the register will not be tested, as seen below.

R2	Fields tested
000	none
001	fraction
010	exponent
011	exponent and fraction
100	sign
101	sign and fraction
110	sign and exponent
111	sign, exponent, and fraction

For sign-testing, only the sign position should be designated. The convention adopted in the design specifies that a zero sign position is used to designate a plus (or false) state, while a 1 is used to designate a minus (or true) state. The bit pattern of the R2 field may be used in any combination desired; an octal 7 (all three bit positions 1) will test the entire 24 bit register; an octal zero will cause no skip, i.e., the condition tested is considered met.

TZO R1 R2 (14) — Test for Zero

The register designated by R1 is tested for zero in those fields specified by R2. If the indicated register contains only zeroes in the field(s) specified for testing, the next sequential micro-step will be executed. If the register is not all zero in the field(s) specified for testing the next micro-step (or micro-pair) will be "skipped".

TNZ R1 R2 (II) — Test for Non-Zero

The register designated by R1 is tested for zero in those fields specified by R2. If the indicated register contains only zeroes in the field(s) specified for testing, the next micro-step (or micro-pair) will not be executed. If the register is not all zero (is non-zero) in the field(s) specified for testing, the next micro-step (or micro-pair) will be executed, since the specified condition (non-zero) is met.

b. TMT and TMF -- Test Mode Bits for True and Test Mode Bits for False

This pair of conditional-test micro-steps is provided to give extensive bit-testing operations on one specified field of the "D" register. This field (called the mode field, or mode bits) can be tested for any desired bit pattern. It is located in bit positions 6-8 of the "D" register.

It is expected that most instruction sets designed for the PB440 will utilize this field to specify index registers or special addressing modes, and this pair of micro-steps is provided to make interpretation and index-register reference as simple and direct as possible.

In both of these micro-steps, the bit pattern to be tested for appears as the R1 field. "Ones" in the R2 field are used to indicate which of the three bit positions in the "D" register are to be tested. If R2 contains an octal 7, for example, all three of the mode bits of the "D"

register will be tested. If it contains a zero, none of the mode bits will be tested, and the next micro-step (or micro-pair) will be executed.

If the bit pattern which comprises the R1 field is identical, in those bit positions specified for testing by the mask in R2, the condition is met; if it is not identical, the condition is not met.

TMT R1 R2 (II)	Test Mode Bits (of "D") for True	The mode bits of the "D" register (bit positions 6-8) are tested against the bit pattern shown in R1, in those positions designated by a mask in R2. If the pattern is identical, (the condition is met) the next micro-step (or micro-pair) is executed. If the tested condition is not met (the pattern is not identical) the next micro-step (or micro-pair) is skipped.
TMF R1 R2 (II)	Test Mode Bits (of "D") for False	The mode bits of the "D" register (bit positions 6-8) are tested against the bit pattern shown in R1, in those bit positions designated by a mask in R2. If the pattern is identical (the condition is met) the next micro-step (or micro-pair) is not executed. If the tested condition is not met (the pattern is not identical) the next micro-step (or micro-pair) is executed.

c. TCT and TCF -- Test Condition True and Test Condition False

This pair of micro-steps is provided to test various conditions not covered by the zero test and mode test micro-steps. In these micro-steps, the R1 field is used to designate which class of device is to be tested, and R2 designates which member (or members) of that class is to be tested. The device classes which can be tested in this way are:

Class 0:

The six program flags. These are individual toggles which can be set under program control to either the true or false state; they may be considered to be 1 bit (Boolean) registers.

Class 1:

Interrupt masks. These masks (toggles) are used to control the effect of interrupt signals received from the input-output devices. They are described in more detail in the section concerned with input-output equipment.

Class 2:

The carry toggles and the parity toggle. A parity toggle test should be made with caution since the results being tested may be destroyed by an interrupt before the test is made. This possibility may be voided by making certain the micro-step making the test is the right-hand micro-step of the word containing the load or store micro-step which sets the parity toggle for the test. It may also be avoided by turning off the interrupt masks for all channels that could cause an interrupt until the test has been made.

Class 3:

The sense switches. These switches are mounted on the computer console and can be set by the operator. Their setting cannot be altered by a program in the computer.

Class 4:

Interrupt lines. These lines are made true or false by the input-output devices, and represent one way of determining the status of any device. Their operation is described in more detail in the section dealing with the input-output equipment.

Class 5:

"A" or "B" register bit positions. Certain of these bit positions can be tested individually for 0 or 1.

Class 6:

"D" register bit positions. Certain specific bit positions of the "D" register may be tested individually for 0 or 1.

Class 7:

Not assigned.

Some possible combinations represented by R1 and R2 have not been defined. Those which have been defined are shown in the following table:

R1 (Class)	R2	Item
0	1	Program Flag #1
0	2	Program Flag #2
0	3	Program Flag #3
0	4	Program Flag #4
0	5	Program Flag #5
0	6	Program Flag #6
1	1	Interrupt Mask for Channel #1
1	2	Interrupt Mask for Channel #2
1	3	Interrupt Mask for Channel #3
1	4	Interrupt Mask for Channel #4
2	0	All Carry Toggles Zero (Off)
2	1	Carry Toggle 0
2	2	Carry Toggle 1
2	3	Carry Toggle 9
2	7	Parity Toggle (This toggle indicates the parity of the last word involved in a memory access micro-step.)
3	0	All Sense Switches Off
3	1	Sense Switch #1
3	2	Sense Switch #2
3	3	Sense Switch #3
3	4	Sense Switch #4
3	5	Sense Switch #5
3	6	Sense Switch #6
4	0	All Interrupt Lines False. (No interrupt present)
4	1	Interrupt Line #1
4	2	Interrupt Line #2
4	3	Interrupt Line #3
4	4	Interrupt Line #4
4	5	Buffer Channel Request Line
5	1	"A" Register, Bit Position 1
5	2	"A" Register, Bit Position 23
5	7	"B" Register, Bit Position 23
6	1	"D" Register, Bit Position 9
6	2	"D" Register, Bit Position 10
6	3	"D" Register, Bit Position 11

TCT R1 R2 (II) Test Condition True

The condition designated by the R1 (class) and R2 (item) fields is tested. If the condition is true, the next micro-step (or micro-pair) is executed. If the condition tested is false, the next micro-step (or micro-pair) is skipped.

TCF R1 R2 (II) Test Condition False

The condition designated by the R1 and R2 fields is tested. If the condition is false, the next micro-step (or micro-pair) is executed. If the condition is true, the next micro-step (or micro-pair) is skipped.

9. Miscellaneous

a. STT and STF -- Set Toggle True and Set Toggle False

These two micro-steps have been provided to allow the programmer to set the program flags, interrupt masks and carry toggles either "on" (true) or "off" (false). As with the "test condition" micro-steps, the R1 field is used to designate the class of device, and the R2 field is used to designate which item (or items) in the class is to be affected. It will be noted that the same class designations apply to both sets of micro-steps, except those which cannot be changed by the computer program are not defined for the "set" micro-steps. Those combinations which can be used are shown in the following table:

R1 (Class)	R2	Item(s)
0	1	Program Flag #1
0	2	Program Flag #2
0	3	Program Flag #3
0	4	Program Flag #4
0	5	Program Flag #5
0	6	Program Flag #6
0	7	All Program Flags
1	1	Interrupt Mask for Channel #1
1	2	Interrupt Mask for Channel #2
1	3	Interrupt Mask for Channel #3
1	4	Interrupt Mask for Channel #4
2	1	Carry Toggle 0
2	2	Carry Toggle 1
2	3	Carry Toggle 9

STT (I)	Set Toggle True	Set the toggle designated by the combination of the R1 and R2 fields to the true or one state.
STF (I)	Set Toggle False	Set the toggle designated by the combination of the R1 and R2 fields to the false or zero state.

b. Other Micro-Steps

HLT R1 R2 (I)	Halt	Execution of this micro-step causes the automatic program sequencing to stop, but has no effect on any of the registers or toggles. The R1 and R2 fields are not used.
NOP R1 R2 (I)	No Operation	Execution of this micro-step causes no operation to be performed other than (perhaps) the automatic incrementation of the "P" register. The R1 and R2 fields are not used. It does nothing at all, and it takes only 1 microsecond not to do it.
EMP R2 (I)	Execute Micro-Pair	The R2 field designates a register containing a micro-pair to be executed immediately. The content of the "P" register is not automatically incremented during execution of the special pair. If EMP is a left-hand micro-step, the right-hand micro-step will be ignored, and will require no execution time.

10. Input-Output System Commands

This discussion is confined to the input and output hardware of the PB440 main frame, and to micro-steps which perform input and output operations. Information on the detailed operation and control of data input and output devices is contained in a peripheral equipment manual.

An input-output device can communicate with the computer over one of four separate channels. The four channels can be assigned at will to any of the devices present on the computer, and this assignment can be changed as desired, under program control.

Each device contains a "controller" which allows the computer to exercise control over the device, and which informs the computer of the status of the device. Most controllers contain either a character or word buffer and data transmission is either in character- or word-parallel mode. The computer actually communicates only with this buffer for data transfer operations.

All data transfer and I/O device control operations are under control of the computer program, and each action taken is specified by the program. Only one action at a time can be designated, but, by careful programming, many devices can be kept operating at full speed "simultaneously" since the computer program can proceed at many times the response rate of most input-output devices.

Each of the four communication channels contains an "interrupt line" which is capable of interrupting the normal computer program sequence if the programmer has made provision for this type of operation. The four interrupt lines are examined, in numerical sequence, by a four-position switch called a "commutator", at the clock rate of the computer. Each line is, therefore, examined once every four clock pulses for the presence of an interrupt signal. The programmer can control the effect that an interrupt signal can have on each of the four lines individually. The commutator action itself is also under program control. It can be "unlocked", so that it will perform its normal scanning function automatically, or it can be "locked" onto any one of the four channels. It must, in fact, be locked for any data transfer operation to take place.

Any of the devices may be "connected" to, or "disconnected" from the computer under program control, although it is unlikely that the computer will be able to turn them on or off physically. For more advanced configurations of input-output equipment, this connect-disconnect feature makes it possible for more than 4 devices to be operated on-line simultaneously.

The Interrupt Masks are a set of toggles which can be set and tested under program control. Their function is to determine the effect that an interrupt signal is to have on their particular channel. If they are set to the false or "0" state, interrupt signals have no effect on the normal program sequence. The particular interrupt line which has been "masked off" by this action will, however, remain in the true state, and may be tested by means of the TCT and TCF micro-steps. This mode of operation has been named "programmed interrupt", since the conditions under which the program responds to an interrupt signal are determined solely by how the program is written.

If an interrupt mask is set to the true, or "1", state, an interrupt signal on that channel will cause an "automatic interrupt" operation to take place. This operation consists of locking the commutator on the channel which caused the interrupt, waiting until the micro-pair in the "E" register has been executed, and then loading the "E" register from one of the first 4 locations in fast memory. If this mode of computer operation is chosen, the programmer must see to it that the proper interrupt program sequence is present in memory, and that the corresponding location chosen for its first micro-pair contains the proper micro-pair to preserve the content of the "P" register and to transfer control to the interrupt program sequence.

Since each of the interrupt masks can be set as desired, the programmer may choose between these different methods of operation for each separate channel. Any interrupt signal which reaches the commutator (that is, is not "masked off") will cause automatic interrupt and will also lock the commutator to the line on which the interrupt signal was detected. No automatic interrupts can be effected while the commutator remains locked.

An additional line (called the "Buffer Channel Request" line) is available to provide an input-output "ready" signal from remote I/O devices. Several remote devices may be connected to this request line when the servicing of these devices is an exceptional occurrence. Devices on the request line need not be assigned an I/O channel, and may not cause an "automatic interrupt" as described above. Instead, the devices connected to this line will provide the computer with a "ready" signal, which will be reflected in the master interrupt line (as well as the "Buffer Channel Request" interrupt line) being set "true". A micro-sequence may then interrogate the devices connected to the request line to determine which device presented the ready signal, assign an I/O channel to the device, and then service that device. After the device is serviced, the I/O channel may be reassigned to the device formerly on that channel.

a. SEL -- Select an Input or Output Device

The first of the three input-output micro-steps selects a particular device, assigns it to one of the four channels, and receives a report of its status. It uses the R2 field to designate a register which must contain a "command word" for the device in question. The approximate format for this command word is shown below:

S	CO	CN	FM	DN	CH	OP

Bit Pos.	Field	Description
0	S	Status Only Bit
1-4	CO	Controller Type (Part of Device Address)
5-6	CN	Controller Number (Part of Device Address)
7-8	FM	Format Designation
9-11	DN	Device Number (Part of Device Address)
12-13	CH	Channel Assignment
14-23	OP	Operation to be Selected

The "S" field is used to select the device addressed, or to merely return a status response from that device. It acts as a master control over the bit positions in the operation field. If the S field contains a "0", the addressed device is selected and assigned to the channel number indicated in the channel assignment field. If the S field contains a "1" the operation field has no effect; it is used to obtain the status of a device without actually selecting it.

Controller Type Field

The device address actually consists of three parts: The type of controller (i.e., photoreader, typewriter, magnetic tape, etc.), the number assigned to that controller (there may be more than one of that type), and the number of the device in question, since certain controllers can handle more than a single device. The controller types already defined are as follows:

Bit Pos. 1	2	3	4	Controller
0	1	0	0	Paper Tape Punch
0	0	1	0	Paper Tape Reader
0	0	0	1	Typewriter
1	1	0	0	Magnetic Tape
1	0	1	0	Card Punch
1	0	0	1	Line Printer
1	0	1	1	Card Reader

Controller Number Field

The controller number field is defined by the particular controller addressed. Each type of controller for basic equipment has the number 00. Other controllers of that type are assigned numbers other than 00 at the time the PB440 is installed.

Format Designation Field

The interpretation of the remainder of the SEL command word (bits 9-23) is determined by the bit configuration in the "FM" field. The special interpretation of bits 9-23 are for the optional high-speed buffered I/O system.

Bits in Positions 7-8	Interpretation
00	Normal interpretation as shown above.
01	Bits 9-23 indicate the number of words of data that are to be transmitted via the optional high-speed buffered I/O system.
10	Bits 9-23 indicate the initial memory location to be associated with data transmitted by the optional high-speed buffered I/O system.
11	Bits 9-23 indicate the number of words of data that are to be transmitted via the optional high-speed buffered I/O system. (An interrupt is sent upon completion of data transmission.)

Device Number Field

The device number field is defined by the particular device addressed. Each of the basic devices is assigned the number 000. Other devices of each type are assigned numbers other than 000 at the time the PB440 is installed.

Channel Assignment Field

The channel to which the device is to be assigned must appear in the "CH" field of the command word. The channel numbers are as follows:

Bits	Channel
00	Channel #1
01	Channel #2
10	Channel #3
11	Channel #4

Any device may be assigned to any of the four available channels, but should not be assigned to a channel already in use. Should two devices be assigned to the same channel and both be capable of transmitting data at the same time, a data transfer at the same time will result in both data words being combined (logical or) during transmission.

Operation Field

The operation (OP) field utilizes individual bit positions to designate the desired operation(s). More than one operation may be selected if the bits have meaning for the selected device. Certain of the operations have no meaning for certain devices, and are ignored by that controller. (You can't rewind the typewritter, for example.)

The status-response word will be returned by the controller to the register designated by R1. The status-response word is primarily to provide the computer with the status information necessary to control the input-output device. For the basic equipment this information is minimal. For optional devices this information may be quite comprehensive.

For all controllers, bit 23 of the response-word will contain a "1" if the device is present, the power is on, and the device is in working condition; otherwise that bit will contain a zero.

SEL R1 R2 (III) Select — The R2 field is used to designate a register which contains an input-output device command word. This command word is executed (if it indicates it should be) and the device is assigned a channel number and is "connected" to the computer on that channel. A status response word replaces the content of the register designated by R1, indicating the status of the selected device. The timing is such that R1 and R2 may designate the same register.

b. COM -- Commutator Control

This micro-step utilizes the R1 field to indicate which of three possible operations is to be performed. Any combination of these operations is possible, and each is called into action by the presence of a "1" in the proper bit position. If we designate the three bits of the R1 field by the letters A, B, and C, the individual operations possible are as follows:

Bit	Value	Resulting Operation
A	0	Unlock the commutator for automatic scanning. If the commutator is already unlocked, no operation results.
A	1	Lock the commutator to a particular line. Note that line number one (channel #1) has the number 00, line number two has the number 01, etc. The specific line is chosen from the register designated by R2 if bit B = 1, or is its present step in sequence if bit B = 0.
B	0	No effect.

	Bit	Value	Resulting Operation
	B	1	Set the commutator to a value determined by the content of bit positions 21 and 22 of the register designated by R2.
	C	0	No effect.
	C	1	The present setting of the commutator is logically "or"ed into bit positions 21 and 22 of the register designated by R2, leaving the remainder of the bit positions of that register unchanged.
COM R1 R2 (I)	Commutator Control		Perform the action, or actions, indicated by the R1 field on the commutator. Utilize bit positions 21-22 of the register designated by R2 for these actions if required.

c. DTR -- Data Transfer

The actual data transfer operation into and out of the computer is controlled by the third, and last micro-step of this group. For it to operate correctly the device must first be selected (by SEL micro-step) and the status of the device must indicate that it is not busy. A chosen device may remain "connected" to the computer indefinitely; it is disconnected only when specifically instructed by a subsequent micro-step.

The commutator must be locked (either as caused by an automatic interrupt or by means of the COM micro-step) to the proper channel. If it is not locked, no data transfer operation will take place. The execution of the data transfer also results in a conditional jump of the next micro-step (or micro-pair) if a "special condition" line is true. For example, if there is a parity error in the character being transmitted by the photoreader, the special condition line is true and next micro-step (or micro-pair) is executed. If there is no parity error, the next micro-step (or micro-pair) is skipped. Thus the DTR micro-step results in a conditional jump identical to that of the "TCT -- Test Condition True" micro-step, where the condition tested is the special condition line of the input-output system.

The data transfer micro-step utilizes the R1 field to indicate which of several operations is to be performed, as indicated in the following table:

R1	Operation to be Performed
0	Disconnect device.
1	Data Transfer In.
2	Data Transfer Out.
3	Data Transfer IN and OUT. This operation is provided to permit controllers which have been designed for special purposes to be used with the PB440 system.
4	No Data Transfer. The computer does not send or accept information. The controller, however, is activated to send or accept information. This combination is not practical and should be avoided.
5	Data Transfer In, and end of message.
6	Data Transfer Out and end of message.
7	Return status response from device to register designated by R2.

DTR R1 R2 (III)	Data Transfer	Data transferred into or out of the register designated by R2, under control of R1. For data transfers of less than a word of data, the character is placed in or taken from the least significant bit positions of the register. The content of the register is unchanged by the output operation. For both input and output data transfers the next micro-step (or micro-pair) is skipped if the special situation is false. If the special situation is true no skip occurs.

D. MICRO TIMING

The Roman numerals and letters which appear under the mnemonic symbol designate the group to which a micro-step is assigned for timing purposes. These groups are as follows:

GROUP	EXECUTION TIME	(microseconds)
I	One clock pulse unless used as a right-half micro-step which replaces or alters the "P" register.	1, (2)
II	One clock pulse when used as a left-half micro-step; two clock pulses when used as a right-half micro-step.	1↔2
III	The time depends upon the device addressed and whether the device is ready to return the status response or other information to the computer; the computer will delay until the status response is received. The normal execution time for a paper tape device which is ready is four clock pulses.	1 + Variable
IV A	N clock pulses, where N is the number contained in the "N" register prior to execution of the group IV micro-step. The execution time may not, however, be zero. Therefore, if N is zero, the execution time is the same as if N were one, although the result will be that no register, except perhaps "P", is changed.	1↔N
B	N clock pulses, where N is the number of bit positions shifted for normalization. A shift of zero bit positions and a shift of one bit position both require one clock pulse.	1↔N
V	2N clock pulses, where N is the number contained in the "N" register prior to execution of the group V micro-step. As in group IV, the execution time is the same for N = 0 and N = 1.	2↔2N
VI A	Two clock pulses with a shadow time of three clock pulses when the operand is obtained from main memory by a left-half micro-step.	2 + 3ST
B	Three clock pulses with a shadow time of two clock pulses when the operand is obtained from main memory by a right-half micro-step.	3 + 2ST
C	The same time as for group II micro-steps (with no shadow time) when the operand is obtained from fast memory.	1↔2
VII A	One clock pulse with a shadow time of four clock pulses when the operand is stored in main memory by a left-half micro-step.	1 + 4ST
B	Two clock pulses with a shadow time of three clock pulses when the operand is stored in main memory by a right-half micro-step.	2 + 3ST
C	Five clock pulses with no shadow time when the operand is stored in fast memory by a left-half micro-step.	5
D	Six clock pulses with no shadow time when the operand is stored in fast memory by a right-half micro-step.	6

APPENDIX II

PB440 SYSTEMS COMMAND SET

This appendix contains the preliminary instruction list for the Systems Command Set (SCS), one of the instruction sets which will be supplied by the computer manufacturer. The SCS is a PB440 programming language system including a set of instructions (referred to as macro-instructions), several optional microutine macro-instruction interpretive control sequences, and a symbolic assembler. Programmers conversant with program assemblers such as FAP, USE, etc. will find little trouble in writing programs in SCS language and need not concern themselves with the micro-coding details of the PB440. The SCS language provides a wide variety of instructions to select from, many of which would be available on conventional computers only in the form of subroutines or library function routines.

The Systems Command Set contains over 100 commands which have been designed with emphasis on speed and ease of data manipulation. Any 64 of these commands may be selected for use at any one time. In addition, through the use of the MRC command, any number of subroutines may be selected from storage in either fast or main memory. Examples include: polar-to-rectangular coordinate conversion and rectangular-to-polar coordinate conversion; code conversion such as gray-to-binary, binary-to-BCD, and any other code transformation based on table storage (a very efficient operation in the PB440).

An unusual feature of the Systems Command Set, made possible by the stored-logic feature of the PB440, is the availability of several floating-point formats and the ability to intermix these floating-point formats within one operating program. These floating-point formats have 39-bit, 24-bit, and 16-bit mantissas and 8-bit exponents. The 16-bit format is of particular interest because both exponent and mantissa are contained within one computer word. Because of its higher speed and greater program storage economy, the 16-bit floating-point format is expected to be of particular interest for use with real-time data systems. In these cases, the 16-bit mantissa allows more than adequate resolution while at the same time providing for program simplicity through floating-point operations.

Up to 64 controllers each with as many as 8 devices may be connected to the basic I/O bus of the PB440. Each device can communicate with the computer over one of four separate channels which are assigned under program control. The Systems Command Set allows for input-output to occur on all four channels simultaneously with computation. External and internal interrupt registers are associated with each communication channel. External interrupts are useful in systems applications which involve priority interrupts and special purpose peripheral devices. Internal interrupts occur upon termination of an internally commanded buffer operation. For more advanced systems an optional shared-memory system is available which permits 2 or more PB440's to share a common portion of their memory. When high speed I/O adapters are used with the shared-memory system, input-output data can be transferred at rates to 9.6 million bits/second without interrupting program execution.

In addition to being able to select a command set from a library of over 100 commands, the Systems Command Set allows the programmer to add additional commands designed specifically for his problem. Through the use of the DML command (Descend to Micro-Level) the programmer may also intermix micro-steps with commands. Hence, functions such as curve-fitting, cross correlation, as well as short-cut arithmetic operations, I/O operations, etc., may be programmed in three ways:

1. by using commands only,
2. by intermixing micro-steps and commands, or
3. by completely coding the functions with micro-steps and adding them to the list of commands or micro-coded subroutines.

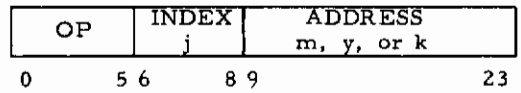
In systems applications when the same operation is frequently repeated, system performance is significantly improved by micro-coding a given special purpose function.

A. THE SYSTEMS COMMAND SET

The Systems Command Set consists of a control sequence and a set of microutines, each of which defines a command. At execution time the control sequence picks up a command, performs any address modification which may be required, interprets the command and transfers to the appropriate microutine. When the microutine has completed its function it returns to the control sequence. The microutines and control sequence are normally stored in fast memory although micro-steps can be executed from main memory at a somewhat slower rate (approximately 50%). Program commands are stored in main memory.

When using a 'canned' command set such as the SCS, the PB440 appears to the programmer as a conventional computer with two 24 bit arithmetic registers A and B, seven 15 bit index registers (only six when indirect addressing option is used), and a storage memory of from 4K to 28K locations. There is an unusual restriction on the addresses which may be used: the first 128 memory locations are not available for conventional use by the programmer. (The explanation for this detail is given in the micro-coding discussion of the LDS and STS micro-steps in Appendix I.)

Systems commands assume the 24-bit format shown in Figure II-1. Bit positions 0-5 contain the operation code which specifies the function of the command. Bit positions 6-8 contain the index designator which is used for address modification either by indexing or indirect addressing. The base execution address in bits 9-23 may be used as the address operand, m; as an operand, y; or as a shift or logic count, k. The address of an operand, m, and the operand, y, occupy the full 15 bits with the least significant bit in position 23. The k count however only occupies positions 13-20 with the least significant bit in position 20.



Note: 0 In the Index Field = No Index

1-6 In the Index Field = Address Modification by Specified Index Register

7 In the Index Field = Indirect Address or Address Modification by Index Register 7

Figure II-1 SCS Command Format

Note that the op code field is six bits in length providing 64 possible instructions in the set. An examination of the SCS instruction list (in a following section) will reveal a great deal more than 64 instructions available. This seemingly awkward situation is taken care of by limiting any program to a maximum of 64 different commands. Do not construe this restriction as troublesome, in view of the widespread use of computers with this same limitation to 64 operation codes. In fact, the PB440 SCS provides the programmer the freedom of choosing which 64 of the many commands available in the SCS list he may use for his program.

An interesting aspect of the PB440 SCS now comes to light; that, as the source program is translated in the assembler to machine language, the machine operation codes are assigned to the mnemonic command designations. This implies that it is extremely unlikely that a symbolically defined instruction such as LDA (Load A from memory) will have the same machine operation code, say 73_8, in two different programs. This need be of no concern to the programmer except to point out that there is little benefit in memorizing the macro-instruction machine operation codes for a given program, since they do not carry over to the next program.

The arithmetic registers available in the SCS system are the A and B registers and may be directly accessed by system commands. These registers may be used independently or as coupled registers as shown in Figure II-2. When they are coupled in conjunction with logical operations, bit position 0 of register B is considered to be adjacent to bit position 23 of the A register. When they are coupled in conjuction with arithmetic operations such as multiply and divide, bit 0 of B is disregarded and bit 1 of B is considered to be adjacent to bit 23 of the A register. The L register, which is used as the command address counter, is not directly accessible by system commands, but may be modified in the course of executing a transfer or skip command. The C, D, P, and N registers cannot be accessed by commands but are available to the programmer at the micro-level. Of course, these registers are used by the microutines during the execution of commands. The eight index registers (0-7) available to the Systems Command Set are located in main memory cells 64_{10}-71_{10}. Indexing commands treat these locations as 15-bit registers. The command set control sequence performs all address modifications.

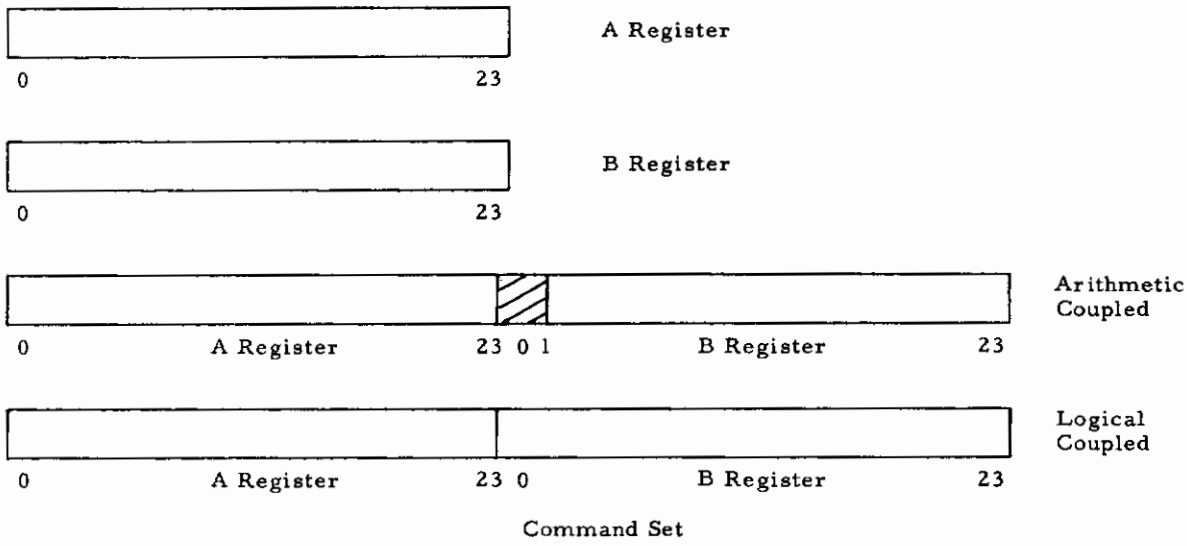


Figure II-2 SCS Arithmetic Registers

B. SCS COMMAND DESCRIPTIONS

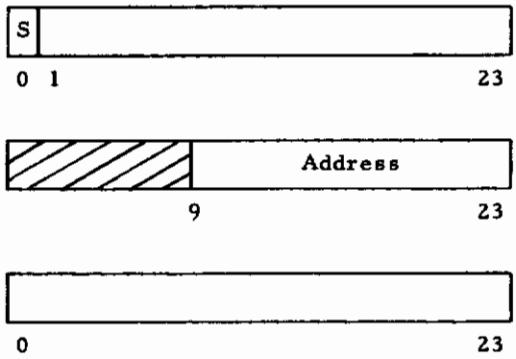
The systems commands are described on the following pages. The title line for each command contains the mnemonic code, the command format, the command name, and the execution time. Abbreviations and symbols are defined as follows:

A	A Register (24 bits)
B	B Register (24 bits)
C	C Register (24 bits)
D	D Register (24 bits)
j	Index Designator
J	Index Register Designated by j
k	Base Shift Count
K	Effective Shift Count
L	Command Address Counter (15-bit register)
m	Base Operand Address
M	Effective Operand Address
N	N Register (8 bits)
P	Micro-Pair Address Counter (15-bit register)
r	Remainder
y	Base Operand
Y	Effective Operand

It is to be noted that the following command list is open ended and is expected to be expanded as the need arises. The times given are in microseconds and, being of a preliminary nature, are subject to change. The quantity x is a function of the interpretive control sequence option as discussed in section C below, and may range from 4 to 18 microseconds.

Fixed Point Arithmetic

The fixed point operand formats are shown in figure II-3 below.



Single precision Fixed point

Index register operand

Logic operand

Figure II-3 Fixed Point Operand Formats

All fixed-point arithmetic operations are performed using a binary, 2's complement number system. Error conditions may arise during fixed-point arithmetic operations which will cause an overflow indicator to be turned on. The existence of an overflow condition may be tested by the Transfer On Overflow (TOV) instruction. The conditions which cause the overflow indicator to be turned on are:

1. The capacity of a 24-bit register is exceeded as a result of addition or subtraction. This can occur only when adding two numbers of like signs or subtracting two numbers of opposite signs. The capacity of all 24-bit registers is is limited by $+(2^{23} - 1)$ and $-(2^{23} - 1)$. The number $-(2^{23})$, which is represented by a 1 in bit position 0 and 0's in bit positions 1 - 23, is invalid and an overflow indication will result when any operation produces this bit configuration.

2. The divisor is zero or the most significant part of the dividend (contents of the A register) is greater in magnitude than the divisor when a division is performed.

3. The multiplier has a value of $-(2^{23})$ when a multiplication is performed. If the multiplicand is also $-(2^{23})$ the error condition will not be detected and the overflow indicator will not be turned on. In either event the multiply operation will result in zero. The invalid number $-(2^{23})$ may be obtained only when performing add and subtract operations and when the overflow condition described in number 1 exists.

In the Systems Command Set the overflow indicator is physically represented by hardware carry toggles zero and one. After all valid arithmetic operations both carry toggles will be in the same condition, either set or not set. After an invalid operation the carry toggles will be in an opposite state. The existence of this condition may be tested by the Transfer On Overflow (TOV) command. The carry toggles are reset by all arithmetic commands but are not reset by the Transfer On Overflow command. The multiply and divide commands utilize a coupled arithmetic register. The product obtained from a multiplication is placed in the coupled AB register. The most significant part of the product will be in the A register (bits 0 to 23); the least significant part will be in the B register (bits 1 to 23). Bit 0 of the B register is not used as part of the product. The coupled AB register is also used to contain the dividend for the divide command.

CLA j m Clear A 2 + x

The contents of the A register are cleared to zero.

CLB j m Clear B 2 + x

The contents of the B register are cleared to zero.

ADA j m Add to A Register 3 + x

The 24-bit operand obtained from memory location M is added to the contents of the A register. Memory location M remains unchanged.

ADB j m Add to B Register 5 + x

The 24-bit operand obtained from memory location M is added to the contents of the B register. Memory location M remains unchanged.

SBA j m Subtract from A Register 7 + x

The contents of memory location M are subtracted from the contents of the A register. Memory location M remains unchanged.

SBB j m Subtract from B Register 7 + x

The 24-bit operand obtained from memory location M is subtracted from the contents of the B register. Memory location M remains unchanged.

MPY j m Multiply B Register Times Memory (36 - 43) + x

The contents of the B register are multiplied by the 24-bit operand obtained from memory location M. The 47-bit product replaces the contents of the A and B registers with the A register containing the most significant bits of the product and the B register containing the least significant. Bit 0 of the B register is not used as part of the product. A correct product will result regardless of whether the A register was zero or not prior to execution of this command. The prior contents of A however will be destroyed by the resultant product.

DIV j m Divide AB Registers by Memory (66 - 69) + x

The contents of the coupled AB registers are divided by the contents of memory location M. Before the command is executed, bit positions 0 to 23 of the A register contain the most significant part of the dividend and bit positions 1 to 23 of the B register contain the least significant part of the dividend. The quotient created by the division replaces the contents of the B register and the remainder replaces the contents of the A register. The remainder is always positive.

RAD j m Replace Add A Register 7 + x

The contents of the A register are added to the contents of M. The sum replaces the contents of the A register and memory location M.

RSB j m Replace Subtract A Register 8 + x

The contents of the A register are subtracted from the contents of memory location M. The difference replaces the contents of the A register and memory location M.

INA j y Increment A Register 8 + x

The 15-bit operand Y (with bit 9 extended into bits 0-8) is added to the contents of the A register and the sum replaces the contents of the A register.

INB j y Increment B Register 8 + x

The 15-bit operand Y (with highest bit extended) is added to the contents of the B register and the sum replaces the contents of the B register.

IMO j m Increment Memory by 1 7 + x

The contents of location M are incremented by 1 and the sum replaces the contents of location M. The A and B registers are not changed by this command.

DMO j m Decrement Memory by 1 7 + x

The contents of memory location M are decremented by 1 and the difference replaces the contents of location M. The A and B registers are not changed by this command.

Data Transmission

LDA j m Load A Register 4 + x

The contents of memory location M replace the contents of the A register.

LDB j m Load B Register 4 + x

The contents of memory location M replace the contents of the B register.

LAC j m Load A Register Complemented 5 + x

The 2's complement of the contents of memory location M replaces the contents of the A register. The contents of M are not changed.

LBC j m Load B Register Complemented 5 + x

The 2's complement of the contents of memory location M replaces the contents of the B register. The contents of M are not changed.

STA j m Store A Register 3 + x

The contents of the A register replace the contents of memory location M. The contents of A remain unchanged.

STB j m Store B Register 3 + x

The contents of the B register replace the contents of memory location M. The contents of B remain unchanged.

ZTS j m Zeros to Storage 2 + x + 5n

The contents of consecutive memory locations beginning at M are cleared to zero. Before the command is executed the B register contains the number of locations to clear. Upon completion of this command the B register will contain zero.

SAA j m Store Address Field of A Register 7 + x

The address field (bits 9-23) of the A register replaces the address field of memory location M. Bits 0-8 of M remain unchanged. The contents of A are not changed.

SAB j m Store Address Field of B Register 7 + x

The address field (bits 9-23) of the B register replaces the address field (bits 9-23) of memory location M. The remaining bits (0-8) of location M remain unchanged. The contents of B remain unchanged.

STZ j m Store Zero to Memory 3 + x

The contents of memory location M are replaced by zero. The contents of the A and B registers remain unchanged.

ENA j y Enter A Register 6 + x

The 15-bit operand Y (with bit 9 extended into bits 0-8) replaces the contents of the A register.

ENB j y Enter B Register 6 + x

The 15-bit operand Y (with sign extended) replaces the contents of the B register.

XAB j m Exchange A and B Register 2 + x

The contents of the A register and the contents of the B register are exchanged. The j and m fields of this command are not used.

MOV j m Move Data Block 10n + x

The contents of consecutive memory locations beginning at M are moved to consecutive locations beginning at the location specified in the address field of the A register. The B register must contain the number of locations to be moved. Upon completion of this command the B register contains zeros and the A register contains the address of the last word stored plus one.

PDQ j m Push Down Queue 10n + 1 + x

The data in the queue beginning at location M are pushed down by 1 position. The contents of the A register replace the contents of the first location in the queue and the contents of the last location in the queue are left in the A register. Prior to execution of PDQ the B register must contain the number of locations in the queue. Upon completion of the command the B register contains zero.

Transfers

TRA j y Unconditional Transfer 2 + x

The 15-bit operand Y replaces the contents of the L register (command address counter) and the next command is taken from memory location M where M = Y.

TAZ j y Transfer if A Register is Zero 3 - 4 + x

The contents of the A register are tested for zero. If the A register contains zero, the 15-bit operand Y replaces the contents of the L register and the next command is taken from M.

TBZ j y Transfer if B Register is Zero 3 - 4 + x

The contents of the B register are tested for zero. If the B register contains zero, the 15-bit operand Y replaces the contents of the L register and the next command is taken from location M.

TAP j y Transfer if A Register is Positive 4 + x

If the contents of the A register are positive, the 15-bit operand Y replaces the contents of the L register and the next command is taken from location M.

TBP j y Transfer if B Register is Positive 4 + x

If the contents of the B register are positive, the 15-bit operand Y replaces the contents of the L register and the next command is taken from location M.

TOV j y Transfer if Overflow 5 + x

If the overflow indicator is set (carry toggles zero and one in opposite state), the 15-bit operand Y replaces the contents of the L register and next command is located at M. This command should be used immediately after an arithmetic operation suspected of overflow. Refer to the discussion on overflow in the description of fixed-point arithmetic commands.

TSR j y Transfer and Set Return 9 + x

The contents of the L register replace the contents of location M and the next command is taken from location M + 1. Bits 0-8 of location M are set to zero by this command.

RET j m Return (Indirect Jump) 4 + x

The contents of memory location M (bits 9-23) replace the contents of the L register and the next command is taken from the location whose memory address is contained in M.

Compares and Skips

CAE j m Compare A Register and Memory for Equal 6 - 7 + x

The contents of memory location M are compared with the contents of the A register for an equal condition. If the condition is met the next sequential command is skipped; otherwise the next command is executed. The contents of the A register are not changed.

CBE j m Compare B Register and Memory for Equal 6 - 7 + x

The contents of memory location M are compared with the contents of the B register for an equal condition. If the condition is met the next sequential command is skipped; otherwise the next command is executed. The contents of the B register are not changed by this command.

CAL j m Compare A Register and Memory for A Register Less 8 - 9 + x

The contents of memory location M are compared with the contents of the A register. If the contents of the A register are less than the contents of memory location M, the next sequential command is skipped; otherwise the next command is executed. The contents of the A register and memory location M remain unchanged.

CBL j m Compare B Register and Memory for B Register Less 8 - 9 + x

The contents of memory location M are compared with the contents of the B register. If the contents of the B register are less than the contents of memory location M, the next sequential command is skipped; otherwise the next command is executed. The contents of the B register and memory location M remain unchanged.

CAG j m Compare A Register and Memory for A Greater 8 - 9 + x

The contents of memory location M are compared with the contents of the A register. If the contents of the A register are greater, the next sequential command is skipped; otherwise the next command is executed. The contents of the A register and memory location M remain unchanged.

CBG j m Compare B Register and Memory for B Greater 8 - 9 + x

The contents of M are compared with the contents of the B register. If the contents of the B register are greater, the next sequential command is skipped; otherwise the next command is executed. The contents of the B register and memory location M remain unchanged.

CAM j m Compare A Register and Memory 8 - 11 + x

The contents of memory location M are compared with the contents of the A register. If the contents of the A register and the contents of memory location M are equal, the next sequential command is executed; if the contents of the A register are less, the next command is skipped; if the contents of the A register are greater, the next two commands are skipped. The contents of the A and B registers and memory location M remain unchanged.

CME j m Compare A Register and Memory Masked by B for Equal 8 + x

The logical product of the contents of location M and the contents of the B register are compared for equal with the logical product of the contents of the A register and the contents of the B register. If the equal condition is met, the next sequential command is skipped; otherwise the next command is executed. The contents of the A and B registers and memory location M remain unchanged.

SMP j m Skip if Memory is Positive 6 + x

Memory location M is examined to determine if the contents are positive. If the positive condition is met, the next sequential command is skipped; otherwise the next command is executed. The contents of M and registers A and B are not disturbed.

SMZ j m Skip if Memory is Zero 6 + x

The contents of memory location M are examined to determine if they are zero. If the zero condition is found, the next sequential command is skipped; otherwise the next command is executed. The contents of the A and B registers and memory location M remain unchanged.

TLU j m Table Look Up 6 + 6n + x

The contents of the table beginning at location M are searched for a value equal to or greater than the argument in the A register. Once the equal/greater condition is located the command is terminated with the count of the number of entries searched less one in the B register. Since this command only terminates when the equal/greater condition has been found, the last table entry must be greater than or equal to the largest possible argument. The contents of A are not changed.

Logicals and Shifts

Shifts are performed in either the A register, B register, or coupled AB register. The k field (bits 13-20 of the command) indicates the number of places the register contents are to be shifted. The k field may be modified by an index register, but care must be taken to insure that the contents of the designated index register are coincident with the scaling of k.

XOR j m Exclusive Or 5 + x

A logical exclusive or is performed between the contents of memory location M and the contents of the A register. The result replaces the contents of the A register. The final contents of A will have a zero in each bit position where the initial contents of A and the contents of M both have a zero or both have a 1. All other bit positions of the final contents of A will have a 1.

LOR j m Logical Or 5 + x

The logical sum of the contents of location M and the contents of the A register replaces the contents of the A register. The final contents of A will have a 1 in any bit position where either the initial contents of A or the contents of M have a 1.

AND j m Logical And 5 + x

The logical product of the contents of location M and the contents of the A register replaces the contents of the A register. The result will contain a 1 in those bit positions where both the initial contents of A and the contents of M have 1's. All other bit positions will have zeros.

CMA j y Complement A Register 4 + x

The contents of the A register are complemented. If the operand, Y, is non-zero a 1's complement is performed. If Y is zero, a 2's complement is performed.

CMB j y Complement B Register 4 + x

The contents of the B register are complemented. If Y is non-zero, a 1's complement is performed. If Y is zero, a 2's complement is performed.

SSU j m Selective Substitute 10 + x

Selected bits of the A register are substituted into corresponding positions of memory location M where there are 1's in the B register (mask). The result in M also replaces the contents of A. The B register remains unchanged.

RLA j k Rotate Left A Register 4 + n + x

The contents of the A register are shifted left-circular K places. The low-order bits of A are replaced with the high-order bits as the word is shifted.

RLB j k Rotate Left B Register 4 + n + x

The contents of the B register are shifted left-circular K places. The low-order bits of B are replaced with the high-order bits as the word is shifted.

RLC j k Rotate Left Coupled AB Register 4 + n + x

The contents of the coupled AB register are shifted left-circular K places. The low-order bits of the A register are replaced by the high-order bits of the B register and the high-order bits of the A register replace the low-order bits of the B register. This shift includes bit zero of B.

SLA j k Shift Left A Register 4 + n + x

The contents of the A register are shifted left K places. Zeros fill the vacated positions at the right and the bits shifted out of position 0 are lost.

SLB j k Shift Left B Register 4 + n + x

The contents of the B register are shifted left K places, the low-order bits replacing the high-order bits and zeros filling the vacated bit positions. Bits shifted out of position 0 are lost.

SLC j k Shift Left Coupled AB Register 4 + n + x

The contents of the combined AB register are shifted left K places. Bits shifted out of position 1 of the B register are shifted into position 23 of the A register and zeros fill vacated positions of the B register. Bit 0 of the A and B registers does not take part in this shift.

SRA j k Shift Right A Register (6 - 8) + n + x

The contents of the A register are shifted right K places. The sign (bit 0) is extended to the right as the register is shifted and bits shifted out of position 23 are lost.

SRB j k Shift Right B Register (6 - 8) + n + x

The contents of the B register are shifted right K places. The sign (bit 0) is extended to the right as the register is shifted and bits shifted out of position 23 are lost.

SRC j k Shift Right Coupled AB Register (6 - 10) + n + x

The contents of the coupled AB register are shifted right K places; bits shifted out of position 23 of the A register are shifted into position 1 of the B register. Bits shifted out of position 23 of the B register are lost. The sign (bit 0) of the A register is extended to the right as the register is shifted. Bit 0 of the B register does not take part in this command.

Indexing

Address modification is performed through the use of up to eight index registers (j = 0 - 7) which are located in memory cells 64-71. Although the index registers are 24-bit memory locations, the Systems Command Set treats them as 15-bit hardware registers. Each register may be used as a 15-bit address modifier or as a 15-bit counter. All index register arithmetic is performed in 2's complement notation utilizing only the low-order 15 bits (positions 9-23). When using an index register to modify a shift count, k, care must be taken to insure that the least significant bit of the modifier is located in bit position 20 to correspond with the k field.

LDX j M Load Index Register 11 + x

The contents of memory location M (bits 0-23) replace the contents of index register j.

STX j M Store Index Register 13 + x

The contents of the address field of index register j with bit 9 extended into bits 0-8 replace the contents of memory location M. The contents of index register j remains unchanged.

ENX j Y Enter Index Register 7 + x

The 15-bit operand Y with sign (bit 9) extended replaces the contents of index register j.

INX j Y Increment Index Register 3 + x

The 15-bit operand Y is added to the contents of the low-order 15 bits of index register j, and the sum replaces the contents of index register j. The A and B registers are not changed by the command.

XTA j Y Index Register to A Register 6 + x

The contents of index register j are added to the operand Y. The sum, with bit 9 extended, replaces the contents of the A register. The contents of the designated index register remain unchanged.

XTB j Y Index Register to B Register 6 + x

The contents of index register j are added to the 15-bit operand Y. The sum, with bit 9 extended, replaces the contents of the B register. The contents of index register j are not disturbed.

ATX j m A Register to Index Register 3 + x

The contents of the A register replace the contents of index register j. The m field of this command is not used. The contents of A are not changed.

BTX j m B Register to Index Register 3 + x

The contents of the B register replace the contents of index register j. The m field of this command is not used. The contents of B are unchanged.

TSX j M Transfer and Set Index 6 + x

The address of the TSX command plus 1 replaces the contents of index register j, and the next command is taken from location M.

DXT j M Decrement Index and Transfer if Non-Zero 9 + x

The contents of index register j are decremented by 1. If the decremented value is not equal to zero, the next command is taken from memory location M; if the value is zero, the next sequential command is executed. The decremented value always replaces the contents of index register j.

IXS j Y Increment Index Register and Skip 9 + x

The contents of index register j is incremented by 1. If the incremented value of index register j is equal to the 15-bit operand Y, the next command is skipped; otherwise the next command is executed.

Floating-Point Arithmetic

Three types of floating point arithmetic are available in the Systems Command Set. They are 16-bit mantissa (sign plus 15 bit magnitude), 24-bit mantissa (sign plus 23 bit magnitude), and 39-bit mantissa (sign plus 38 bit magnitude) arithmetic.

The floating point formats have an 8 bit exponent field which is biased so that positive and negative exponents are handled. The normal range of floating point numbers is from $1/2 \times 2^{-128}$ to $1 \times 2^{+127}$. The floating point operand formats are shown in Figure II-4.

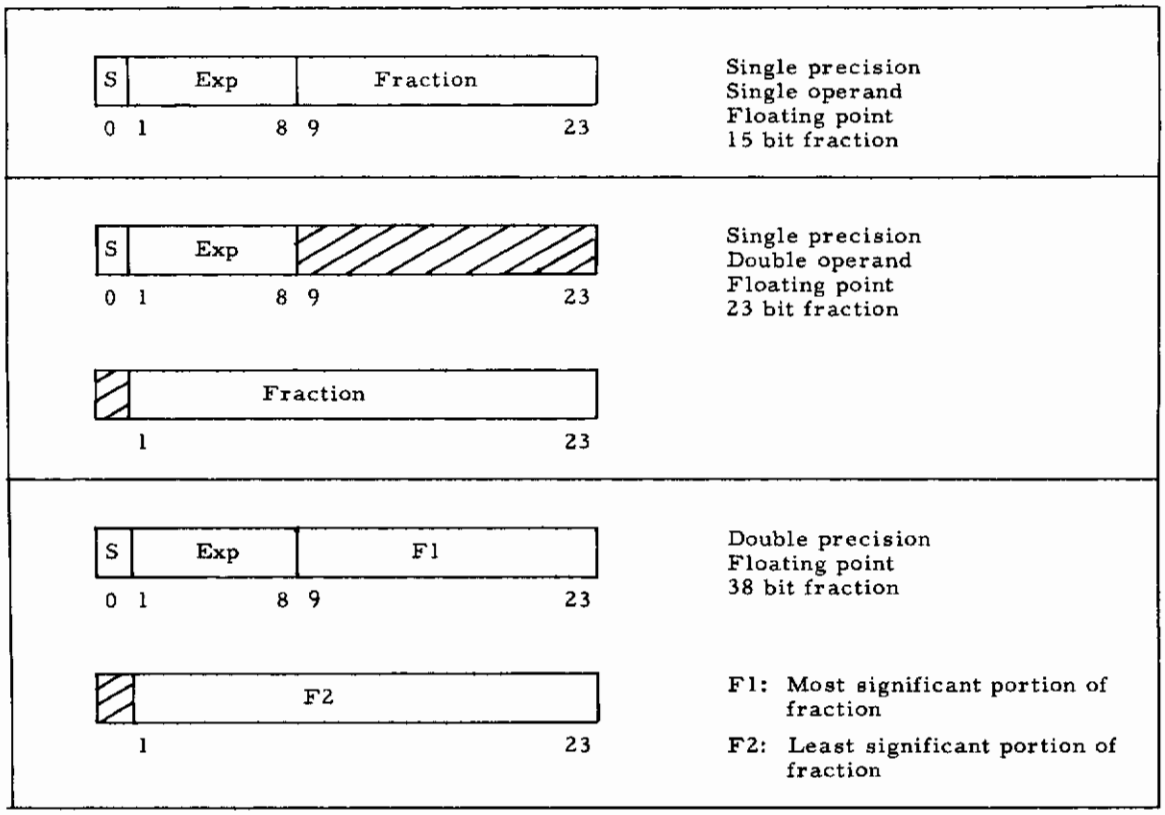


Figure II-4 Floating-Point Operand Formats

In all formats the mantissas utilize binary sign-and-magnitude notation and the exponents are in binary 2's complement form. All floating-point operations utilize both the A and B registers. The 39-bit floating-point is most useful for problems which require a high degree of accuracy in computation. The 24-bit configuration offers the advantage of fast execution times and moderate accuracy. Both of these configurations require two storage locations for each data value. The 16-bit configuration offers the advantage of requiring only one storage location for each data value. This format can be very useful in systems where input and output data are 15 bits or less. In all three of the configurations exponent overflow or division by zero will set an indicator which can be tested by the Floating Transfer on Overflow (FTO) command. The overflow indicator in all floating-point operations is reset only by execution of the FTO command. Program Flag #6 is used as the floating-point overflow indicator. All floating-point numbers are normalized and a floating-point zero will always have a zero exponent and a plus sign. Note that all three floating-point formats may be intermixed within one program. The command descriptions which follow are valid for all three formats. Data in parentheses should be omitted for the single word format and included for double word formats. In all double word floating-point operations the effective execution address M refers to the double word located in M and M + 1. In the expressions given for execution time, n is the number of places shifted or searched for aligning and normalizing operations on the mantissas.

FLD j m Floating Load

16: 4 + x
24: 7 + x
39: 7 + x

The floating-point data word in memory location M (and M + 1) replaces the contents of the A register (and the B register).

FST j m Floating Store

16: 3 + x
24: 6 + x
39: 6 + x

The floating-point data word in the A register (and the B register) replaces the contents of memory location M (and M + 1). The contents of the A and B registers remain unchanged.

FAD j m Floating Add

16: (21 - 33) + n + x
24: (25 - 35) + n + x
39: (24 - 36) + n + x

The floating-point data word in memory location M (and M + 1) is added to the floating-point word in the A register (and the B register). The result replaces the contents of the A register (and the B register). If the sum is zero, the exponent and mantissa fields will both be cleared to zeros. Exponent overflow may result from this command. The sum is always normalized.

FSB j m Floating Subtract

16: (21 - 23) + n + x
24: (25 - 35) + n + x
39: (26 - 38) + n + x

The floating-point data word in M (and M + 1) is subtracted from the contents of the A register (and B register). The difference replaces the contents of the A register (and the B register). If the result is zero, both the exponent and mantissa are cleared. Exponent overflow may result from this command. The difference is always normalized.

FMP j m Floating Multiply

16: (38 - 41) + x
24: (50 - 53) + x
39: (111-117) + x

The floating-point value in the A register (and B register) is multiplied by the floating-point value in memory location M (and M+1). The product replaces the contents of the A register (and the B register). The product is normalized and if zero results, the exponent and mantissa are both cleared. Exponent overflow may result.

FTO j y Floating Transfer on Overflow

16: 4 + x
24: 4 + x
39: 4 + x

If the floating-point overflow indicator is set (program flag 6) the 15-bit operand Y replaces the contents of the L register and the next command is executed from location M, where M = Y. The overflow indicator is reset.

FDV j m Floating Divide

16: (64 - 67) + x
24: (66 - 69) + x
39: (224-229) + x

The floating-point dividend in the A register (and the B register) is divided by the divisor in M (and M+1). The quotient replaces the contents of the A register (and the B register) and is always normalized. This command does not produce a remainder. Division by zero or exponent overflow will set the floating-point overflow indicator.

FIX j k Fix a Floating-Point Number

16: (14 - 18) + n + x
24: (14 - 18) + n + x
39: (14 - 18) + n + x

The floating-point number in the A register (and the B register) is converted to a 47-bit fixed-point number which is in 2's complement notation and has a binary point at K. The resultant fixed-point number replaces the contents of the A and B registers. The binary point of 0 is located between bits 0 and 1 of the A register. Bit 0 of the B register is not used.

FLT j k — Float a Fixed-Point Number — 16: (19 - 21) + n + x; 24: (11 - 13) + n + x; 39: (19 - 21) + n + x

The fixed-point number in the coupled AB register which has a binary point at K is converted to a floating-point number. The result replaces the contents of the A and B registers.

Special Commands and Micro-Coded Subroutines

The Systems Command Set allows the programmer to intermix microsteps with commands and to utilize special purpose commands or micro-programmed subroutines. The following commands provide these capabilities:

EMP j m — Execute Micro-Pair — 5 + PAIR + x

The micro-pair at location M is executed. The A and B registers remain unchanged unless the micro-pair executed create the change.

DML j m — Descend to Micro Level — 2 + x

This command allows micro-programming to be intermixed with normal command programming by exchanging the contents of the L register (Command Address Counter) and the P register (Micro-Program Address Counter). After execution of the DML command and prior to executing the first micro-pair the registers will contain the following:

A Register - Contents from previous command.
B Register - Contents from previous command.
C Register - Contents of index register j; if j = 0, the contents of the C register will depend on the control sequence being used.
D Register - DML command, j designator and effective address M.
L Register - Location of control sequence.
N Register - Unknown

The micro-programming must be ended with an exchange (L and P) micro-step. This will return control to the Control Sequence and set the L register (Command Address Counter) to the address of the next instruction. The j and m field are not needed by this instruction but may be used to set up an effective address in the D register.

NOP j m — No Operation — 2 + x

This command does nothing. The j and m field are not used.

HLT j m — Halt and Transfer — 1 + x

This command causes an unconditional halt. Upon depressing the start key on the console the next command will be taken from location M. The A and B registers remain unchanged.

MRC j m — Microutine Call — 2 + x

This instruction executes the micro-coded subroutine beginning at location M. The registers will be the same as stated for the DML instruction with the exception that the L register (Command Address Counter) will contain the address of the MRC instruction + 1.

SCS j y — Set Control Sequence — (10 - 15) + x

This instruction sets one of two control sequences for execution of all instructions following the SCS instruction. If Y = 0 the fast (no address modification) control sequence is set. The control sequence which is set when $Y \neq 0$ must be selected prior to assembling the program. This instruction will operate only between the fast control sequence and one of the more sophisticated control sequences.

CLM j m — Compare A Register and Limits — (8 - 17) + x

The contents of the A register are compared with the set of limits in memory locations M and M+1. Memory location M must contain the lower limit and M+1 the upper limit. If the contents of A are less than the contents of M, the next sequential command will be executed; if the contents of A are greater than the contents of M+1, the next command will be skipped; if the contents of A are less than the contents of M+1, and greater than the contents of M, the next two commands will be skipped. The contents of A, B, M, and M+1 remain unchanged.

A large number of micro-coded subroutines which may be used with the Systems Command Set are available. These subroutines are called with the MRC command or may be treated as special purpose commands. Some of the more common functions are listed in Figure II-5 along with their storage requirements and execution times.

Micro-Subroutine	Operation	Execution Biax	Time Main	Micro-Pair
SIN	Sine of Angle in the A Register to A	190	285	27
COS	Cosine of the Angle in the A Register to A. (Uses SIN Micro-Subroutine.)	200	300	3+SIN
ATN	Arctan of the Value in the A Register to A	270	350	35
SQR	Square Root of the Value in the A Register to A	144	270	20
PTR	Polar to Rectangular Coordinate Conversion (12-Bit Data)	264	380	40
RTP	Rectangular to Polar Coordinate Conversion (12-Bit Data)	264	380	40
LOG	Natural Log of the Value in the A Register to A	200	310	30
EXP	Exponent of the Value in the A Register to A	280	410	30
GRY	Gray to Binary Conversion (Value in A)	4+6n	5+15n	4
BCD	Binary to Binary Coded Decimal (Value in A)	6	10	3
DTB	Binary Coded Decimal to Binary (Value in A)	6	10	3

Figure II-5 Micro-Coded Subroutines

Input-Output Commands

Input-output data are transferred in blocks one word at a time. A data word is normally a full 24-bits although some controllers accept and transmit words of fewer bits. (For example, the paper tape reader transmits 8-bit characters.) For characters, or for word formats with less than 24-bits, the low-order bits of a full computer word are used in the data transmission. Up to four independent input-output block transfer operations may take place concurrent with program execution.

Block transfers are initiated by an input (INP) or output (OUT) command which puts a command word on the computer I/O bus. This command word contains the information necessary to control the selected peripheral device. Each of the four communication channels has five main memory locations assigned as control words (Figure II-6). Two of these locations are assigned for use as interrupt registers, two are used for initial and terminal address control, and another location is reserved for status response words. Also, program flags 1-4 are used as channel-active indicators for the four communication channels. Memory locations 120_{10} - 127_{10} are used as temporary storage for the hardware registers whenever an automatic interrupt occurs.

Before the initiation of an input or output operation the initial and terminal address control words must be loaded by the programmer. The initial address must contain the first address of the buffer to be used, while the terminal address must contain the last address of the buffer plus one. Once a buffer operation has been initiated with an INP or OUT command, the initial address control word will contain the address of the next data word in the buffer. The program may interrogate this word at any time in order to determine the progress of a buffered operation. Also, any buffer operation may be terminated prematurely by setting the initial address control word to the last address of the buffer.

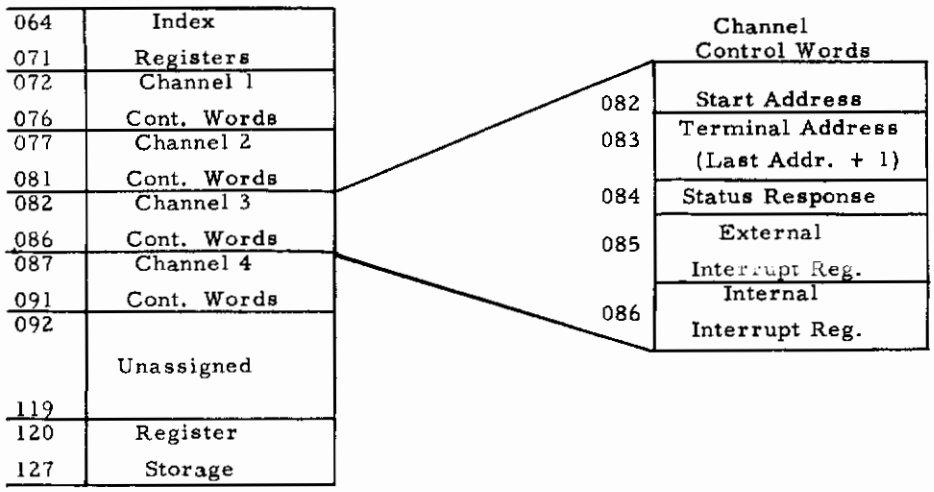


Figure II-6 Channel Control Words

The initiation of a buffer operation will set the appropriate channel-active indicator (program flag) true. Termination of a buffer operation will set the appropriate program flag false. The condition of the program flags may be tested with the SBI (Skip on Buffer Inactive) command. The termination of a buffer operation will always produce an internal interrupt condition. An internal interrupt will also occur if an error condition occurs in the course of a data transmission.

INP j m Input

An input buffer operation is initiated according to the command word in memory location M. The contents of the status response word specified by the channel number of the reference command word are replaced by the status response returned from the selected peripheral device. If the referenced command word contains a zero in bit 0, the appropriate channel-active indicator will be set true and an input buffer operation will be initiated. With a 1 in bit 0 only a status response will result.

OUT j m Output

An output buffer operation is initiated according to the command word in memory location M. The contents of the status response word specified by the channel number of the reference command word are replaced by the status response returned from the selected peripheral device. If the referenced command word contains a zero in bit 0, the appropriate channel-active indicator will be set true and an output buffer operation will be initiated. With a 1 in bit 0 only a status response will result.

SBI o y Skip on Buffer Inactive

The operand Y, which must equal 1, 2, 3, or 4, specifies the channel-active indicator (program flag 1, 2, 3, or 4) to be tested. If the channel specified by Y is inactive (the program flag is false), the next sequential command is skipped. If active, the next sequential command is executed.

Interrupts

Two types of interrupts are defined by the System Command Set. An internal interrupt occurs whenever a buffered input-output operation is terminated. An external interrupt results whenever an external device places a signal on the interrupt line of a particular channel at a time when that channel is inactive (the program flag associated with the channel is false). The occurrence of either of these interrupts on a given communication channel will cause the following steps to take place:

1. The interrupt commutator will be locked on the communication channel which produced the interrupt.

2. Execution of the micro-pair contained in the E register will be completed.

3. The contents of hardware registers A, B, C, D, P, L, and N will replace the contents of main memory locations 121_{10} - 127_{10}; the contents of carry toggles 0, 1, and 9 will be stored in bits 0, 1, and 9 of main memory location 120_{10}.

4. The command in the specified interrupt register will be executed.

Associated with each of the four communication channels are an internal interrupt register and an external interrupt register. These memory locations must contain either an unconditional transfer to an interrupt subroutine or an IRL (Interrupt Release) command. This command is described below:

IRL j y Interrupt Release

The interrupt commutator is released by this command. If the operand Y is non-zero, the contents of the eight working storage locations (120_{10} - 127_{10}) replace the contents of the PB440 hardware registers and carry toggles before the interrupt commutator is released. This operation returns the program to the condition which existed when the interrupt occurred. If the operand Y is zero, the registers are not restored and the next sequential command is executed.

Whenever an internal or external interrupt occurs, the interrupt commutator is locked on the communication channel which produced the interrupt. No further interrupts can be acknowledged until the interrupt commutator has been released. Storing an IRL with a non-zero operand will have the effect of ignoring an interrupt.

C. INTERPRETIVE CONTROL SEQUENCE

Basically the PB440 is a micro-coded computer where the wired-in sequence and control respond only to micro-steps. The instructions in the SCS list contained in the preceeding section cannot directly control the computer; control is only through microutines.

The SCS assembler supplies a coded microutine for each unique command employed by the programmer. These microutines (maximum limitation of 64) are loaded into fast memory together with a command interpreter or control sequence.

The control sequence is an interpretive system which picks up a command, computes an effective operand or operand address, and branches to the appropriate microutine. For maximum performance, several control sequences are provided; the most complex being capable of indexing a direct and indirect address, the simplest, direct addressing only.

The most general control sequence permits address modification by indexing and indirect addressing. Three possible modes of address modification are identified by the index designator:

1. When j = 0 no address modification takes place. The execution time for the control sequence is 6 microseconds.

2. When j = 1 - 6 the execution address is modified by the contents of the designated index register. The effective address is equal to the sum of the base execution address and the contents of the designated index register. Two's complement arithmetic is used in determining the modified address. The control sequence execution time for this mode is 11 microseconds.

3. When j = 7 indirect addressing is designated. In this mode a storage reference is made to the location designated by the base execution address. The low-order 18 bits of the word read from storage are interpreted as the j designator and the execution address of the present instruction. The new index designator may refer to any one of the three modes. The control sequence execution time for this mode is 11+7n microseconds.

The basic flow of this control sequence is shown in Figure II-7.

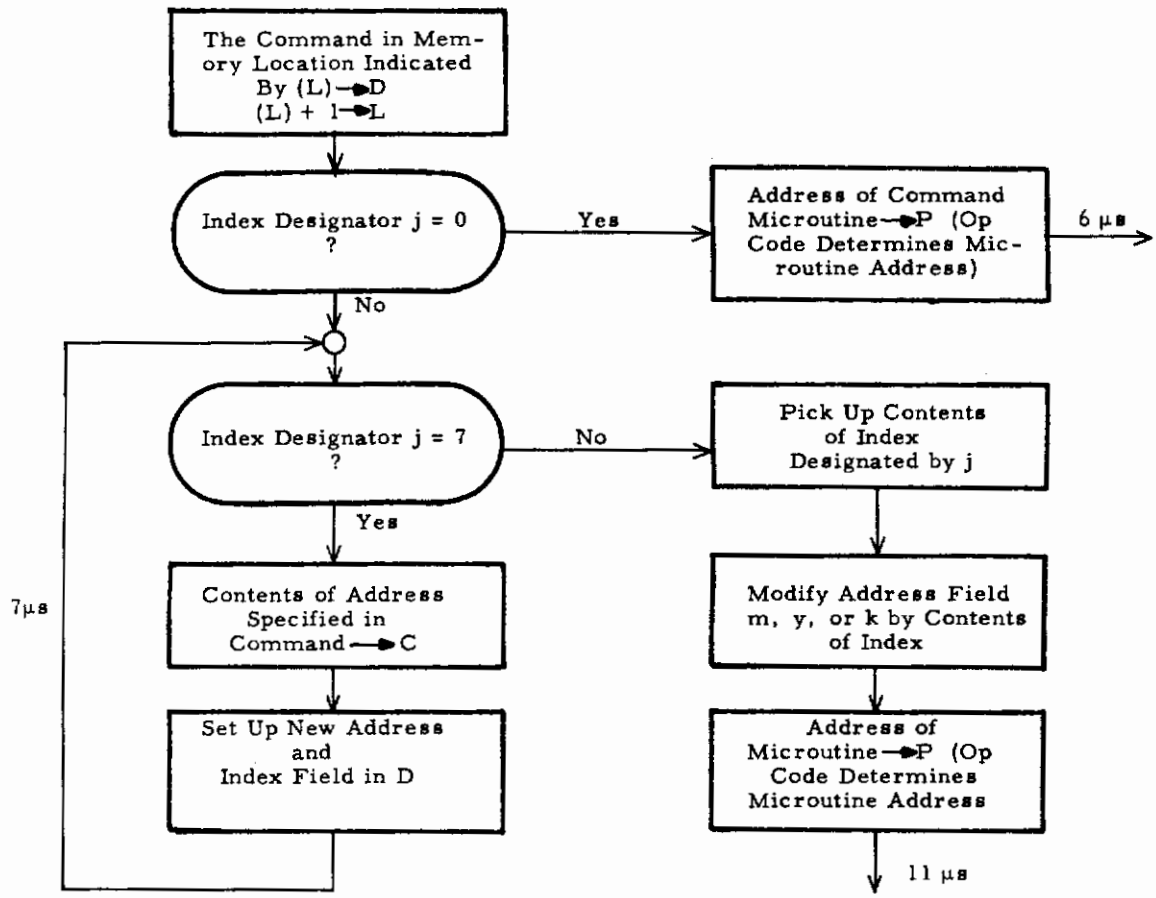


Figure II-7 General Control Sequence

If indirect addressing is not required, index designator 7 may designate another index register. There are two control sequences designed for use with this configuration. The first operates much in the same manner as the general control sequence described earlier and is shown in Figure II-8. However, index designator j = 7 is interpreted as indexed addressing instead of indirect addressing. The control sequence execution time for j = 0 is 6 microseconds and for j = 1 - 7 is 10 microseconds.

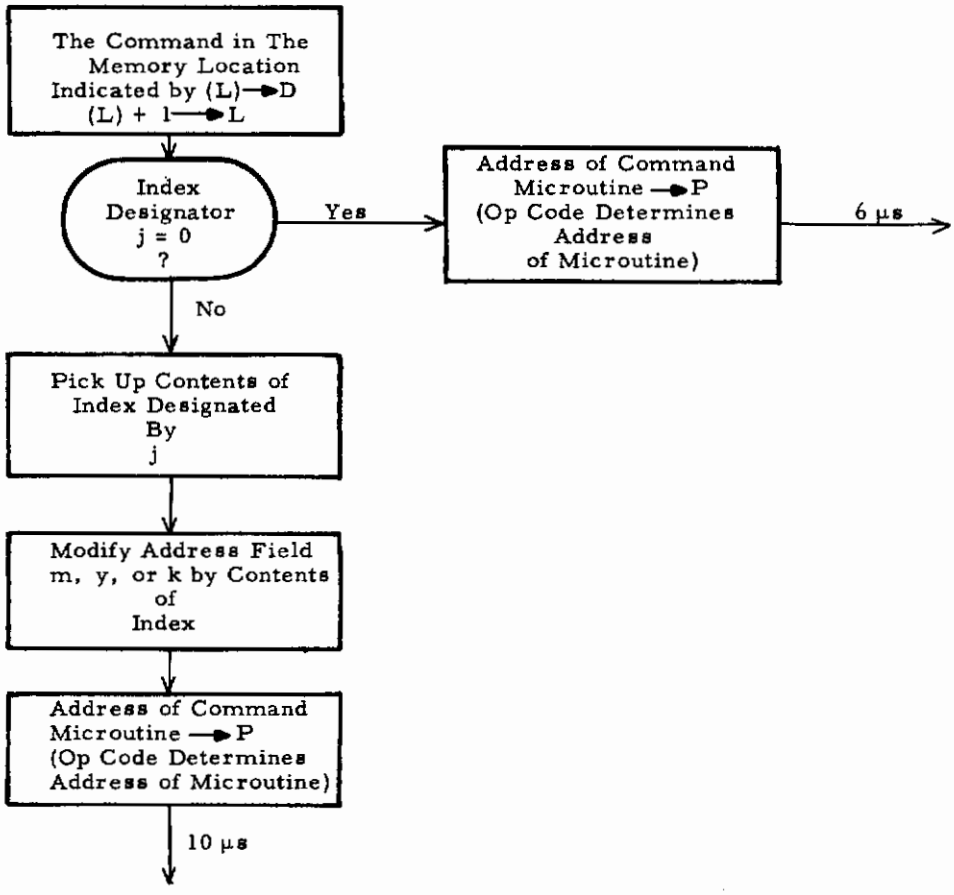


Figure II-8 Index Control Sequence (Option 1)

The second control sequence for this index designator configuration takes advantage of the more rapid memory references which can be made in computer configurations with 2 or more main memory modules. All index designators, including j = 0, result in effective address modification by the designated index register. In this mode index register 0 must always contain zeros so that direct operand and address references can be made. The programmer must therefore refrain from changing the contents of index register 0. The control sequence, which is illustrated in Figure II-9, requires 7 microseconds to execute. Its use can prove very effective in routines and programs which require extensive indexing.

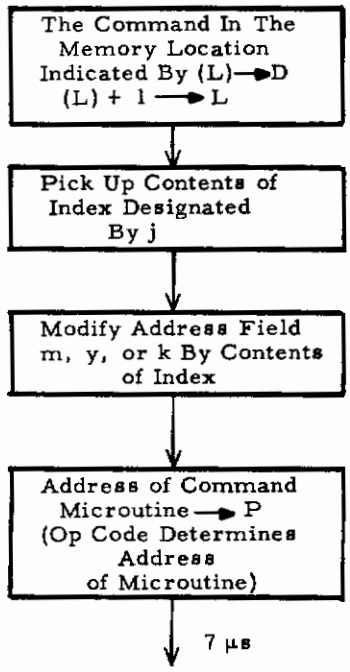


Figure II-9 Index Control Sequence (Option 2)

For programs and routines which do not require indexing and indirect addressing a 4-microsecond control sequence is provided. This is the fastest control sequence available. The flow diagram in Figure II-10 illustrates this operation. The index designator field has no effect on the command operands. Commands which manipulate and test the index register contents cannot be used with this control sequence.

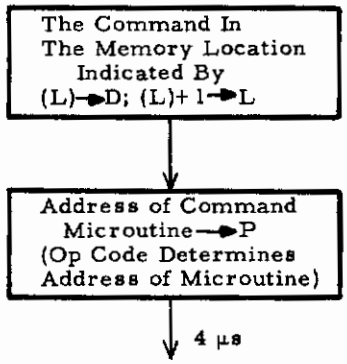


Figure II-10 Fast Control Sequence

The choice of control sequence for a particular program is a function of the speed and addressing requirements of the problem. Normally, one of the above sequences will satisfy both of these requirements. However, the more sophisticated programmer may choose to use the fast 4-microsecond control sequence in conjunction with one of the others. This can be accomplished by using the command, Set Control Sequence (SCS). In this way subroutines and parts of the program which do not require address modification can take advantage of the fast control sequence while other parts of the program may utilize the address modification capabilities of a more sophisticated control sequence.

D. SAMPLE SCS COMMAND MICROUTINES

This section presents a selection of the microutines employed in the Systems Command Set. The study of these routines is not required for a working knowledge of the SCS. However such a study will be very useful if microcoding is to be mastered to fully utilize the potential of the PB440.

1. ADA: (A) + (M)—►A

Location	Time	OP.	R1	R2	Comment
ADA	2	LDM	D	C	Data—►C
	1	ADL	C	A	(A) + (C)—►A.

This instruction is followed immediately by the control sequence; thus a transfer to the control sequence does not require a CPL.

2. SBA: (A) - (M)—►A

Location	Time	OP.	R1	R2	Comment
SBA	2	LDM	D	C	Data—►C.
	1	CCL	C	C	Complement C.
	1	CIL	C	C	Increment C.
	1	ADL	C	D	(A) + (-C)—►A.
	1	CLP	CSQ		Return to control.
	1	NOP			

3. TOV: Transfer on Overflow

Location	Time	OP.	R1	R2	Comment
TOV	1	AFK	N	P	Increment P by overflow from 1.
	2	TCF	CT	O	If C to 1, skip.
	1	CLP	CSQ		Return to control.
	1	NOP			No overflow.
	1	CLP	CSQ		Return to control.
	1	CPF	D	L	Transfer control to M.
	1	CLP	CSQ		Return to control.
	1	NOP			No overflow.

The TOV instruction causes a transfer to location M if there was an overflow from a preceding arithmetic operation such as ADA or SUA above. It depends for its working on the fact that such an overflow is signalled by carry toggles 1 and 0 having unequal values. In this case, when control is returned to the control sequence, L, the macro-instruction counter, is altered to take the next macro from M.

4. LDA: (M)—►A

Location	Time	OP.	R1	R2	Comment
LDA	2	LDM	D	A	Data word to A.
	2	CLP	CSQ		Return to control.

5. TRA: M—►L (L is location counter)

Location	Time	OP.	R1	R2	Comment
TRA	1	CLP	CSQ		Return to control.
	1	CPF	D	L	Set L to address of next macro.

The TRA instruction causes the next macro-instruction to be taken from the effective address; hence it causes the macro-instruction sequence to branch unconditionally.

6. ENX: $M \longrightarrow X_i$
   INX: $M + (X_i) \longrightarrow X_i$

Location	Time	OP.	R1	R2	Comment
ENX	1	CCL	C	C	Subtract from address portion the index
	1	ADF	C	D	added by the control sequence.
	1	CIL	D	D	
	1	NOP			
INX	1	STS	O	D	Store address portion in index.
	2	CLP	CSQ		Return to control.

The ENX instruction stores the address portion of the instruction in the index register designated with the instruction. Note that the control sequence adds the contents of the index register to the address portion of the instruction, and hence it is necessary in ENX to first restore the original address. INX simply stores the address portion of D in X_i since the control sequence has already formed $(M + X_i)$ in D. Since this operation is identical to the last part of the ENX instruction, the two can be made to overlap saving a micropair storage location.

7. XOR: $(A\overline{M}) + (\overline{A}M) \longrightarrow A$ (Exclusive Or)

Location	Time	OP.	R1	R2	Comment
XOR	2	LDM	D	C	Operand$\longrightarrow$C.
	1	XOR	C	A	Form logical result in A.
	1	CLP	CSQ		Return to control sequence.
	1	NOP			

8. SLA: Shift A Left Open Logical

Location	Time	OP.	R1	R2	Comment
SLA	1	SLS	D	D	Shift D left six.
	1	SLS	D	N	Shift D left six, place count in N.
	1	CLP			Set return to control sequence.
	1	SSL	LOL	A	Shift A left N places.

The SLA instruction has the shift count located in bit positions 15 through 20 of the address field. This unusual field assignment has the advantage of decreasing the execution time of the shift instructions by permitting the shift count to be loaded into the N register in only two microseconds. If the shift count had been right justified in the address field, an additional 4 microseconds and two micropairs are required in each of the shift commands.

9. MPY: $M * B \longrightarrow A$ Multiply

Location	Time	OP.	R1	R2	Comment
MPY	2	LDM	D	C	Multiplier to C.
	1	XOR	A	A	Clear A.
	1	LRC	2	3	Load N with 23.
	2	TNZ	B	S	If B is positive skip.
	1	CCM	B	B	Complement B logical.
	1	CIL	B	B	Increment B.
	1	ADS	C	B	Add signs $C_o + B_o \longrightarrow B_o$.
	2	TNZ	C	S	If C is positive skip.
	1	CCM	C	C	Complement C logical.
	1	CIL	C	C	Increment C.
	23	MPS	C	A	Multiply.
	1	CPS	B	A	Copy sign of B to A.
	1	TZO	A	S	If product minus, skip.
	2	CLP	CSQ		Return to control sequence.

Location	Time	OP.	R1	R2	Comment
MPY	1	CCM	A	A	Complement A.
	1	CCM	B	B	Complement B.
	1	CIL	B	B	Increment B.
	1	AFK	N	A	If B was zero, increment A.
	1	CLP	CSQ		Return to control sequence.
	1	NOP			

The MPY multiplies the contents of the B register and the operand in memory location M, placing the 47 bit product in the A and B registers combined, with the most significant portion in A. Bit 0 of B is ignored in the product.

10. DIV: AB ÷ M→B; Remainder in A Divide

Location	Time	OP.	R1	R2	Comment
DIV	2	LDM	D	C	Divisor to C.
	1	CCL	C	D	Divisor complement to D.
	1	TZO	C	S	If C minus, skip.
	1	CIL	D	C	2's Complement to C.
	1	CCM	B	D	Least sig. half of dividend to D.
	1	TNZ	A	S	If dividend plus, skip.
	1	CIL	D	B	2's complement to B.
	1	CCM	A	A	Complement most sig. half of dividend. (Sign and magnitude in AB)
	1	AFK	N	A	If B was zero, increment A.
	1	CCS	D	B	Dividend sign to B.
	1	ADS	A	B	
	1	ALC	A	C	Trial subtraction.
	1	TCT	CT	1	If A ≥ C, skip.
	1	STF	CT	0	Clear CTO
	1	LRC	2	3	23 to N.
	46	DVS	C	A	Divide.
	1	CPS	N	A	Set A plus.
	1	CDL	B	D	Quotient - 1 to D.
	1	TZO	B	S	If B minus, skip.
	2	CLP	CSQ		Return to control sequence.
	1	CLP	CSQ		Set return to control sequence.
	1	CCM	D	B	2's complement to B.

The double length dividend in A and B is divided by the operand in memory location M. The quotient is in B at the completion of the command.

APPENDIX III

COMPLEMENT ARITHMETIC ON THE PB440

When microcoding arithmetic commands, the programmer must have a good working familiarity of the number system he is employing as well as the micro-instruction operations. The material presented in this brief appendix supplements the micro-programming information in Appendix I and the Systems Command Set arithmetic command descriptions given in Appendix II by presenting the descriptions of the two complement number systems commonly encountered in binary computers: one's complement and two's complement. The Systems Command Set is restricted to two's complement, but to take full advantage of the microcoding capability of the PB440, the programmer should know both systems.

A. ONE'S COMPLEMENT ARITHMETIC

The essential feature of one's complement arithmetic is that negative numbers are written as the logical complement of their positive counterparts. Thus, taking for example a 7 bit register, with the first bit a sign bit, +5 would be represented as

0 000 101,

and -5 would be represented as

1 111 010.

A +0 would be represented as

0 000 000,

and -0 would be represented as

1 111 111.

Another way to describe one's complement representation is as follows: in a k-bit register, a negative number (-n) is represented by the number

$2^k - n - 1.$

This is equivalent to the first description; since the logical complement of n may be formed by subtracting n from a word of all ones. But in a k-bit register a word of all ones is the number $2^k - 1$. Hence the logical complement of n is $2^k - 1 - n$.

The above form of number representation is well suited to the PB440 since this machine lacks actual hardware for subtraction, and must combine addition and logical complementing to perform this operation. Unfortunately operations with this form of number are not perfectly simple, in that the logical sum of 2 one's complement numbers is not necessarily the one's complement form of their sum. Hence the algorithms for performing arithmetic on these numbers efficiently must be kept in mind when microcoding the arithmetic operations. In order to understand microcoded arithmetic operations, it is necessary to read thoroughly the paragraphs describing the relevant micro-steps, and also to grasp the concept of "Carry Toggles" as they are described in Appendix I. For the reader's benefit, the part about carry toggles is reproduced here.

"The carry toggles are set by certain micro-steps automatically, and are designed to indicate carry-out from bit positions, 0, 1, and 9. That is, a carry toggle is made true (or "1") if carry-out occurs from its bit position, or made false if no carry-out occurs. They may be set and tested individually as well."

Generally the only instructions which alter the carry toggles are those which cause addition or incrementation, together with the Set Toggle True, Set Toggle False, and Copy to Special micro-steps. Instructions which set carry toggles in the course of addition are given in Figure III-1 below.

Micro-step	Carry toggles which may change state in the operation		
CIL	CT0	CT1	CT9
ADL	CT0	CT1	CT9
ALC	CT0	CT1	CT9
ADS	CT0		
ADM		CT1	CT9
AMK		CT1	CT9
CIX		CT1	
ADX		CT1	
ADF			CT9
AFK			CT9
CTS 4	CT0	CT1	CT9
CTS 7	CT0	CT1	CT9
STT 2	CT0	CT1	CT9
STF 2	CT0	CT1	CT9

Figure III-1 Instructions Affecting Carry Toggles

The Add Magnitude with Carry-in and Add Fraction with Carry-in operations should be specially noted; their function depends on the contents of the carry toggles at the start of the operation, and certain carry toggles are then reset by the operation itself. The condition of the carry toggles may be tested at any time by the micro-steps Test Condition True and Test Condition False. The contents of the carry toggles may be stored by Copy From Special.

The carry toggle which records carries from bit position 0, the sign bit, will be called Carry Toggle 0. The one which records carries from bit position 1 will be called Carry Toggle 1. We will not be concerned here with the other carry toggle.

It is first necessary to describe the result of performing the ADL micro-step, described in Appendix I, on various numbers. The essential results are presented in Figure III-2. These will be clarified below. The column marked "overflow" should perhaps be explained first.

The highest number that can be represented in a k-bit register with one's complement representation is $2^{k-1} - 1$. The lowest number is $-(2^{k-1} - 1)$. Any addition or subtraction which produces a result which lies outside these limits is said to give rise to overflow. In the last three entries in the overflow column, where the two operands A and B have opposite signs, there is no possibility of overflow and the space is used to distinguish the cases where the relative magnitudes of the positive and negative terms differ.

Sign A	Sign B	Overflow	Result	CT-0	CT-1
+	+	no	Correct	0	0
+	+	yes	Correct 24 bit magnitude	0	1
-	-	no*	Correct 1's comp. -1	1	1
-	-	yes	Correct 24 bit 1's comp. mag. -1	1	0
+	-	$\|A\| > \|B\|$	Correct -1 (a positive no.)	1	1
+	-	$\|A\| < \|B\|$	Correct (a negative no.)	0	0
+	-	$\|A\| = \|B\|$	-0, Correct	0	0

*An exception to this occurs with the unique case of full negative magnitude result which is considered to be an Overflow.

Figure III-2 Overflow Cases

A line-by-line explanation of this table follows. The reader should bear in mind that ADL A B will add the entire contents of the A register logically (i.e., as if the whole word were a single binary number, without regard to sign) to the B register, and record carries from positions 0 and 1 in the appropriate carry toggles.

Line 1. Sign A +, Sign B +, no overflow.

In this case the logical sum is equal to the algebraic sum. Since the result is equal to or less than $2^{k-1} - 1$, there is no carry from position one. Since both signs are +, there is no carry from position zero.

Example:	+ 5	0 000 101
	+ 7	0 000 111
	+12	0 001 100

Line 2. Sign A +, Sign B +, overflow.

In this case the overflow bit from position 1 is left in the sign position, giving a correct 24-bit magnitude result. Carry Toggle 1 contains a bit, and carry toggle zero does not.

Example:	+ 7	0 000 111
	+59	0 111 011
	+66	1 000 010

Line 3. Sign A -, Sign B-, no overflow.

In this case we can represent the contents of A as $(2^k - a - 1)$, and those of B as $(2^k - b - 1)$. The result of the ADL instruction will then be the k low-order bits of the quantity

$$2^k + 2^k - a - b - 2,$$

which are the same as the k low-order bits of the quantity

$$2^k - a - b - 2.$$

The correct one's complement result of adding (-a) and (-b) is

$$2^k - a - b - 1.$$

Hence we say that the result is the correct one's complement minus one. Both A and B have bits in the sign position; hence there is a carry from position zero. Since (-a -b) is $\geqslant -(2^{k-1} - 1)$, we have

$$2^k - a - b - 2 \geqslant 2^k - 2^{k-1} + 1 - 2 = 2^{k-1} - 1$$

Suppose for the moment that the " $>$ " sign holds in the above inequalities. Then the result left in the B register is $\geqslant 2^{k-1}$, and hence there is a bit in the 0-position. This bit must have carried out from bit position 1, and hence carry toggle 1 is set.

In the case where $(-a\ -b) = -(2^{k-1} - 1)$, carry toggle 1 is not set, as is seen in the example below. This results in full negative magnitude being treated as overflow. If it were imperative to be able to handle this quantity also, a slightly longer add instruction would be needed than the ADA to be described later.

Example:	- 5	1 111 010
	- 7	1 111 000
	-13	1 110 010
Example:	-40	1 010 111
	-23	1 101 000
	+63	0 111 111

Line 4. Sign A -, Sign B -, overflow.

In this case the quantity left in the B register is again

$$2^k - a - b - 2$$

where here $(-a\ -b) < -(2^{k-1} - 1)$.

Hence

$$2^k - a - b - 2 < 2^k - 2^{k-1} - 1 = 2^{k-1} - 1.$$

Therefore the sign position does not have a bit in it, indicating no carry from position 1. There is a carry from position 0 since both signs are minus. We see that the result is one less than a correct one's complement 24-bit magnitude.

Example:	- 7	1 111 000
	-59	1 000 100
	(-67)	0 111 100

Line 5. Sign A +, Sign B -, $|A| > |B|$

In this case the result left in the B register is

$$2^k + a - b - 1.$$

Since $0 < (a - b) < 2^{k-1}$, the 2^k does not show and the result is the correct one minus one. The positive sign in the result must result from a carry from the 1 position, and produce a carry from the 0 position.

Example:	+ 5	0 000 101
	- 3	1 111 100
	+ 1	0 000 001

Line 6. Sign A +, Sign B -, $|A| < |B|$

In this case the result left in the B register is

$$2^k + a - b - 1, \text{ where } 0 > (a - b) \geq -(2^{k-1} - 1).$$

Hence $2^k - 1 > 2^k + a - b - 1 \geq 2^{k-1}$.

Therefore the result has a bit in its sign position, indicating no carry from positions 1 or 0. The result is seen to be a correct one's complement.

Example:	+ 3	0 000 011
	- 5	1 111 010
	- 2	1 111 101

Line 7. Sign A +, Sign B -, $|A| = |B|$

In this case the result is clearly a word of all 1's, with no carries

Example:	+ 3	0 000 011
	- 3	1 111 100
	- 0	1 111 111

The significance of the above table is that it shows that the carry toggles hold all the information necessary for modifying the result of an addition. In particular;

1) An overflow condition is signalled by the presence of unlike conditions in Carry Toggles 1 and 0. Here a result is considered overflowed if it results in full negative magnitude.

2) The necessity for adding one to a result is signalled by the presence of a bit in Carry Toggle 1, again except in the case of full negative magnitude or overflow.

Applications of the above are shown in the following microroutines.

1. The ADA Microroutine

This microroutine adds the contents of the addressed memory location to the A register.

				Time
1.	LDM	D	C	2
2.	ADL	C	A	1
3.	CLP	"CSQ"		1
4.	AFK	Q	A	1
				5

Line 1. The address portion of the D register is used to obtain a word from memory, which is loaded into the C register. Since the control sequence caused the original macro-instruction to be put into the D register, the address portion of the D register consists of the location of the desired operand.

Line 2. The contents of the C register, which now hold the desired operand, are added logically to the contents of the A register.

Line 3. The address of the control sequence, here symbolized by "CSQ", is copied into the P register. As a result of this the next micro-pair to be inserted in the E register is the first micro-pair of the control sequence, causing a branch to the control sequence after the present micro-pair is executed. Lines 3 and 4 could be reversed, with the same logical function resulting, but the micro-pair would then take one microsecond longer since the right half would change the P register thus delaying the next micro-pair fetch.

Line 4. This operation causes

1) the fraction portion of the Q register to be added to the fraction portion of the A register, and

2) the contents of Carry Toggle 1 (at the start of the operation) to be added to the low order position of the A register.

Part 1 has no effect, since the Q register contains all zeros. Part 2 causes the A register to be incremented if and only if there was a carry from position one during the ADL operation. As was shown in the previous section, this is just the operation that is needed to modify a one's complement result. It is the ability to test and modify a result in this one step that makes one's complement addition more efficient than sign and magnitude addition on the PB440.

2. The SUB Microroutine

This microroutine subtracts the contents of the addressed memory location from the A register.

				Time
1.	LDM	D	C	2
2.	CCL	C	C	1
3.	ADL	C	A	1
4.	AFK	Q	A	1
5.	CLP	"CSQ"		1
6.	NOP			1
				7

This microroutine is the same as the ADA microroutine, with two exceptions,

a.) In line 2, after the operand is brought into the C register, it is logically complemented, which in one's complement arithmetic is the same as changing the sign of the quantity. This results in its being subtracted from, rather than added, to the A register.

b.) In line 6, a NOP appears, because although there is no more useful work to perform, there must nevertheless be a right micro-step. (Note that if steps 5 and 6 are reversed the micro-pair would require 3 microseconds instead of 2.)

3. A MPY Microutine

This microutine multiplies the contents of the specified memory location by the contents of the B register and leaves the result in the A and B registers. It should be noted that the Packard Bell SCS MPY command employs 2's complement notation rather than 1's complement as used here.

	OP	R1	R2	Time
1.	LDM	D	C	2
2.	CLN	23		1
3.	TNZ	B	S	1
4.	CCM	B	B	1
5.	ADS	C	B	1
6.	CPS	B	A	1
7.	TNZ	C	S	1
8.	CCL	C	C	1
9.	CPM	Q	A	1
10.	MPS	C	A	23
11.	TZO	A	S	1
12.	CLP	"CSQ"		1-2
13.	CCM	A	A	1-0
14.	CCM	B	B	1-0
15.	CLP	"CSQ"		1-0
16.	NOP			1-0
				36-39

Line 1. This brings the operand into the C register.

Line 2. This loads the N register, or Repeat Counter, with the number of digits to be used in the multiplication.

Line 3. The step following this will be executed only if B is -.

Line 4. If B is -, its magnitude portion is here changed to absolute value.

Line 5. This adds the sign of the C register to that of the B register. If the signs are the same a + sign will result; if the signs were different a - sign will result. Hence the correct sign of the result is formed in the B register.

Line 6. The sign of the result is put into the A register also.

Line 7 and 8. The same as 3 and 4, applied to C.

Line 9. Clear the A register (except for the sign).

Line 10. Perform the multiplication of absolute values.

Line 11. The step following this is performed only if the result is +.

Line 12. If the result is +, it is now correct. Return to control.

Line 13 and 14. If the result is -, form the one's complements of the magnitudes.

Line 15 and 16. Return to control.

4. The DIV Microutine

This microutine divides the contents of the A register and the magnitude portion of the B register, considered as a single double precision number, by the contents of the addressed memory location. The quotient is left in the B register in one's complement form if negative.

	OP	R1	R2	Time
1.	LDM	D	C	2
2.	CPS	A	B	1
3.	ADS	C	B	1
4.	TNZ	A	S	2
5.	CCL	A	A	1 (0)
6.	CCM	B	B	1 (0)
7.	TZO	C	S	1
8.	CCL	C	C	1
9.	CIL	C	C	1
10.	CLN	23		1
11.	DVS	C	A	46
12.	TZO	B	S	2
13.	CLP	"CSQ"		1 (0)
14.	NOP			1 (0)
15.	CLP	"CSQ"		1 (0)
16.	CCM	B	B	1 (0)
				60-62

Line 1. Place the operand in the C register.

Line 2. Place the sign of the A register in the sign position of the B register.

Line 3. Add the sign of the operand to the sign of the dividend. The result is the sign of the quotient.

Line 4. The next micro-pair is skipped if the sign of A is +.

Line 5 and 6. If the A and B registers were negative they are converted to absolute values. The sign of the quotient, however, is preserved in the B register.

Line 7. Skip the next micro-step if the C register is negative.

Line 8. Form the one's complement of a positive operand.

Line 9. Form the two's complement of the divisor.

Line 10. Load the N register, or repeat counter, with the number of division steps to be made.

Line 11. Perform the division.

Line 12. Test the sign of the result.

Line 13 and 14. If the result is positive, return to the control sequence.

Line 15 and 16. If the result is negative, return to the control sequence and form the one's complement of the quotient.

B. TWO'S COMPLEMENT ARITHMETIC

A number n in a k-bit register, where $|n| \leqslant 2^{k-1} - 1$, is represented in the 2's complement system by the quantity n if n is positive and by $2^k - n$ if n is negative. Since ADL essentially performs addition (mod 2^k), it follows that ADL applied to two 2's complement quantities will give the 2's complement of the sum of the quantities, unless there is overflow, without the necessity for endaround carries. Hence, addition of 2's complement numbers is simpler than 1's complement. On the other hand, negation in the PB440 requires two steps, CCL and CIL, instead of one. As in one's complement, overflow will be indicated by unlike conditions in carry toggles zero and one.

A peculiarity of the 2's complement system is that $-(2^{k-1})$ can be represented in a k-bit register, but not $+(2^{k-1})$. In the SCS system, $-(2^{k-1})$ is not allowed as a representable quantity since it does not have a negative, and since numbers which sum to give this quantity leave an overflow indication in the carry toggles.

APPENDIX IV

FLIGHT SIMULATION SAMPLE PROGRAMS

This appendix contains sample programs, pertinent to the F100 simulation program, for the DDP24 and PB440 computers. These programs indicate the comparative computational rates of the two computers.

Some insight into micro-programming may be gained by close examination of the microutines included in this appendix. In addition, an appreciation for the operating rate gained by micro-coding subroutines may be realized by comparing the SCS and DAP computing times with the microutine times.

The microutines in this appendix may contain errors due to: (1) possible misinterpretation of the micro-instruction description and (2) the lack of extensive familiarity with the PB440 computer. It is likely that operational experience with the PB440 would reveal more efficient methods of using the micro-coding capabilities.

A. DIRECTION COSINES FOR DAP AND SYSTEMS COMMAND SET

1. Introduction

The following routine is the "Direction Cosines" portion of the F100 simulation recoded in DAP for the DDP24, with times also given for analagous operations with the PB440 Systems Command Set. It is included as an example of a fairly lengthy section of routine calculation in these two different systems. The control time for the Systems Command Set is taken to be 5 microseconds. Times are also given for the SDS 9300 in terms of machine cycles, with total time computed by the formula in Appendix V. (It should be noted that there is a slight difference in the operation of the multiply instructions of the DDP24 and the SDS 9300, in that the latter uses the A register instead of the B register as a multiplier. The only alteration this would cause in the following program would be to change a number of pairs "LDB, MPY" to "LDA, MUL" and eliminate 3 "TAB" operations.)

A good description of the direction cosines routine may be found in Krasny (Reference 5). Since we are interested here primarily in comparing execution times we will say only that this routine:

1) calculates 6 derivatives of direction cosines, in terms of previous direction cosines and present angular velocities, by formulas of the form

$$l_1 = rm_1 - q_1 n_1,$$

2) integrates these 6 derivatives by branching to an integration routine 6 times,

3) normalizes the results of these integrations by multiplying by an appropriate factor, and

4) generates the remaining 3 direction cosines from these 6 normalized integrands by formulas of the form

$$l_3 = m_1 n_2 - m_2 n_1.$$

The following routine does not include the integration portion of the program. Section B below contains the Mod Gurk integration used in the F100 simulation.

2. Direction Cosine Program

The operation codes given are DDP24 mnemonics.

OP.	Address	DAP	SCS	SDS	Comment
LDX	CSD	5	11	2	store a value in x-register
LDB	Q1	10	9	2	
MPY	N3	31	45	5	
STA	S1	10	9	3	$q_1 \times n_3$
LDB	RB3	10	9	2	
MPY	M3	31	45	5	
SUB	S1	10	12	2	$rb_3 \times m_3 - q_1 \times n_3$
ALS	1	6	12	2	
STA	L3DOT	10	9	3	
STA (X)	DCL3D	10	9	3	$\dot{l}_3$

OP.	Address	DAP	SCS	SDS	Comment
LDB	RB3	10	9	2	
MPY	L3	31	45	5	
STA	S1	10	9	3	$rb_3 \times l_3$
LDB	P	10	9	2	
MPY	N3	31	45	5	
SUB	S1	10	12	2	$p \times n_3 - rb_3 \times l_3$
ALS	1	6	12	2	
STA (X)	DCM3D	10	9	3	$\dot{m}_3$
LDB	P	10	9	2	
MPY	M3	31	45	5	
STA	S1	10	9	3	$p \times m_3$
LDB	Q1	10	9	2	
MPY	L3	31	45	5	
SUB	S1	10	12	2	$q_1 \times l_3 - p \times m_3$
ALS	1	6	12	2	
STA (X)	DCN3D	10	9	3	$\dot{n}_3$
LDB	Q1	10	9	2	
MPY	N2	31	45	5	
STA	S1	10	9	3	$q_1 \times n_2$
LDB	RB3	10	9	2	
MPY	M2	31	45	5	
SUB	S1	10	12	2	$rb_3 \times m_2 - q_1 \times n_2$
ALS	1	6	12	2	
STA (X)	DCL2D	10	9	3	$\dot{l}_2$
LDB	RB3	10	9	2	
MPY	L2	31	45	5	
STA	S1	10	9	3	$rb_3 \times l_2$
LDB	P	10	9	2	
MPY	N2	31	45	5	
SUB	S1	10	12	2	$p \times n_2 - rb_3 \times l_2$
ALS	1	6	12	2	
STA (X)	DCM2D	10	9	3	$\dot{m}_2$
LDB	P	10	9	2	
MPY	M2	31	45	5	
STA	S1	10	9	3	$p \times m_2$
LDB	Q1	10	9	2	
MPY	L2	31	45	5	
SUB	S1	10	12	2	$q_1 \times l_2 - m_2 \times p$
ALS	1	6	12	2	
STA (X)	DCN2D	10	9	3	$\dot{n}_2$
LDX	DCL2	5	11	2	$\dot{l}_2$ location to index
JST	MG	10	9	1	jump to Mod Gurk
ALS	2	7	13	3	
STA	L2	10	9	3	store new l_2
LDX	DCL3	5	11	2	$\dot{l}_3$ location to index
JST	MG	10	9	1	jump to Mod Gurk
ALS	2	7	13	3	
STA	L3	10	9	3	store new l_3
LDX	DCM2	5	11	2	$\dot{m}_2$ location to index
JST	MG	10	9	1	jump to Mod Gurk
ALS	2	7	13	3	
STA	M2	10	9	3	store new m_2
LDX	DCM3	5	11	2	$\dot{m}_3$ location to index
JST	MG	10	9	1	jump to Mod Gurk
ALS	2	7	13	3	
STA	M3	10	9	3	store new m_3
LDX	DCN2	5	11	2	$\dot{n}_2$ location to index
JST	MG	10	9	1	jump to Mod Gurk
ALS	2	7	13	3	
STA	N2	10	9	3	store new n_2
LDX	DCN3	5	11	2	$\dot{n}_3$ location to index
JST	MG	10	9	1	jump to Mod Gurk
ALS	2	7	13	3	
STA	N3	10	9	3	store new n_3

OP.	Address	DAP	SCS	SDS	Comment
LDX	CSD	5	11	2	restore index value
LDB	L3	10	9	2	e = normalize factor
MPY	L3	31	45	5	
STA	S1	10	9	3	l_3^2
LDB	M3	10	9	2	
MPY	M3	31	45	5	
STA	S2	10	9	3	m_3^2
LDB	N3	10	9	2	
MPY	N3	31	45	5	
STA	S3	10	9	3	n_3^2
LDA	"3"	10	9	2	
SUB	S1	10	12	2	
SUB	S2	10	12	2	
SUB	S3	10	12	2	
STA	DCEP3	10	9	3	$e = 3 - l_3^2 - m_3^2 - n_3^2$
TAB		5	7	-	normalize l_3
MPY	L3	31	45	5	
STA	L3	10	9	3	$l_3 = e \times l_3$
ALS	2	7	13	3	
STA (X)	DCL3	10	9	3	
LDB	DCEP3	10	9	2	normalize n_3
MPY	N3	31	45	5	
STA	N3	10	9	3	$n_3 = e \times n_3$
ALS	2	7	13	3	
STA (X)	DCN3	10	9	3	
LDB	DCEP3	10	9	2	normalize m_3
MPY	M3	31	45	5	
STA	M3	10	9	3	$m_3 = e \times m_3$
ALS	2	7	13	3	
STA (X)	DCM3	10	9	3	
CRA		5	7	1	Find largest of
ADM	M3	10	14	2	$\lvert l_3 \rvert$, $\lvert m_3 \rvert$, and $\lvert n_3 \rvert$.
SBM	L3	10	14	2	
JPL	M > L	6	9	3	
SBM	M3	10	14	2	
ADM	N3	10	14	2	
JPL	NBIG	6	9	3	
LDB	M2	10	9	2	case where $\lvert l_3 \rvert$ is largest.
MPY	M3	31	45	5	
STA	S1	10	9	3	
LDB	N2	10	9	2	
MPY	N3	31	45	5	
ADD	S1	10	8	2	$\frac{m_2 \times m_3 + n_2 \times n_3}{l_3}$
DIV	L3	33	68	10	
TAB		5	7	-	
ORA	"-000"	16	10	1	change sign
STA	L2	10	9	3	l_2 = orthogonalized l_2
LDB	L2	10	9	2	$e_2 = 3 - l_2^2 - m_2^2 - n_2^2$
MPY	L2	31	45	5	
STA	S1	10	9	3	
LDB	M2	10	9	2	
MPY	M2	31	45	5	
STA	S2	10	9	3	
LDB	N2	10	9	2	
MPY	N2	31	45	5	
STA	S3	10	9	3	
LDA	"3"	10	9	2	
SUB	S1	10	12	2	
SUB	S2	10	12	2	
SUB	S3	10	12	2	
STA	DCEP2	10	9	3	store e_2

OP.	Address	DAP	SCS	SDS	Comment
TAB		5	7	-	normalize l_2, m_2, and n_2
MPY	L2	31	45	5	
STA	L2	10	9	3	$l_2 = e_2 \times l_2$
ALS	2	7	13	3	
STA (X)	DCL2	10	9	3	
LDB	DCEP2	10	9	2	
MPY	N2	31	45	5	
STA	N2	10	9	3	$n_2 = e_2 \times n_2$
ALS	2	7	13	3	
STA (X)	DCN2	10	9	3	
LDB	DCEP2	10	9	2	
MPY	M2	31	45	5	
STA	M2	10	9	3	$m_2 = e_2 \times m_2$
ALS	2	7	13	3	
STA (X)	DCM2	10	9	3	
LDB	M3	10	9	2	
MPY	N2	31	45	5	
STA	S1	10	9	3	
LDB	M2	10	9	2	
MPY	N3	31	45	5	
SUB	S1	10	12	2	
SHL	1	6	13	2	
STA	L1	10	9	3	$l_1 = m_2n_3 - m_3n_2$
LDB	N3	10	9	2	
MPY	L2	31	45	5	
STA	S1	10	9	3	
LDB	L3	10	9	2	
MPY	N2	31	45	5	
SUB	S1	10	12	2	
SHL	1	6	13	2	
STA	M1	10	9	3	$m_1 = n_2l_3 - n_3l_2$
LDB	L3	10	9	2	
MPY	M2	31	45	5	
STA	S1	10	9	3	
LDB	M3	10	9	2	
MPY	L2	31	45	5	
SUB	S1	10	12	2	
SHL	1	6	13	2	
STA	N1	10	9	3	$n_1 = l_2m_3 - l_3m_2$
JMP	EXIT	5	7	1	
		2296	2915	500 cycles = 874 microseconds	

$$\frac{\text{PB440 - SCS}}{\text{DDP24 - DAP}} = \frac{2915}{2296} = 1.27$$

$$\frac{\text{SDS 9300}}{\text{DDP24 - DAP}} = \frac{874}{2296} = .38$$

B. MOD GURK INTEGRATION ON DDP24 AND PB440 COMPUTERS

1. Comparison of DAP and SCS

The following program shows the computation of the 0_{33} Mod Gurk integration formula,

$$x_n = ax_{n-1} + bx_{n-2} + cx_{n-3} + d\dot{x}_{n-1} + e\dot{x}_{n-2} + f\dot{x}_{n-3},$$

as it is given for the F100 simulation on UDOFT. The program is written in DAP, but times are also given for the execution of analagous instructions in the Systems Command Set. These latter times include 5 microseconds for control sequence execution, common to every SCS instruction. Times are also given for the SDS 9300 in terms of machine cycles. The total SDS time is computed according to the formula in Appendix V. (Note LDB would be changed to LDA for the SCS 9300 program.)

Location	Index	OP	Address	Comment	DAP	SCS	SDS
MG	X	LDB	MGA		10	9	2
	X	MPY	XN-1	ax_{n-1}	31	45	5
		STA	S1		10	9	3
		LDB	MGB		10	9	2
	X	MPY	XN-2		31	45	5
		ADD	S1	$+bx_{n-2}$	10	8	2
		STA	S1		10	9	3
		LDB	MGC		10	9	2
	X	MPY	XN-3		31	45	5
		ADD	S1	$+cx_{n-3}$	10	8	2
		STA	S1		10	9	3
		LDB	MGD		10	9	2
	X	MPY	XDOTN-1		31	45	5
		ADD	S1	$+d\dot{x}_{n-1}$	10	8	2
		STA	S1		10	9	3
		LDB	MGE		10	9	2
	X	MPY	XDOTN-2		31	45	5
		ADD	S1	$+e\dot{x}_{n-2}$	10	8	2
		STA	S1		10	9	3
		LDB	MGF		10	9	2
	X	MPY	XDOTN-3		31	45	5
		ADD	S1	$+f\dot{x}_{n-3}$	10	8	2
		JMP	PRB		5	7	1
PRB	X	STA	XN-1	new x_{n-1}	10	9	3
		JMP	EXIT		5	7	1
					366μS	432μS	72 cycles 125μS

2. Mod Gurk Integration for Systems Command Set with MAD Instruction Added

The following program shows the result of adding a special instruction, "Multiply-Add", to the list of microutines provided with the Systems Command Set. The MAD instruction multiplies the word at the effective address by the word in the B-register, and adds the product to the contents of the A-register. The MAD-microutine is shown below:

Time	OP	R1	R2	Comment
2	LDM	D	D	(M)→D
1	CPL	A	C	Save A→C
1	CPS	B	A	$(B)_S \rightarrow A_S$
1	ADS	D	A	form sign of result.
1	LRC	"23"		
2	TNZ	B	S	Test B-
1-0	CCL	B	B	Form B
1-0	CIL	B	B	
1	CPM	Q	A	Clear A
2	TNZ	D	S	Test D-
1-0	CCL	D	D	Form D
1-0	CIL	D	D	
23	MPS	D	A	Multiply
2	TNZ	A	S	Test result -
1-0	CCL	A	A	Form 2's comp.
1-0	CIL	A	A	
1	CLP	CSQ		Return to control.
1	ADL	C	A	Add (C).
38-44				
+5	control sequence			
43-49	microseconds			
9	micropair required for MAD			

The Mod Gurk integration on the PB440 employing the MAD-augmented Systems Command Set is:

	OP	Address	Comment	Time
Enter	LDB	MGA		9
	MPY	XN-1	ax_{n-1}	45
	LDB	MGB		9
	MAD	XN-2	$+bx_{n-2}$	46
	LDB	MGC		9
	MAD	XN-3	$+cx_{n-3}$	46
	LDB	MGD		9
	MAD	XDOTN-1	$+d\dot{x}_{n-1}$	46
	LDB	MGE		9
	MAD	XDOTN-2	$+e\dot{x}_{n-2}$	46
	LDB	MGF		9
	MAD	XDOTN-3	$+f\dot{x}_{n-3}$	46
	JMP	PRB		7
PRB	STA	XN-1	new x_{n-1}	9
	JMP	EXIT		7
				352 microseconds

3. Microroutine for Mod Gurk Integration

The following program shows Mod Gurk integration programmed entirely on the micro-level. It is assumed that this routine would be called by a macro-instruction of the form

MGK A ,

where A is the address of x_{n-1}. x_{n-2}, x_{n-3}, $\dot{x}_{n-1}$, $\dot{x}_{n-2}$, and $\dot{x}_{n-3}$ are to be stored sequentially after x_{n-1}.

Note that the integration coefficients are stored within the microroutine itself, and are assumed to be in sign-magnitude form.

Time	OP	R1	R2	Comment
1	STW	1	L	Save Loc. of next macro
1	CPL	D	L	Address of x_{n-1} → L
1	LDI	P	B	a → B, skip next pair
1	LRC	"23"		x_{n-1} → D, (L) + 1 → L
	[	a	]	
2	LDI	L	D	
2	TNZ	D	S	Test for D-.
1-0	CCM	D	D	Find D if D-.
1-0	CIL	D	D	
1	XOR	A	A	Clear A.
23	MPS	D	A	ax_{n-1} → A.
1	LRC	"23"		
2	TNZ	D	S	Test sign of product
1-0	CCL	A	A	Complement if -.
1-0	CIL	A	A	
2	LDI	L	D	x_{n-2} → D
1	CPL	A	C	ax_{n-1} → C
2	LDI	P	B	b → B, skip next pair.
2	TNZ	D	S	Test D-.
	[	b	]	
1-0	CCM	D	D	Complement D if -.
1-0	CIL	D	D	
1	XOR	A	A	Clear A
23	MPS	D	A	b x_{n-2} → A
1	LRC	"23"		
2	TZO	D	S	Complement product

Time	OP	R1	R2	Comment
1-0	CCL	A	A	if D+, (since b is -)
1-0	CIL	A	A	
1	ADL	A	C	$+bx_{n-2} \rightarrow C$
1	XOR	A	A	Clear A
2	LDI	L	D	$X_{n-3} \rightarrow D$
2	TNZ	D	S	Complement D if -.
1-0	CCL	D	D	
1-0	CIL	D	D	
1	LDI	P	B	$C \rightarrow B$
23	MPS	D	A	$Cx_{n-3} \rightarrow A$
	(	c	)	
1	XOR	A	A	Clear A
2	TNZ	D	S	Complement product if D-.
1-0	CCL	A	A	
1-0	CIL	A	A	
1	ADL	A	C	$+Cx_{n-3} \rightarrow C$
1	LRC	"23"		
2	LDI	L	D	$\dot{x}_{n-1} \rightarrow D$, (L) + 1 $\rightarrow$ L
2	TNZ	D	S	Complement D if -.
1-0	CCL	D	D	
1-0	CIL	D	D	
1	LDI	P	B	d $\rightarrow$ B, skip next pair
23	MPS	B	A	$d\dot{x}_{n-1}$.
	(	d	)	
1	LRC	"23"		
2	TNZ	D	S	Complement product if D-.
1-0	CCL	A	A	
1-0	CIL	A	A	
2	LDI	L	D	$\dot{x}_{n-2} \rightarrow D$, (L) + 1 $\rightarrow$ L
1	ADL	A	C	$+d\dot{x}_{n-1} \rightarrow C$
1	LDI	P	B	e $\rightarrow$ B, skip next pair
2	TNZ	D	S	Complement D if -.
	(	e	)	
1-0	CCM	D	D	Complement D if -.
1-0	CIL	D	D	
1	XOR	A	A	Clear A
23	MPS	D	A	$e\dot{x}_{n-2} \rightarrow A$
1	LRC	"23"		
2	TNZ	D	S	Complement product if D-.
1-0	CCL	A	A	
1-0	CIL	A	A	
1	ADL	A	C	$+e\dot{x}_{n-2} \rightarrow C$
1	XOR	A	A	Clear A
2	LDI	L	D	$\dot{x}_{n-3} \rightarrow D$
2	TNZ	D	S	Complement D if -.
1-0	CCM	D	D	
1-0	CIL	D	D	
1	LDI	P	B	f $\rightarrow$ B, skip next pair
23	MPS	D	A	$f\dot{x}_{n-3} \rightarrow A$
	(	f	)	
2	LDW	1	L	Pick up loc. of next macro
2	TNZ	D	S	complement product if D -.

Time	OP	R1	R2	Comment
1-0	CCL	A	A	
1-0	CIL	A	A	
1	CLP	CSQ		Return to control
1	ADL	C	A	MGURK A.

204-228 microseconds
43 micro-pairs

In summary, the times to perform the Mod Gurk integration using the various programming languages and computers are:

Computer and Language	Time in microseconds
DDP24-DAP	366
PB440-SCS	432
PB440-SCS-MAD	352
PB440-Micro-coded	209-233
SDS 9300	125

C. LEVEL SELECT AND SLOPE COMPUTATION

1. DAP and SCS

The following routine locates tabular values X_i and $X_i + 1$ which bracket an argument X, and computes the quantity $(X - X_i)/(X_{i+1} - X_i)$. This is the level-select-and-slope routine suggested by Krasny (Reference 5) for function generation. The routine assumes that the previous interval of the argument is known, and that the argument now lies in the same interval or an adjoining interval. Times given for the Systems Command Set include 5 microseconds control time. The worst-case time assumes that all three intervals must be checked.

Location	OP	Address	DAP	SCS	Comment
	LDX	OLDX	5	11	Old level→X reg.
	LDA	X	10	9	X→A reg.
LOOP	SKG(X)	TABLE	10	15	Loop if $X > X_{i+1}$
	JMP	COMPUT	5	7	Compute if $X \leq X_{i+1}$
	JXI	LOOP	7	11	
COMPUT	STX	OLDX	10	12	Store new level.
	SUB(X)	TABLE	10	12	
	STA	S1	10	9	
	LDA(X)	TABLE	10	9	
	SUB(X)	TABLE-1	10	12	
	DIV	S1	33	68	Compute slope.
	STB	SLOPE	10	9	
worst case time			157	225 microseconds	

2. PB440 Microroutine

The following microroutine shows the level-select-and-slope operation coded as a microroutine. It is called by a macro of the form

```
1. LSS   X
2.       K
```

where X is the address of the argument and K is the address of the previous level.

Location	Time	OP	R1	R2	Comment
LSS	2	LDM	L	B	Old Level (K)→B
	3	LDM	D	D	X→D
LOOP	2	LDI	B	C	X_k→C, K + 1→B
	1	CPS	C	A	Test X and X_k for same sign.
	1	ADS	D	A	
	2	TZO	A	S	
	1-0	FTR	SAMSIN		same sign
	1-0	NOP			
	1	TNZ	D	S	Opposite signs, X is -.
	1-2	BTR	LOOP		
LEVEL	1	CDL	B	B	
	1	CDL	B	B	K - 1→B
	2	LDM	B	A	X_{k-1}→A
	1	STI	L	B	Store new k, set L
	1	CCL	C	C	
	1	CIL	C	C	$-X_k$→C
	1	ADL	A	C	$X_{k-1} - X_k$→C
	1	CCL	A	A	
	1	CIL	A	A	$-X_{k-1}$→A
	1	ADL	D	A	$X - X_{k-1}$→A
	1	XOR	B	B	Clear B
	1	CCS	C	B	
	1	ADS	A	B	Set sign of result
	1	LRC	"23"		
	1	CLP	CSQ		Return to control
	46	DVS	C	A	with slope in B
SAMSIN	1	CCL	C	A	
	1	ADL	D	A	$X - X_k$ - 1→A
	1	TNZ	A	S	
	1-2	BTR	LOOP		
	1-0	BTR	LEVEL		
	1-0	NOP			

worst case time, assuming three trips through LOOP and SAMSIN; 112 microseconds.

Summarizing the level-select-and-slope routine times:

Computer	Time in microseconds
DDP24-DAP	157
PB440-SCS	225
PB440-Micro-coded	112

D. INDEXING OPERATIONS WITH THE PB440

Unlike most computers the PB440 does not have hardware which performs the function of index registers. Instead, the indexing process must be programmed into the command set and its control sequence. Although theoretically this allows the user to construct the system any way he pleases, with, say, an arbitrary number of memory locations serving as index registers, nevertheless it will be seen that the most efficient indexing procedure will be one which takes advantage of certain special commands on the PB440. The most important of these is the LDS command, which is described in detail in Appendix I.

The uses of LDS (and STS) which particularly concern indexing are those in which R1 is 0. In this case one of the 8 memory locations 64-71 is referenced, depending on the configuration of the three bits, 6, 7, and 8, in the D register. Thus, if these 8 memory locations are set aside to function as index registers, and if bits 6-8 of the macro-instructions are set aside to specify indexing then LDS, with R1 = 0, may be used to give quick access to the appropriate register. This process is illustrated in the following control sequence.

	OP	R1	R2	Time
1.	LDI	L	D	2 + 3ST
2.	LDS	O	C	3 (6: see note 1)
3.	LDS	7	P	1
4.	ADF	C	D	1

Line 1. At the start of this operation the L register is assumed to contain the address of the next macro-instruction. Hence the first operation performed in line 1 is that of loading this instruction into the D register. Simultaneously the L register is incremented by one; thus it will contain the location of the macro-instruction next in sequence to the one just called.

Line 2. The contents of one of the memory locations 64-71 are loaded into the C register. Which one depends on bits 6-8 of the D register, which are now bits 6-8 of the macro-instruction. Hence for example, if the macro-instruction had a 5 contained in positions 6-8, the contents of "index register" 5 would now be in the C register.

Note 1:
Note that this memory access requires 3 microseconds instead of 2, because it is in the right half of the micro-pair. (If the same memory bank is used for both index registers and the macro-instruction, 6 microseconds will be required for this access of the index.)

Line 3. This step causes the instruction to be "interpreted", i.e., it loads the P register with the contents of a memory cell whose location depends on bits 0-5 of the D register, which contain the macro operation code. Thus the next micro-pair to be executed will be the first micro-pair of the appropriate micro-routine.

Line 4. This step performs the actual indexing. The address portion of the index register whose contents are now in the C register, is added to the address portion of the macro-instruction, which is in the D register. Thus when the microutine begins, it will find in the D register not the original macro-instruction with its address as it was in memory, but the macro-instruction with its address incremented by the contents of the specified index register.

The reader will note that in the above sequence every macro-instruction will be indexed; instructions with all index bits zero have the address portion of memory location 64, index 0, added to their addresses. Thus with the above control sequence one would keep all zeroes in memory location 64.

Other indexing control sequences and methods are shown below. These control sequences illustrate methods for achieving a trade off between generality and operating speed.

1. EIR: Enable Index Registers

Location	Time	OP	R1	R2	Comment
EIR	1	LDI	P	C	Next instruction→C, (P) + 1→P
	1	NOP			No operation
		LDI	L	P	Part of new control sequence,
		LDS	O	C	not executed here.
	1	CLP	CSQ		Address of control→P.
	5	STM	P	C	New control sequence→address of control. (C)→CSQ

The above microutine, Enable Index Registers, could be used to change the control sequence from the non-indexing type to an indexing type. Thus parts of the program which did not require indexing could be done with the 4 microsecond control sequence, and the EIR instruction could be used before a section where indexing was needed.

2. DIR: Disable Index Registers

Location	Time	OP	R1	R2	Comment
DIR	1	LDI	P	C	Next instruction→C, (P) + 1→P
	1	NOP			No operation
		LDI	L	D	New control sequence, not
		LDS	7	P	executed here.
	1	CLP	CSQ		Address of control→P.
	5	STM	P	C	New control sequence address of control. (C)→CSQ

This microroutine, Disable Index Register, could be used to change from the 7-microsecond control sequence back to the 4-microsecond one.

3. INI: Index Next Instruction

Location	Time	OP	R1	R2	Comment
INI	2	LDI	L	D	Pick up next instruction, (L) + 1→L
	3	LDS	O	C	Pick up index register. Index→C
	1	LDS	7	P	Branch to microroutine.
	1	ADF	C	D	Index address.

This microroutine, Index Next Instruction, could be used when an isolated instruction is to be indexed. In this case it would be quicker (4 microseconds versus 26 microseconds) than "enabling" and the "disabling" the index registers. The microroutine for this instruction consists of a single application, to the next macro-instruction, of the 7-microsecond control sequence.

4. EMR: Execute Microroutine

Location	Time	OP	R1	R2	Comment
EMR	1	CPL	D	P	Address of microroutine→P.
	1	NOP			No operation

This microroutine, Execute Microroutine, could be used to overcome the limitation imposed by having only 64 locations in the jump-table. By means of EMR it would be possible to enter any number of microroutines from the macro-level by putting the address of the microroutine in the address portion of an EMR instruction.

APPENDIX V

COMPUTER SYSTEM FUNCTIONAL DESCRIPTIONS

A. PB440

The PB440 Computer is a core-memory computer whose basic operations can be performed at the clock rate of the computer. The programmer has complete control over the specific operation performed during nearly every clock pulse; in this sense the computer is said to be micro-programmed. Sequences of basic operations can be stored in memory and executed much like a conventional subroutine, so that operations normally considered to be "commands" on a conventional computer can be described in terms of their elementary operations, and can be changed at will. In this sense the computer is said to be of the stored-logic type.

Two separate types of core memory are provided, although the memory is addressed as a single homogeneous block. Main memory, comprising the bulk of the storage capacity of the machine, consists of magnetic cores from which one word of information can be obtained in 2 microseconds. The memory cycle time is 5 microseconds. Fast memory, designed to hold the sequences of elementary steps which define an "instruction", consists of a set of BIAX cores, from which a word can be obtained in less than 1 microsecond. Since these cores are read non-destructively, the term "cycle time" does not apply to fast memory. The word length of both memories is 24 binary digits (bits). (Internal parity checking is provided automatically, and utilizes an additional bit in each word for this purpose. This bit is not available for programming purposes.)

The recommended PB440 system is to have 8,192 words of main memory and 512 words (two modules) of fast memory. The main memory can be expanded to a maximum of 28,672 words in 4096 word modules, and fast memory can be expanded to 4096 words in 256 word modules. The second fast memory module to be supplied permits the 64 word jump table used in macro-instruction interpretation to be stored in fast memory with 448 words left for microutine storage.

The fast memory containing the microutines is to be considered part of the control portion of the PB440 rather than as part of the memory. This microutine control may be changed at will by the programmer to provide a change in the instruction repertoire. In a conventional computer, a modification in the hardware would be required to make a corresponding alteration in the repertoire. The advantages of the microutine concept are:

1. Permits selection from a wide variety of instructions, and
2. Permits versatile partial word operations to be performed at a corresponding savings in time.

The disadvantage is that for a given bandwidth capability of computer circuits, the micro-coded computer is slower than the conventional computer.

The standard input/output devices to be supplied with the PB440 in this application are:

1. Paper tape reader, bidirectional, 500 character per second, with spooler, Digitronics;
2. Paper tape punch, 110 character per second, Teletype BRPE; and
3. Input/Output typewriter, IBM Selectric.

Paper Tape Reader

The photoreader may be considered the primary input device for the PB440. It is capable of operating at a rate of 500 characters per second. Input can be requested in one of two modes: 8-bit characters, for binary tape input or for internal parity checking, or 7-bit input with parity checking by the controller automatically. The photoreader operation proceeds as follows:

When the photoreader is initially "connected" to the computer its internal buffer is cleared and the tape begins to move. It can move either forward or backward, depending on the command used to "connect" it. It ignores blank, or "tape feed" characters, and "code delete" characters, in 7-bit mode, and ignores tape without sprocket holes in 8-bit mode. When an acceptable character is read, that character is loaded into the character buffer, and a signal is sent to the computer along the interrupt line.

The photoreader does not stop its tape motion at this time, however. The tape continues to move until the next acceptable character is available under the photocells. If, by this time,

the first character has not been removed from the buffer, the tape motion is stopped. If the computer has performed a "data transfer in" operation which referenced the photoreader buffer (this operation clears the buffer to zero) the new character is inserted into the buffer, a new interrupt signal is sent, and the tape continues to move in the chosen direction. The interrupt signal is actually a DC voltage change, so it is available any time the computer program examines it. The two micro-steps, TCT and TCF, may be used to examine any one of the 4 lines to determine whether an interrupt signal is present on any one of them.

Paper Tape Punch

The paper tape punch may be considered the primary output device for the PB440. It is capable of operating at a maximum rate of 110 characters per second, and can punch in either of the two modes available for input on the photoreader: 7-bit mode, with parity automatically supplied by the punch controller, or 8-bit mode. Its operating sequence is very similar to that of the photoreader. When it is first "connected" to the computer it examines its character buffer. If this buffer is not in use (a character is not waiting to be punched) it signals the computer by means of the interrupt line to which it has been assigned. When the computer provides it a character to punch, the interrupt signal is turned off, and the tape is set in motion. The punch action involves activating a set of 8 "interposers", which are mechanical links that determine which holes are to be punched and which are to be left unpunched. As soon as the interposer action has been properly initiated, the punch signals the computer, via the interrupt line, that it is ready to accept another character for punching. This action also clears the character buffer to zero. If the computer provides another character within the proper time interval the tape will continue to move, and the new character will be punched. If the computer fails to provide a new character in time, the tape motion will be stopped and the punch will wait to be activated again.

Typewriter

The typewriter can be considered both an input and an output device, and, for purposes of discussion, these two distinct operations will be considered separately.

The typewriter, as in input device, is "connected" to the computer under control of the computer program. When the operator depresses a key, the corresponding code is entered into the typewriter's character buffer, and an interrupt signal is sent to the computer. If the computer does not respond, i.e., does not remove the character from the buffer, the keyboard is locked to forestall further typing action. When the computer empties the typewriter character buffer, the keyboard is unlocked to indicate that typing can proceed.

When the typewriter is "connected" to the computer for output its character buffer is examined, to see if it is still in the process of typing a character. If it is not, it responds, via the proper interrupt line, to inform the computer that it is capable of accepting a character for printing. While it is in the process of printing the character, it will be in the "busy" state, and will so inform the computer if queried. As soon as the proper key has been chosen, so the typewriter is committed to print that character, the typewriter character buffer is cleared and an interrupt signal is sent to the computer to indicate that it can receive another character for printing.

When any one of the devices has been "disconnected" from the computer, the interrupt line has also been disconnected so that unwanted interrupt signals will not occur. Should they be "re-connected" while still busy, they will inform the computer of this fact by means of a "status-response", which is provided automatically when the "connect" operation is initiated.

It can be seen that the meaning of a "busy" response from the typewriter might mean different things under different conditions. If, for example, the typewriter is being used as an output device, as soon as the chosen key begins to move, the typewriter is no longer "output busy"; that is, it can accept another output character for printing. If the typewriter is "re-connected" for input while a key is still in action, the code corresponding to this key might enter the character buffer. This action is prevented by special logic in the controller and cannot occur. Under these conditions the unit will indicate its "busy" state by a "busy" status response and a "false" interrupt line. The status response returned by the typewriter will indicate its state in enough detail so that the program can determine how it may be used at any time.

Interrupt Feature

The PB440 interrupt feature is associated with the input/output control and provides for four channels of interrupt. The description of this interrupt feature is contained in the Input-Output System discussion in Appendix I.

Program Input/Output

The program input medium will be via 8 level paper tape, read into the computer at 500 characters per second. Program listing will be via the output typewriter. Binary object programs will be punched on paper tape for later reloading via the paper tape reader.

The format of program input is yet to be published by the Packard-Bell software people, but can be expected to conform to the conventional tabular assembly coding sheet layouts with fields for statement number, tag, mnemonic op code, address, and comment.

Maintenance Considerations

Maintenance features of the PB440 include:

1. Comprehensive register status displays,
2. DC power supply voltage marginal test,
3. Single clock step and single micro-step operating rates,
4. Memory parity bit and error detector, and
5. Machine check and diagnostic routines.

The content of registers is displayed in either of two forms; as binary indicators on the circuit cards which may be observed from a seated position at the console when the top deck of the computer is raised (a possible permanent arrangement at the discretion of the user), or as an octal equivalent in a single 8 digit console register display, selection of which is by a register select switch. The power supply voltage margin adjustment facility employed in conjunction with machine check and diagnostic routines permits detection of incipient failures and isolation of intermittant high speed failures. The failures which occur only at high speed are usually the most difficult to isolate. A comprehensive machine diagnostic program is valuable in this instance, using the computer to detect its own failures.

A feature of the PB440 configuration recommended, which aids in detecting memory failures, is the dual main memory and fast memory modules. This allows the diagnostic routine to be executed out of one set of memories, say main module 1 and fast module 1, to check the other set. Reversal of the process then permits the checkout of main module 1 and fast module 1 with the routine being executed from main module 2 and fast module 2.

The register display used in conjunction with the Single Clock Step manual push button, permits stepping through instruction sequences clock pulse by clock pulse and visual monitoring of the register contents for improper operation. This technique is employed to locate catastrophic or complete component failures. An aid in this area is the address halt facility which permits running at normal computer speed until a manually selected memory address is accessed, at which time the computer halts. This feature is useful in proceeding through a lengthy routine quickly to reach the section of interest which may then be examined a clock pulse at a time.

The memory parity bit feature is quite valuable in quickly pinpointing a memory location or set of locations which may be intermittantly failing. Without the memory parity bit, isolation of memory bit dropouts or pickups is a very tedious and time-consuming operation since the symptoms often appear in running programs many instruction execution times after the failure occurred and may not be indicative of a memory failure.

Operator's Console

The operator's console layout is shown in figure V-1. Most of the controls are labeled in a self-explanatory manner, such as: Power ON-OFF, Paper Tape Reader Rewind, Run, and Load, etc. Those controls that are self-explanatory will not be further elaborated upon.

Program loading is performed using a wired in bootstrap routine which reads paper tape into the computer. The Bootstrap Fill switch is provided for this function and causes the bootstrap routine to be inserted into the first few locations in main memory. This operation passes current through a special winding which sets and clears the proper bit cores, destroying the previous contents of those few locations.

The Register Select switch permits manual selection of a register. When the round unlabeled knob to the lower right of the Register Select switch is rotated to the position enabling the Register Select push button and when that push button is depressed, the content of the specified register is brought to the 8 octal digit display register in the upper right portion of the

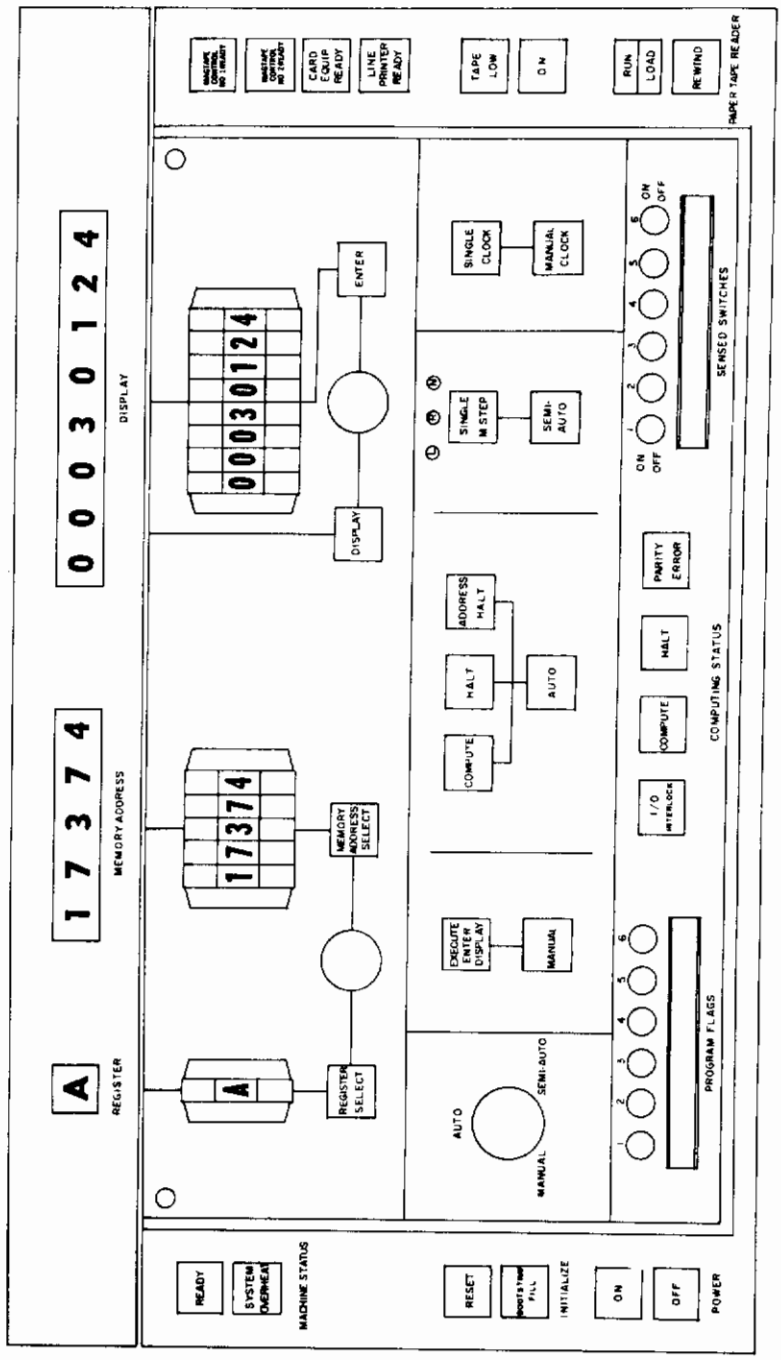


Figure V-1 PB440 Control Console

console. In similar manner a memory location may be selected in the Memory Address octal switches, and if the Memory Address Select button is enabled and depressed, the content of the selected location is displayed.

To the right of the Memory Address Select control is a group of 8 octal digit switches which are employed to manually insert an operand into the location selected either by the Register Select or Memory Address Select. If the Enter button is enabled, the data is deposited into the desired location and simultaneously the entered data is displayed in octal form. When the Enter button is disabled, the octal display presents data as selected by either the Register Select or Memory Address Select controls.

Immediately below the Display-Enter controls, are the computer rate selection controls. The L and R indicators permit the operator, while in the single micro-step mode, to determine which of the two micro-steps in the pair are being executed, the left micro or the right micro. The N indicator will show when the next micropair is being accessed. Normally R and N are on simultaneously since the execution of the right micro is simultaneous with the fetch of the next micropair.

The Address Halt button permits the operator to insert an address in the Memory Address Select switches and have the computer operate at compute speed until the address is reached in the program, at which time the computer halts. Program debugging at the console is facilitated with the Address Halt feature.

B. SDS 9300

The SDS 9300 is in prototype assembly at present. In view of its nebulousness, the descriptive information contained herein is subject to change. Gaps in the information are primarily due to the manufacturer's reluctance to release information about the SDS 9300 system at this early date.

The SDS 9300 is a general-purpose digital computer intended for scientific and systems computation application. The word length is 24 bits plus a parity bit in memory and I/O transfers. The number system is binary and arithmetic is performed using two's complement code. Floating point operands are double precision, requiring two 24 bit words for the sign, 8 bit exponent, and 38 bit mantissa. Built-in floating point instructions are an extra cost option not included in the recommended SDS 9300 system.

The instruction repertoire has many double-precision instructions as well as a half-precision twin multiply which would find use in simulation computation. The double precision operations are so fast on the SDS 9300 (Add Double Precision taking only 3.50 microseconds), that the computer compares favorably with a 48 bit word computer in simulation computation, until memory limitations crop up due to two-word operand storage problems.

Supplied as part of the basic SDS 9300 are:

1. 3 Index Registers
2. 1 Automatic I/O Channel
3. Paper Tape Reader, 300cps
4. Tape Spooler
5. Paper Tape Punch, 60cps
6. Input/Output Typewriter

Address modification capability includes Index Registers which can be incremented or decremented by any integer from 0 to 225, indirect addressing, and combined indexing and indirect addressing.

Memory

The recommended SDS 9300 system has 8,192 words of memory in two modules of 4,096 words each. The memory may be expandable to 32,768 words in modules of 4,096 words. Memory cycle time is 1.75 microseconds with an access time of 0.7 microseconds. Each 4096 word module has independent read/write electronics, allowing memory cycle overlapping between modules.

In the recommended system employing two memory modules, memory accesses alternating between modules will permit overlapping so that accessing may be carried on at an average rate of one every 0.875 microseconds. This minimum average access period can be approached by storing instructions in one module and operands in another. The minimum of 0.875 microseconds cannot be realized in practical programs however, since more instructions are fetched, as a rule, than operands. This is due to normal use of instructions which do not fetch memory operands such as skips, branches, shifts, register manipulations, etc.

The memory is non-volatile with power failure.

Automatic I/O Channel

The Automatic I/O channel permits I/O-memory transfers independent of the central computer. Transfer rates are memory-cycle limited to a 550,000 words per second maximum. The channel is bi-directional and contains two 24 bit word buffers with character assembly, disassembly, and parity manipulation features. A Count Control register of 15 bits permits 32,768 words to be processed in a block. An Address register of 15 bits holds the address of the first word of the block. This address may be both indexed and indirect. The Automatic I/O channel may conduct search operations without requiring memory access.

I/O Channel Interrupt

Upon completion of an I/O operation, or at an end of file indication, or upon detection of a parity error, an interrupt may be issued from the Automatic I/O channel to the main program. The programmer has the option of enabling or disabling this interrupt.

Priority Interrupt

When an interrupt occurs, the program, at the completion of the ongoing instruction, traps to the location assigned to that channel. All channels have an assigned priority such that they cannot be interrupted by a channel with lower priority but only by one of higher priority. The subroutine servicing instructions permit the program to service the interrupts in the correct order and to return to the main program after all interrupts have been serviced.

In certain cases, it is appropriate to inactivate one or a number of interrupt channels and so effectively change the priority configuration. An Arm-Disarm feature is provided for this function. The two highest priority channels are reserved for the Power Fail-Safe System. When power drops below a threshold, the power failure interrupt occurs and stores all volatile register information in memory and halts the computer. When power rises above the threshold, the power resume interrupt restores the contents of the volatile registers and starts the computer. These two interrupts cannot be disarmed.

Operator's Console

The SDS 9300 is in an early equipment development stage and mechanical description is sparse. The console features are not readily available, however the claim is made that there is display of all programmable registers with extensive manual controls, six sense switches, two manual interrupt switches, and a Selective Halt control.

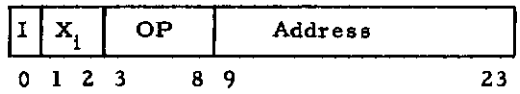
Software

A Symbolic Assembler, a Monitor System, and a FORTRAN compiler will be delivered with the SDS 9300 computer.

Maintenance Provisions

Maintenance provisions are not known, but it is anticipated that the Manual Clock Step, Margin Test, and register display features of the 910 and 920 computers will form the basis for SDS 9300 maintenance. The memory parity bit error detector in the SDS 9300 merits the same value as in the PB440 discussion in the previous section.

SDS 9300 Instruction Format



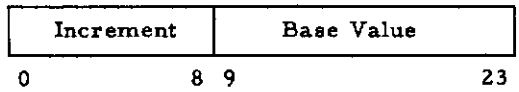
Bit 0 - Indirect address bit. A 'one' causes the computer to interpret bits 9-23 of the instruction (possibly modified by indexing) as the memory location where the effective address of the instruction may be found. A 'zero' causes bits 9-23 (possibly modified by indexing) to be interpreted as the effective address of the instruction.

Bits 1, 2 - Index Register bits. A non-zero entry in these positions causes the contents of bits 9-23 of the appropriate Index Register to be added to the address portion of the instruction prior to execution. Three Index Registers are provided.

Bits 3-8 - Instruction Code.

Bits 9-23 - Address.

Index Format



The increment will decrement if the sign is minus.

Instruction Repertoire

The preliminary SDS 9300 instruction repertoire listed below is subject to change as the logic design is firmed. The tentative listing below provides the instruction name, mnemonic code, function, and timing in machine cycles.

The execution times for the SDS 9300 instructions may be derived from the number of machine cycles required simply by multiplying the cycle time of 1.75 microseconds by the number of cycles. Thus the execution time T_e is given by $T_e = 1.75\ N$, where N is the number of machine cycles in the instruction. The resulting time applies to a computer with one 4K memory bank. When two or more 4K memory banks are employed, one may gain execution speed by storing instructions in one bank and data operands in another bank. With this storage assignment feature used optimally, usually one machine cycle may be saved for each instruction executed since the instruction fetch memory access may overlap the operand fetch. Thus an ADD instruction might require only one effective machine cycle rather than two if the instruction was fetched from one bank and the operand from the other.

Since it is not always possible or economical to program in the optimum fashion, it is reasonable to expect that the achievable operating rate would lie at some point between the optimum rate and the non-interlaced rate. A reasonable assumption to make, then, is that the time for an instruction execution with interlaced memories, T_i, would be the average of the minimum and maximum times, T_{min} and T_e. Since only one machine cycle is saved for single operand instructions,

$$T_{min} = T_e - 1.75$$

$$T_{min} = 1.75\ N - 1.75$$

$$T_{min} = 1.75\ (N - 1)$$

Solving for T_i:

$$T_i = \frac{T_{min} + T_e}{2}$$

$$T_i = \frac{1.75\ (N - 1) + 1.75\ N}{2} = 0.875\ (2N - 1)$$

It is expected that this expression for T_i will not hold for every instruction but should permit a quick, reasonably close approximation for groups of instructions used in programs.

SDS 9300 INSTRUCTION LIST

Designation	Name	Function	Timing in Machine Cycles
LOAD-STORE			
LDA	LOAD A	(M)→A	2
STA	STORE A	(A)→M	3
LDB	LOAD B	(M)→B	2
STB	STORE B	(B)→M	3
LDX	LOAD INDEX	(M)→X_i	2
STX	STORE INDEX	(X_i)→M	3
LDP	LOAD DOUBLE PRECISION	(M, M+1)→B, A	3
STD	STORE DOUBLE PRECISION	(B, A)→M, M+1	4
XMA	EXCHANGE M AND A	(M)↔(A)	3
XMB	EXCHANGE M AND B	(M)↔(B)	3
LDS	LOAD SELECTIVE	(M)(B)+(A)($\overline{B}$)→A	2
STS	STORE SELECTIVE	(A)(B)+(M)($\overline{B}$)→M	3
XMX	EXCHANGE MEMORY AND INDEX	(M)→(X_i)	3
ARITHMETIC GROUP			
ADD	ADD M TO A	(A)+(M)→A	2
DPA	DOUBLE PRECISION ADD	(B, A)+(M, M+1)→B, A	3
SUB	SUBTRACT	(A)-(M)→A	2
DPS	DOUBLE PRECISION SUBTRACT	(B, A)-(M, M+1)→B, A	3
MPO	MEMORY PLUS ONE	(M)+1→M	3
MPT	MEMORY PLUS TWO	(M)+2→M	3
MUL	MULTIPLY	(A)x(M)→A, B	5
DIV	DIVIDE	(A, B)÷(M)→A, Rem→B	10
ADM	ADD A TO M	(A)+(M)→M	3
TMU	TWIN MULTIPLY	(A_{0-11})x(M_{0-11})→A; (A_{12-23})x(M_{12-23})→B	5
FLOATING POINT GROUP (OPTIONAL FEATURE ON SDS 9300)			
FLA	FLOATING ADD	(A, B)+(M, M+1)→A, B Floating	7-11
FLS	FLOATING SUBTRACT	(A, B)-(M, M+1)→A, B Floating	7-11
FLM	FLOATING MULTIPLY	(A, B)x(M, M+1)→A, B Floating	8
FLD	FLOATING DIVIDE	(A, B)÷(M, M+1)→A, B Floating	17
UFA	UNNORMALIZED FLOATING ADD	(A, B)+(M, M+1)→A, B Unnormalized Floating	7-11
FLN	FLOATING NEGATE	-(A, B)→AB Floating	1

Designation	Name	Function	Timing in Machine Cycles
LOGICAL GROUP			
ETR	EXTRACT	(A)(M)→A (Logical And)	2
MRG	MERGE	(A)⊕(M)→A (Logical Or)	2
EOR	EXCLUSIVE OR	$(A)(\overline{M})\oplus(\overline{A})(M)\rightarrow A$	2
BRANCH GROUP			
BRU	BRANCH UNCONDITIONALLY	M→P	2
BRX	INCREASE INDEX AND BRANCH	$i \neq 0$; $(X_i)_{0-8}+(X_i)_{9-23} \rightarrow X_{i\ 9-23}$ if $(X_i)_9 = 1$, M→P i = 0; M→P, clear interrupt	2-3
BRM	MARK PLACE AND BRANCH	$(P)\rightarrow M_{10-23}$; $1\rightarrow M_9$; $(FR^*)\rightarrow M_{3-8}$; $(OV)\rightarrow M_2$, M+1→P	3
BRR	RETURN BRANCH	$(M)_{10-23}+1\rightarrow P$; $(M)_{3-8}+(FR)\rightarrow SR$: $(M)_2+(OV)\rightarrow OV$	2
TESTS/SKIP (If no skip, the shorter execution time prevails)			
SKE	SKIP IF A EQUALS M	If (A) = (M) skip	2-3
SKU	SKIP IF A UNEQUAL TO M	If $(A) \neq (M)$ skip	2-3
SKG	SKIP IF A GREATER THAN M	If $(A) > (M)$ skip	2-3
SKL	SKIP IF A LESS THAN M	If $(A) < (M)$	2-3
SKR	REDUCE M, SKIP IF NEGATIVE	(M)-1→M If $(M) < 0$ skip	3
SKM	SKIP IF A = M ON B MASK	If (A)(B) = (M)(B) skip	2-3
SKN	SKIP IF M NEGATIVE	If $(M) < 0$ skip	2-3
SKA	SKIP IF M AND A DO NOT COMPARE ONES	If (A)(M) = 0 skip	2-3
SKB	SKIP IF M AND B DO COMPARE ONES	If $(B)(M) \neq 0$ skip	2-3
SKP	SKIP IF BIT SUM EVEN	If sum of (B)(M) even skip	2-3
SKS	SKIP IF SIGNAL NOT SET		1-2
SKF	SKIP IF EXPONENT IN B ≥ M	If $(B)_{15-23} \geq (M)_{15-23}$ skip	2-3
CONTROL			
HLT	HALT	Halts Computation	1
NOP	NO OPERATION	- - -	1
EXU	EXECUTE	Execute the instruction in M	1
EAX	COPY EFFECTIVE ADDRESS INTO INDEX REGISTER 1	$M\rightarrow X_1$	2
REP	REPEAT INSTRUCTION IN M	Repeat instruction in M until $X_i \geq 0$ or skip occurs.	3+N(T-1)

*FR - FLAG REGISTER (6 Bits)

Designation	Name	Function	Timing in Machine Cycles
INPUT/OUTPUT			
EQM	ENERGIZE OUTPUT M	1.75 sec pulse to external device M	1
PIN	PARALLEL INPUT	External Lines→M	4 + wait
POT	PARALLEL OUTPUT	M→External Lines	3 + wait
LCH	LOAD AUTOMATIC DATA	(M, M+1)→I/O Channel$_k$	3
SET AND TEST INDICATORS (If no skip, the shorter execution time prevails)			
STI	SET AND TEST INDICATORS		1-2
LFR	LOAD FLAG REGISTER	M→FR	1-2
FRST	FLAG REGISTER SET TEST	No skip if flag set	1-2
FRRT	FLAG REGISTER RESET TEST	No skip if flag reset	1-2
SWT	SENSE SWITCH TEST	No skip if sense switch set set and/or reset.	1-2
SHIFT GROUP			
RSA	RIGHT SHIFT A		2-7
RSB	RIGHT SHIFT B		2-7
RSD	RIGHT SHIFT DOUBLE		2-7
RSAB	RIGHT SHIFT A AND B		2-7
LRSA	LOGICAL RIGHT SHIFT A		2-7
LRSD	LOGICAL RIGHT SHIFT DOUBLE		2-7
LRAB	LOGICAL RIGHT SHIFT A AND B		2-7
RCA	RIGHT CYCLE A		2-7
RCB	RIGHT CYCLE B		2-7
RCD	RIGHT CYCLE DOUBLE		2-7
RCAB	RIGHT CYCLE A AND B		2-7
LSA	LEFT SHIFT A		2-5
LSB	LEFT SHIFT B		2-5
LSD	LEFT SHIFT DOUBLE		2-5
LSAB	LEFT SHIFT A AND B		2-5
LLSA	LOGICAL LEFT SHIFT A		2-5
LLSD	LOGICAL LEFT SHIFT DOUBLE		2-5
LLAB	LOGICAL LEFT SHIFT A AND B		2-5
LCA	LEFT CYCLE A		2-5
LCB	LEFT CYCLE B		2-5
LCD	LEFT CYCLE DOUBLE		2-5

Designation	Name	Function	Timing in Machine Cycles
SET AND TEST INDICATORS (Continued)			
LCAB	LEFT CYCLE A AND B		2-5
NORA	NORMALIZE A		2-5
NORD	NORMALIZE DOUBLE		2-5
REGISTER CHANGE GROUP			
STZ	STORE ZERO	$0 \rightarrow M$	3
DPN	DOUBLE PRECISION NEGATE	$-(A, B) \rightarrow A, B$, fixed or floating	3
CNA	COPY NEGATIVE TO A	$-(A) \rightarrow A$	1
CLA	CLEAR A	$0 \rightarrow A$	1
CLB	CLEAR B	$0 \rightarrow B$	1
CLAB	CLEAR AB	$0 \rightarrow A, B$	1
CAB	COPY A INTO B	$(A) \rightarrow B$	1
CIAB	COPY INVERSE A TO B	$-(A) \rightarrow B$	1
CBA	COPY B TO A	$(B) \leftrightarrow A$	1
XAB	EXCHANGE A AND B	$(A) \rightarrow (B)$	1
CIBA	COPY INVERSE B TO A	$-(B) \rightarrow A$	1
ABC	COPY A INTO B, CLEAR A	$(A) \rightarrow B, 0 \rightarrow A$	1
BAC	COPY B INTO A, CLEAR B	$(B) \rightarrow A, 0 \rightarrow B$	1
MAB	MERGE A INTO B	$(A) \oplus (B) \rightarrow B$	1
MBA	MERGE B INTO A	$(B) \oplus (A) \rightarrow A$	1
FMB	FORM MASK IN B	$1's \rightarrow B$	1
FMA	FORM MASK IN A	$1's \rightarrow A$	1
CXA	COPY INDEX INTO A	$(X_i) \leftrightarrow A$	1
CXB	COPY INDEX INTO B	$(X_i) \leftrightarrow B$	1
CAX	COPY A INTO INDEX	$(A) \rightarrow X_i$	1
CBX	COPY B INTO INDEX	$(B) \rightarrow X_i$	1
CX1X	COPY INDEX 1 TO INDEX	$(X_1) \rightarrow X_i$	1
CX2X	COPY INDEX 2 TO INDEX	$(X_2) \rightarrow X_i$	1
CS3X	COPY INDEX 3 TO INDEX	$(X_3) \rightarrow X_i$	1
XXA	EXCHANGE INDEX AND A	$(X_i) \rightarrow (A)$	1
XXB	EXCHANGE INDEX AND B	$(X_i) \rightarrow (B)$	1
CLX	CLEAR INDEX	$0 \rightarrow X_i$	1
AXB	ADDRESS TO INDEX BASE	$M \rightarrow X_{i\ 9-23}$	1

Designation	Name	Function	Timing in Machine Cycles
REGISTER CHANGE GROUP (Continued)			
BRC	BRANCH AND CLEAR INTERRUPT	i = 0; M→P clear interrupt	2
BMA	BRANCH AND MARK PLACE OF ARGUMENT ADDRESS		3
(illegible)	SKIP IF MASK QUANTITY IN A GREATER THAN M		2-3