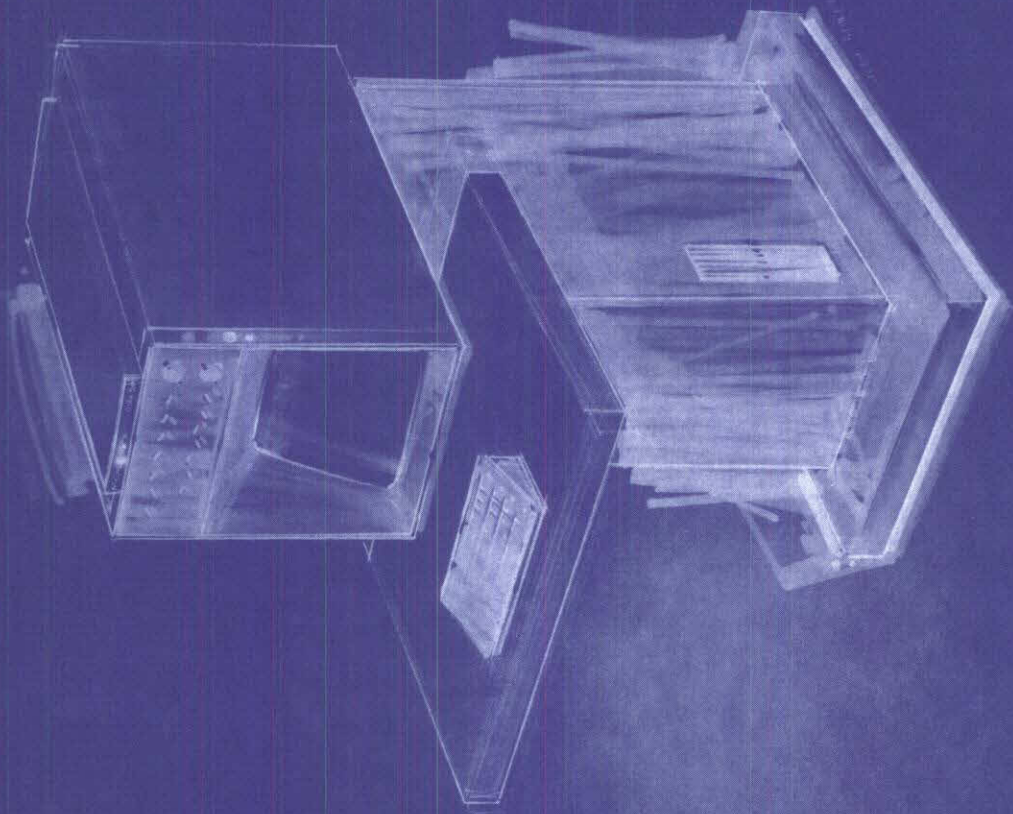


MEMOREX 7100

SYSTEM REFERENCE MANUAL

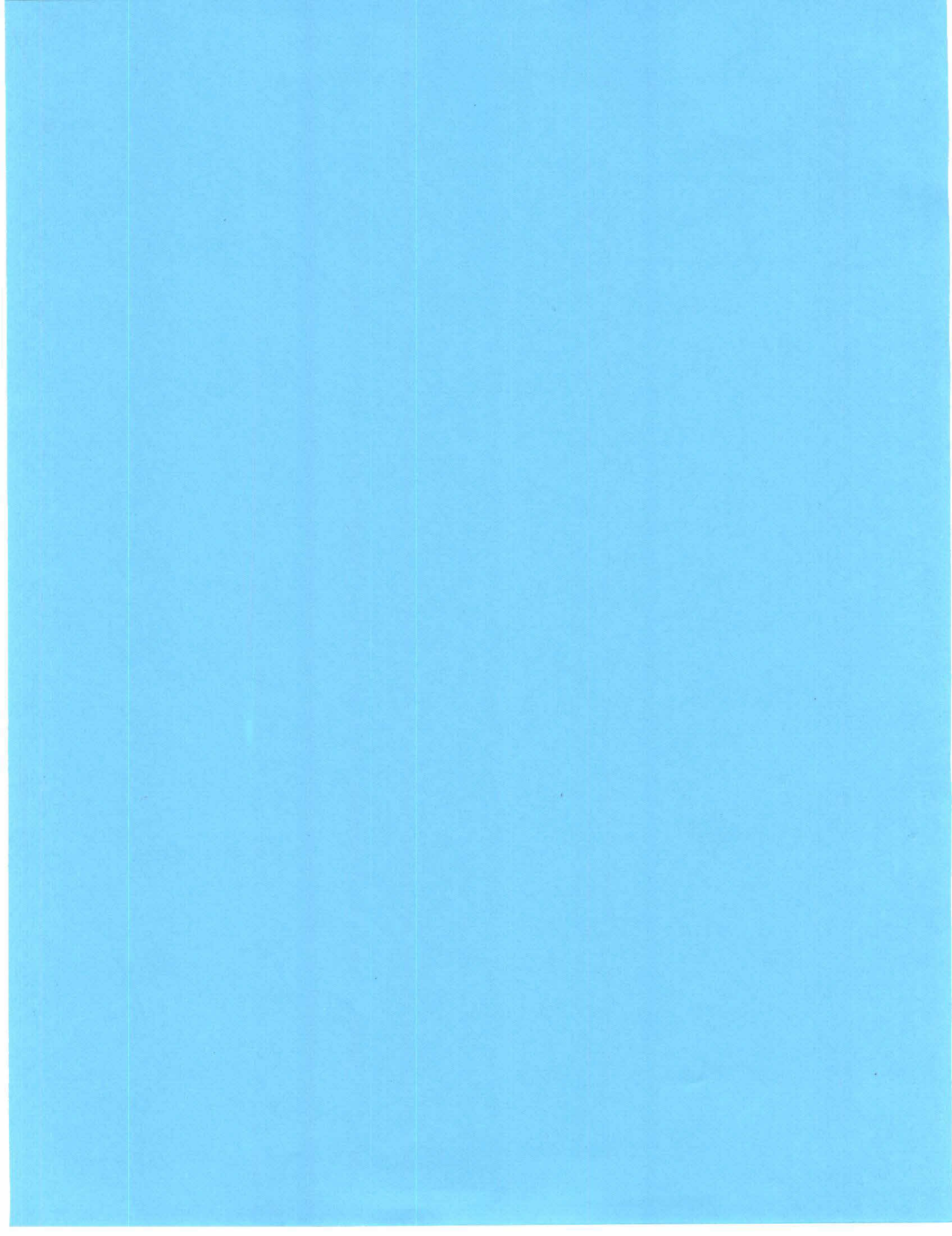
7/19/72

MEMOREX CONFIDENTIAL



CONTENTS: This Memorex 7100 System Reference Manual contains the following papers authored by the members of the 7100 Design Team:

INTRODUCTION	M. Gregory
SYSTEM ARCHITECTURE	J. Miller
CPU ARCHITECTURE AND MICRO-PROGRAMMING	L. Conway
CPU LOGIC DESIGN	G. Yee, R. Stallman
MEMORY SYSTEM	R. Peterson
SYSTEM CONTROL AND DISPLAY PANEL	A. Heme1
I/O SYSTEM	R. Hoehnle, R. Holland, D. Pesavento
PHYSICAL DESCRIPTION	M. Gregory
POWER SUPPLY	G. Ewart
OPSYS1 EMULATION PACKAGE	R. Chueh, R. Hoehnle
MRX30 EMULATION AND PERFORMANCE	J. Miller
MRX30 PHASE "0" COST ESTIMATE	M. Gregory



MEMOREX 7100

INTRODUCTION

M. GREGORY

7/19/72

MEMOREX CONFIDENTIAL

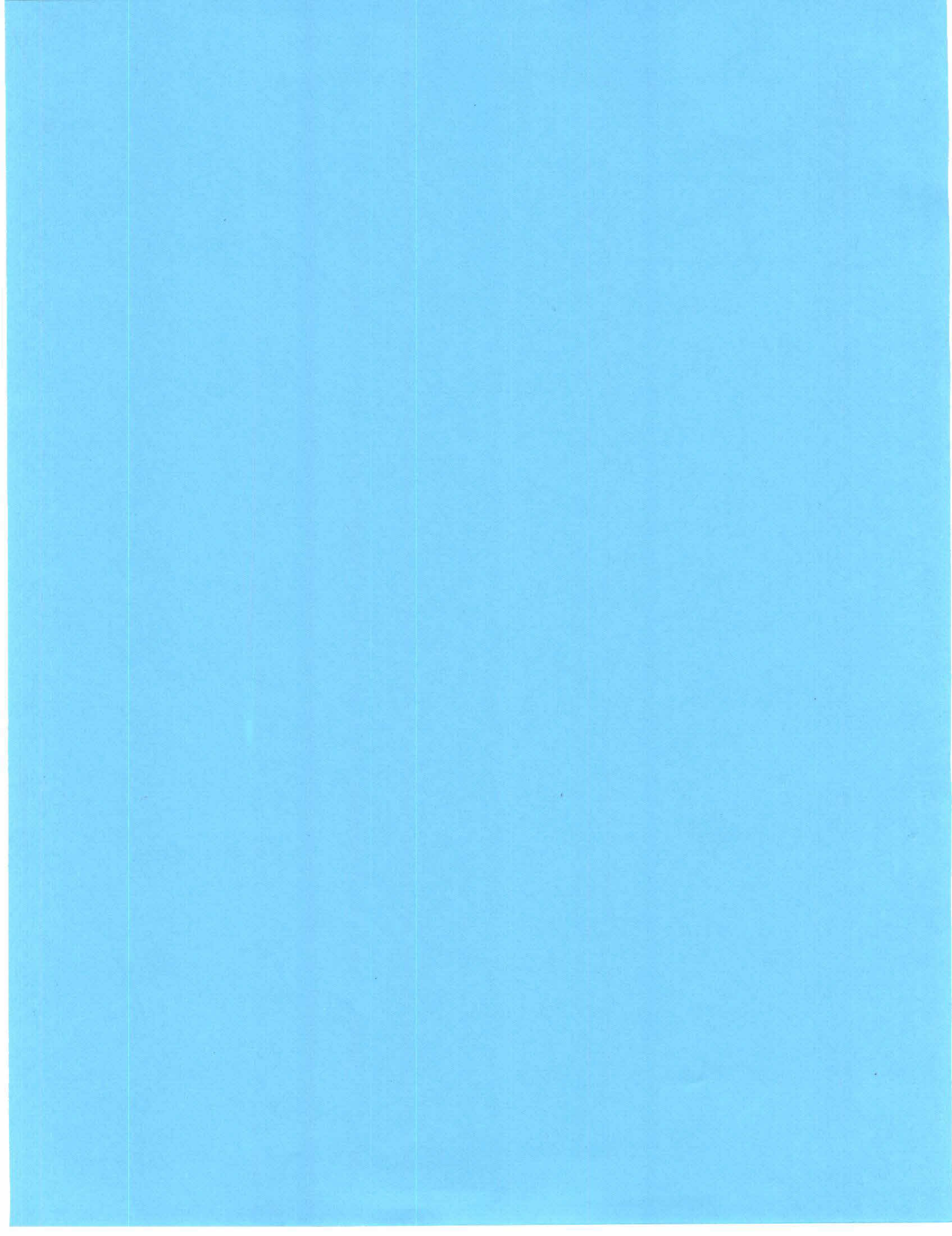
The Memorex 7100 System is the result of a three month design effort by the 7100 Design Team. The primary charter given the team was to "design a processor building block which could be economically used across a variety of product lines".

The primary design parameter was cost. The cost must be low enough so that the processor can be used in several market applications. However, performance of the 7100 System when used as a MRX30 is required to equal an IBM System/3. Thus the design centered around developing a minimum cost CPU which would enable the 7100 System to have the required performance.

The 7100 CPU is an inexpensive micro-programmed processor which gives the 7100 System the required performance. It has been tailored to operate in a communications oriented environment, and may be used in non-programmable applications as well as programmable ones. It contains the hardware necessary to operate under OPSYS1, thus giving the 7100 System upward compatibility with the MRX40 and MRX50 when used as a general purpose small commercial batch system.

The 7100 System design is based on implementation in T²L logic. The use of MOS-LSI for various components of the system is under investigation.

This Memorex 7100 System Reference Manual is a collection of papers by the members of the 7100 Design Team which give a detailed description of the components of the 7100 System. Included are detailed analyses of system performance and cost.



MEMOREX 7100

SYSTEM ARCHITECTURE

J.A. MILLER

7/19/72

MEMOREX CONFIDENTIAL

INTRODUCTION

The 7100 system consists of four separate subsystems, the CPU, Control and Display (C&D), Memory, and I/O (See Figure 1). The CPU is a microprogrammed general purpose processor using a 16 bit microinstruction word. It has a 16 bit data path for 2's complement binary arithmetic, and several special purpose and general purpose registers. The C&D subsystem is the panel lights and switches used to manually control and monitor the system. The memory subsystem consists of a maximum of 32768 16 bit words of MOS-LSI storage. The memory may be referenced on either a word or byte basis. The I/O subsystem consists of a number of device adapters logically integral with the CPU, each of which may control one or more I/O devices. These I/O adapters may access memory directly without going through the CPU. Each of these component subsystems is described in detail elsewhere.

These subsystems are interconnected via two independent data paths, the memory bus and the micro bus. In addition, an interrupt system connects the I/O subsystem with the CPU. The purpose of this section is to describe in general the interaction of the subsections via these busses and control lines, and to discuss the overall 7100 system timing.

THE MEMORY BUS

The memory bus is the path whereby the elements of the system (the CPU and the various I/O adapters) can access the memory subsystem. The interface to the memory bus is described in detail in the memory subsystem writeup.

Since several elements may desire concurrent access to memory, these elements may not access memory directly, but rather must request that

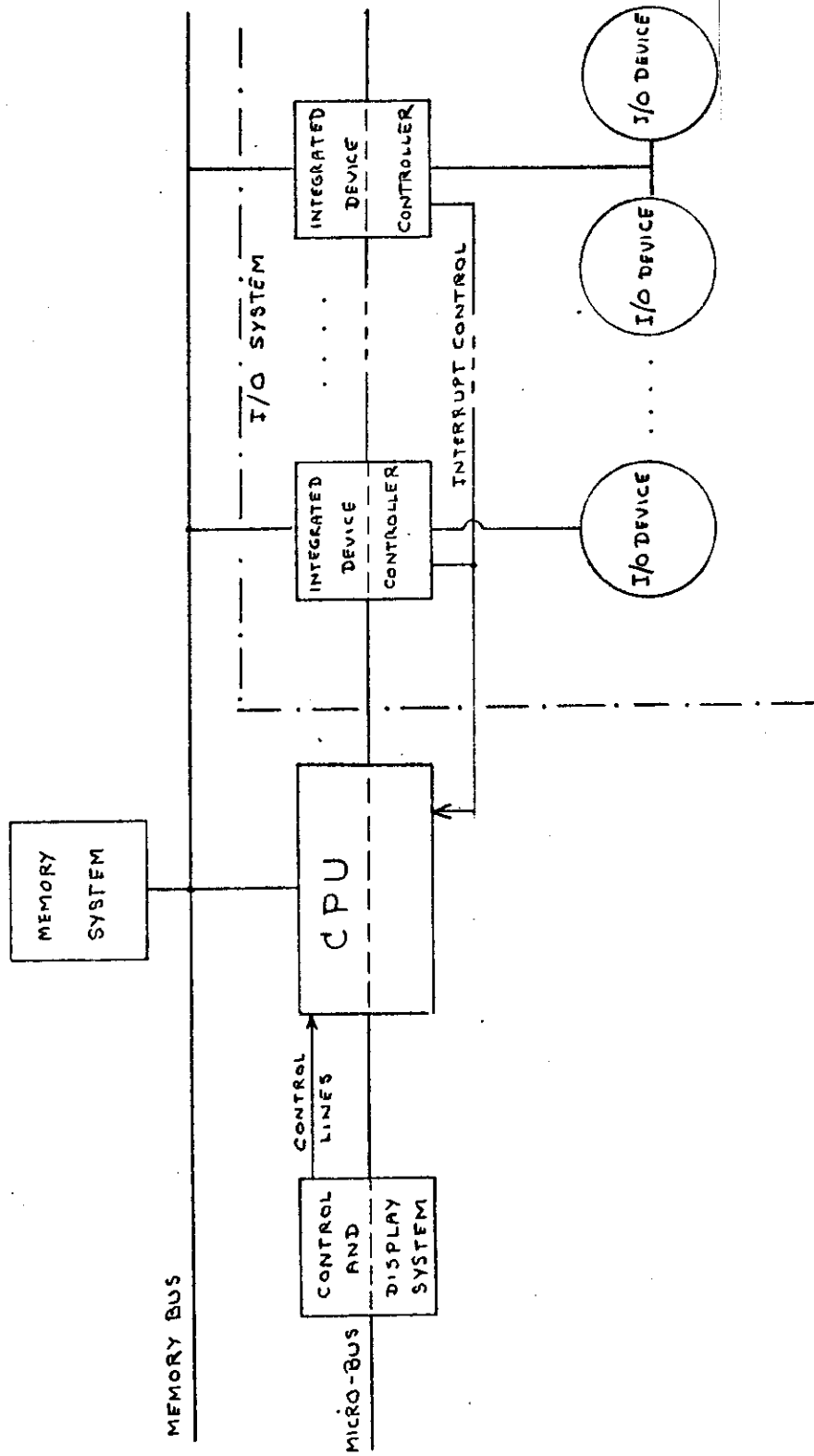


FIGURE 1. 7100 SYSTEM ARCHITECTURE

access first. The memory bus controls access to memory on a fixed priority basis. Since each element cannot control when access to memory is granted, the memory bus provides controls back to the elements that indicate both when access has been granted, and when the requested operation is complete.

THE MICRO BUS

The micro bus connects the C&D and I/O subsystems with the CPU. It provides a direct data and control path between those two subsystems and the CPU. The relationship between the CPU and C&D subsystem is rather special and is described in detail in the C&D writeup. It is sufficient to say that the C&D subsystem is permitted some degree of control over the CPU and the microbus.

The interface between the I/O subsystem and the microbus is described in the I/O writeup. To each of the device adapters, the microbus appears as an 8 bit address bus, a 16 bit data bus, and read and write control lines. There are no interlocking signal sequences nor any bus control lines returning from the device adapters to the CPU. The CPU retains complete control over the micro bus at all times. The device adapters use the system clock to control their bus operations.

Within the CPU the micro bus becomes the internal CPU microbus. Operations over the microbus to the I/O subsystem are the same as operations between the internal components in the CPU. Indeed, the CPU contains sets of registers that, for microprogram purposes, may be dedicated to the individual I/O adapters.

The general register address space within the CPU admits of 256 addresses. These may be viewed as 16 groups of 16 registers each. (16 column addresses X 16 row addresses). There are not actually 256 registers in the CPU. Those addresses that have no associated registers become operations directed to the I/O subsystem.

The 8 bit register address is formed of two components. The 4 low order bits (row address) reference one of the 16 registers within a group, and arise directly from a microinstruction. The 4 high order bits (column address) reference a register group, and arise from either the X (extended register pointer) register or the P (priority interrupt) register of the

THE INTERRUPT SYSTEM

The priority interrupt system consists of an interrupt request line for each device adapter. These lines enter the CPU where a priority encoder encodes the address of the highest priority device adapter whose request line is active. This address is loaded into the P (priority interrupt) register under microprogram control. Since P also controls the register addressing this action also permits the CPU to direct I/O operations at the interrupting device adapter. The column address recognized by a device adapter must correspond to its interrupt request line.

SYSTEM TIMING

The basic clock that is used throughout the system is a 400 nanosecond clock. This clock is available to all subsystems within the 7100.

Within the CPU this clock is used for microcycle timing. It is the time required to execute one microinstruction.

The memory subsystem works on a 1.2 microsecond cycle time. This

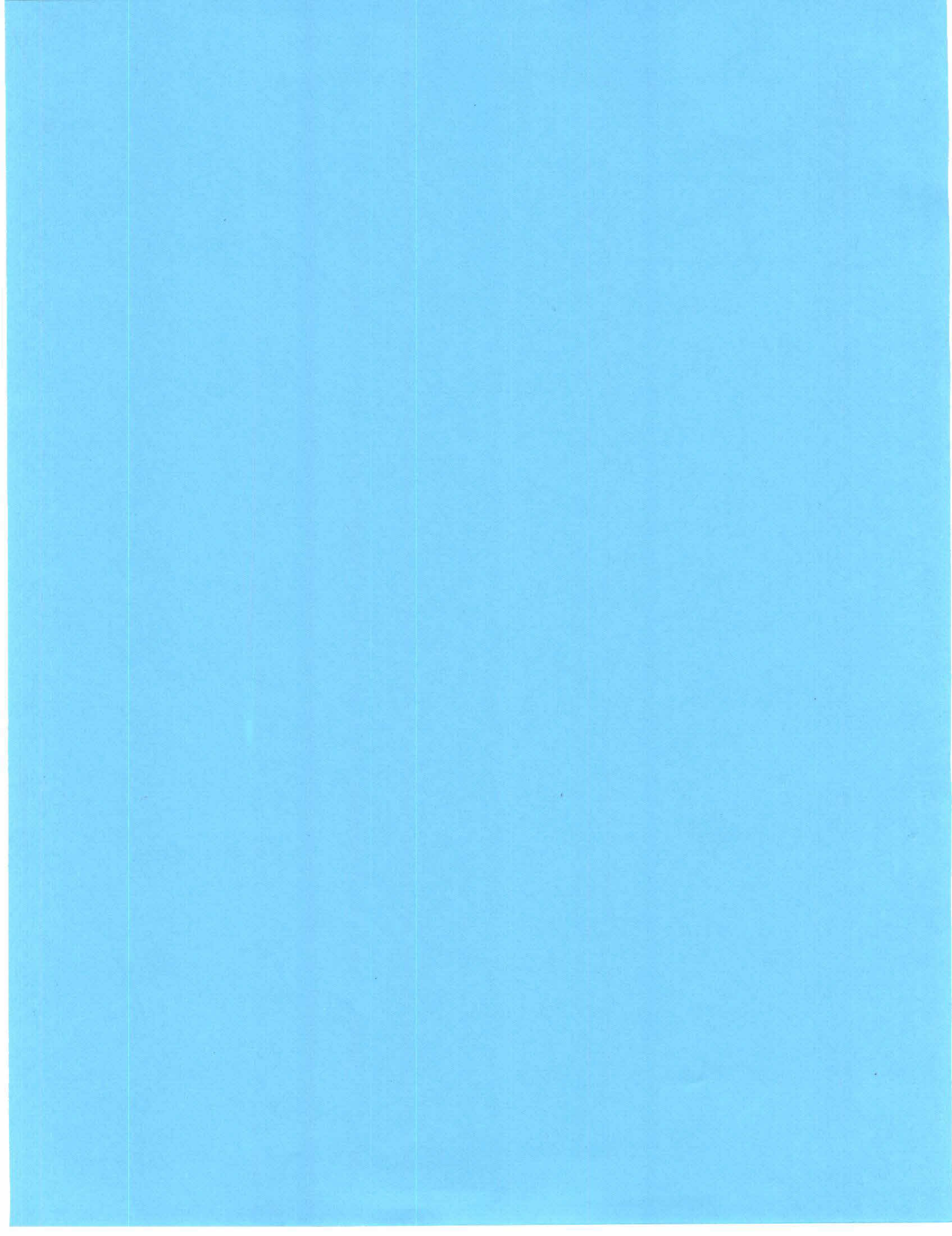
1.2 microsecond cycle represents 3 system clock times. The memory may start on any system clock pulse.

The system clock is also sent to the device adapters that comprise the I/O subsystem. It may there be used in any way the adapter desires. It must be used to control information passing between the CPU and the adapter via the microbus.

TABLE 1

7100 SYSTEM SPECIFICATIONS
SUMMARY

o MEMORY SIZE	Up to 64K bytes (128K bytes with option)
o MEMORY WORD SIZE	16 bits + 2 Parity bits
o MEMORY SPEED	1.2 μ s cycle .8 μ s access
o DIRECT MEMORY ACCESS	Yes, up to 7 modules
o DMA TRANSFER RATE	750K bytes/sec.
o MEMORY TYPE	MOS-LSI (2048 bit chips)
o PROCESSOR CYCLE TIME	400 ns
o PROCESSOR REGISTER FILE	16-16 bit registers
o MICRO-INSTRUCTION WORD	16 bits
o PROCESSOR TECHNOLOGY	T ² L Series 74
o I/O SYSTEM	Direct & DMA
o I/O CONTROL UNITS	7 max
o INTEGRATED ADAPTERS	Yes; up to 7
o DEVICE ADDRESSING	Through register file
o I/O TRANSFER RATE	30KB direct, 750KB DMA



MEMOREX 7100

CPU ARCHITECTURE AND MICRO-PROGRAMMING

L. CONWAY

7/19/72

MEMOREX CONFIDENTIAL

<u>CONTENTS</u>	<u>PAGE</u>
<u>1.0 INTRODUCTION</u>	1
<u>2.0 7100 CPU ARCHITECTURE</u>	2
2.1 7100 CPU CONTROLS	2
2.2 7100 CPU DISCRETE REGISTERS	4
2.3 7100 CPU ROMS	6
2.4 7100 CPU RAM REGISTERS	6
<u>3.0 MICRO-INSTRUCTION SET</u>	7
3.1 SOURCE AND DEST FIELD CODES	9
3.2 ALU OPERATIONS	12
3.3 PORT OPERATIONS	27
3.4 BRANCH CONDITION CODES	31
<u>4.0 MICRO-ASSEMBLY LANGUAGE</u>	31
4.1 COMMENT CARDS	32
4.2 GENERAL CARD FORMAT	32
4.3 LABELS	32
4.4 VALUES	33
4.5 ASSEMBLY CONTROL STATEMENTS	33
4.6 DEFINE CONSTANT STATEMENTS	33
4.7 MICRO-INSTRUCTION STATEMENTS	34
<u>5.0 MICRO-PROGRAMMING TECHNIQUES</u>	36
5.1 SOURCE-DESTINATION RESTRICTIONS	36
5.2 SCB BRANCH TIMING	37
5.3 MEMORY PORT OPERATION TIMING	38
5.4 DECIMAL ALU OP TIMING	40

TABLESPAGE

TABLE 1.	7100 MICRO-INST SET FORMATS	8
TABLE 2.	SOURCE AND DEST FIELD CODES	10
TABLE 3.	ALU OP CODES	25
TABLE 4.	PORT OPERATIONS	28
TABLE 5.	BRANCH CONDITION CODES	29
TABLE 6.	SUMMARY OF MICRO-INSTRUCTION FIELD CODES	30

FIGURESPAGE

FIGURE 1.	7100 CPU ARCHITECTURE	3
FIGURE 2.	CONDITION REGISTER SETTING	26
FIGURE 3.	CPU - MEMORY PORT TIMING	41

1.0 INTRODUCTION

This manual documents and describes the Memorex 7100 CPU architecture, and the 7100 micro-instruction set. Also included is a description of an assembly language for the symbolic encoding of 7100 micro-programs, and some examples of techniques for micro-programming the 7100.

The Memorex 7100 is a micro-programmed special purpose processor. It executes Micro-instructions fetched from a "read-only" memory. The 7100 may be micro-programmed to function as a general purpose processor (MRX/30), in which case it would emulate the execution of (MRX/30) instructions fetched from a "read-write" memory. The 7100 may also be micro-programmed to serve other special purposes. For example, it could be micro-programmed to function as a communications adapter.

The 7100 CPU is designed to rapidly execute (by micro-code emulation) 16-bit instructions from a main memory, using simply-structured, minimum-cost hardware.

The purpose of this manual is to document the 7100 CPU design at the architectural level, and to serve as a reference manual for those who design and implement the various micro-programs for the 7100.

2.0 7100 CPU ARCHITECTURE

This section describes the 7100 CPU architecture diagrammed in Figure 1.

The 7100 CPU is organized primarily as a set of 16-bit registers on a 16-bit bus. Many micro-program functions are implemented simply by gating data from one register (SOURCE) to another (DEST), via the bus.

Additionally, a simple ALU which has two input feeder registers (A and B) may be used to perform logical or arithmetic operations on data (in A and B) prior to gating the resulting data to a register (DEST) via the bus.

Some of the registers are discrete and serve specific functions. Of these, some may only function as SOURCES or DESTINATIONS of gating operations but not as both. Figure 1 indicates symbolically which function(s) each specific register may serve.

Other registers are contained in a RAM. These are used as either SOURCES or DESTS and serve as registers for MRX/30 register emulation and MRX/30 Input-Output emulation.

The 7100 is controlled by fetching and executing micro-instructions from a "read-only" memory (UROM).

2.1 7100 CPU CONTROLS

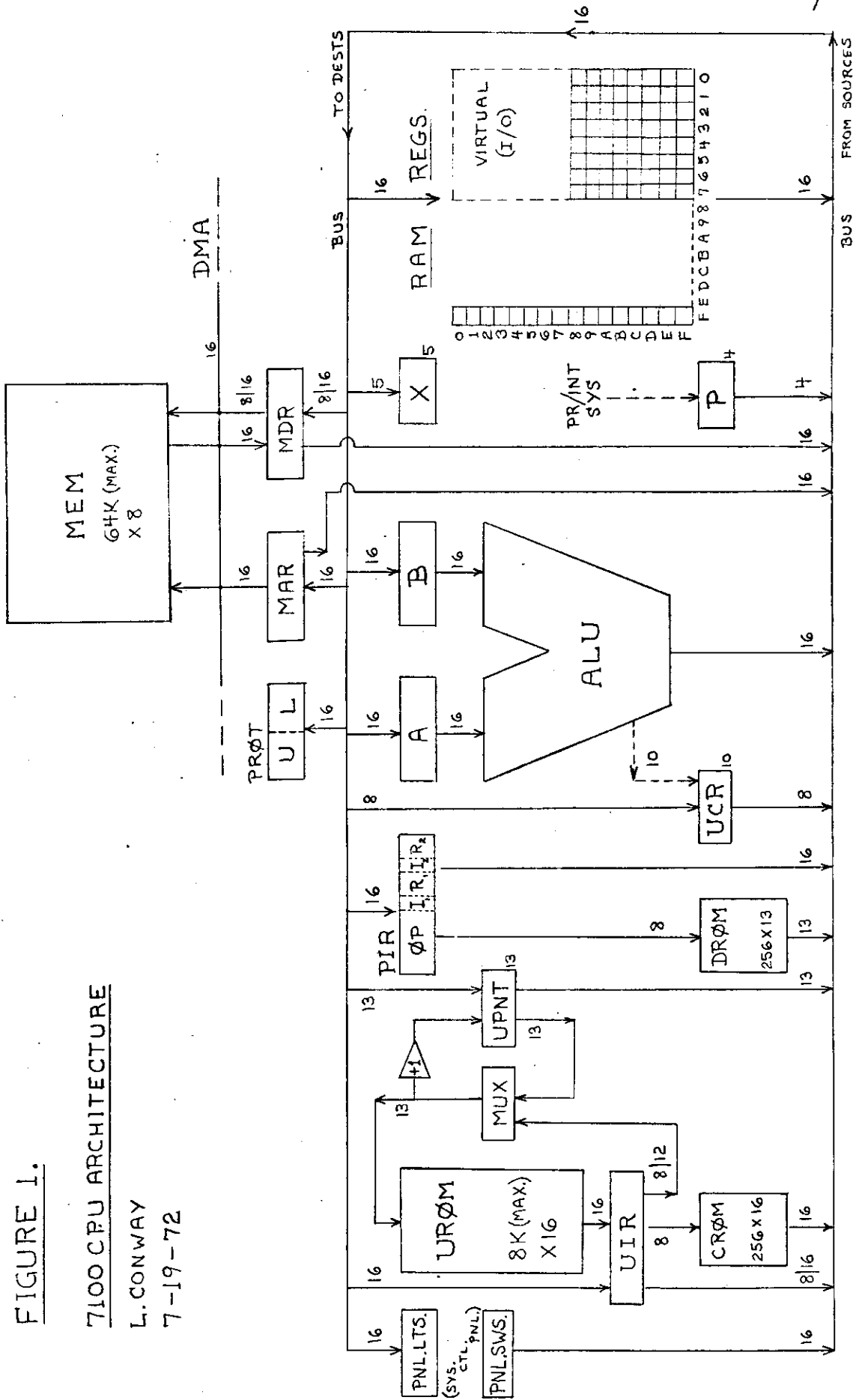
Each micro-cycle (400 ns), the 7100 fetches a micro-instruction from an address in UROM determined by the preceding instruction. The micro-instruction is fetched into the micro-instruction register (UIR). The UIR feeds the control logic which decodes and executes the micro-instruction.

FIGURE 1.

7100 CPU ARCHITECTURE

L. CONWAY

7-19-72



Encoded in the 16 bit micro-instruction are its FORMAT, and the desired SOURCE and DEST registers for gating operations or the OP and DEST for ALU operations. For gating and ALU type operations, the address of the next instruction to be fetched into UIR is formed by incrementing the current address at the time of access and holding the incremented value in the micro-instruction pointer register (UPNT). This incremented value is then used to address the next instruction fetched for execution the following micro-cycle.

Certain FORMATS, however, indicate a BRANCH instruction. These instructions may alter the sequence of instruction execution by specifying a SOURCE for the next instruction address other than the incremented value stored in UPNT.

The details of micro-instruction encoding are fully described in a later section of this manual.

2.2 7100 CPU DISCRETE REGISTERS

The 7100 CPU contains a number of discrete registers with specific, unique functions. These may be functionally separated into 3 groups: control registers, memory interface registers, and data feeders.

The following registers are control registers: UIR, UPNT, PIR, UCR, X, P, and PROT.

UIR, the micro-instruction register and UPNT, the micro-instruction pointer register have already been described (Section 2.1).

PIR, the Program Instruction Register, is used to hold MRX/30 instructions during MRX/30 emulation. An instruction decode read-only memory (DROM) is used to decode the 8-bit OP field (left 8 bits) of PIR and provide a 13-bit jump address for emulation micro-code OP decode jumps. Additionally, logic decodes the I₁, R₁, I₂, R₂, fields of the PIR so that conditional micro-code branches may be

made on these fields being zero/non-zero. $I_1 = \text{PIR}(8)$, $R_1 = \text{PIR}(9-11)$, $I_2 = \text{PIR}(12)$, $R_2 = \text{PIR}(13-15)$.

The UCR is a 10-bit register, whose bits are set/reset as the result of ALU logical or arithmetic operations. Conditional micro-code branches may be made on each specific bit of the UCR (micro-condition register) being zero/one.

Registers X and P are used in RAM register addressing which is described in Section 2.4. PROT is used for memory write protection, described below.

The Memory Address Register (MAR) and the Memory Data Register (MDR) are used as interface registers for memory operations. Encoded in certain micro-instructions are "PORT" operations which initiate/control CPU - memory data transfers.

The MAR holds the memory address for these data transfers. The 16-bit MAR can address up to 64K bytes in main memory. The MDR holds the data for writes and receives the data for reads. The PROT (protect) register participates in the control of protected writes. It holds 8-bit fields, U and L, which are the upper and lower bounds of the high order 8-bits of the write address in MAR. If a protected write is attempted out of bounds, it is not executed and an error condition is set.

The data feeder registers A and B are used to hold the input data for ALU operations.

The 'registers' PNL.LTS. and PNL.SWS. in Figure 1 represents the system control panel display lights and data-entry switches. During single-cycle operation of the CPU (under system control panel control) various CPU registers (UIR, UPNT, MAR, MDR) may be displayed in the lights or altered via the switches. These functions are described in detail in the reference manual "MEMOREX 7100 SYSTEM CONTROL PANEL", by A. Hemel. The panel switches may also be read as sources by micro-instructions.

2.3 7100 CPU ROMS

The 7100 CPU data flow contains three read-only memories (ROMS), the UROM, CROM, and DROM.

The UROM is 8192 (max) 16-bit words. This is the memory which holds the micro-instructions which control the 7100. Note that 13 bit addresses are required to address the span of the memory (8K).

The CROM is a smaller read-only memory used to hold 16-bit constants. Certain micro-instructions may specify an 8-bit CROM (constant ROM) address and thus select one of CROM's 256 constants for gating onto the bus to a destination.

The DROM (decode ROM) is a small 256 by 13-bit read only memory used to decode the 8-bit OP field of the PIR to obtain a 13-bit jump address.

2.4 7100 RAM REGISTERS

The 7100 CPU contains 80 registers (16-bit) in a small Random Access Memory (RAM), as indicated in Figure 1.

Micro-instructions may address these RAM registers and use them as sources or destinations of data to/from the bus. Such micro-instructions encode the 'row' of the RAM directly into their SOURCE or DEST field code. This selects one of the 16 rows of the RAM for addressing.

The 'column' (one of 16) is selected by the data (4-bits) in the P or X register at the time the RAM is accessed by a micro-instruction. If the high order bit in the (5-bit) X register is 0, then P is used for RAM column selection. If the high order X bit is 1, then X is used.

Of the 256 possible RAM addresses (16 rows by 16 columns), only 80 actually contain CPU registers. Certain of the remaining addresses correspond to registers or command codes for devices external to the CPU (I/O).

This organization of the RAM register file enables the CPU to communicate with I/O devices in the same simple manner in which it uses its own internal registers. It also enables the "state" of the CPU (corresponding to the value in P) to be switched under control of a Priority-Interrupt System. These functions are described in detail in the reference manual "MEMOREX 7100 I/O SYSTEM".

3.0 7100 MICRO-INSTRUCTION SET

The 7100 CPU executes 16-bit micro-instructions. The first 3-bits of each micro-inst encode its FORMAT. Table 1 lists the various FORMATS of the micro-instruction set. Seven of the eight possible are currently defined.

The micro-instruction set logically groups into 3 basic types of instructions:

- (i) ALU operations. FORMATS: ALU
- (ii) GATING operations. FORMATS: GSD, GCD, GID
- (iii) BRANCH operations. FORMATS: SCB, ACB, UCB

The ALU type of micro-instruction specifies that the ALU operation encoded into its OP field be performed on the data in feeder registers A and B with the result bused to a specified DEST register.

The GATING type of micro-instruction specifies a SOURCE for data and a DEST to which that data is to be bused. In the GSD FORMAT, the source is an addressed CPU register. In the GCD FORMAT, the source is an addressed CROM constant. In the GID FORMAT, the source is 8-bits of immediate data in the micro-instruction itself (R.J. on bus).

The BRANCH type of micro-instruction may alter the sequence of micro-instruction execution by specifying some address for the next instruction other than the incremented current address. In the case of the UCB FORMAT, the BRANCH is "Unconditional" and is always taken.

TABLE 1
7100 MICRO-INST SET FORMATS

<u>MNEM</u>	<u>FORMAT</u>	<u>DESCRIPTION</u>										
ALU	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> </tr> <tr> <td style="text-align: center;">000</td> <td style="text-align: center;">DEST</td> <td style="text-align: center;">PORT</td> <td style="text-align: center;">OP</td> </tr> </table>	3	5	3	5	000	DEST	PORT	OP	ALU 'OP' performed on feeder REGS 'A' and 'B', result gated to destination.		
3	5	3	5									
000	DEST	PORT	OP									
GSD	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> </tr> <tr> <td style="text-align: center;">001</td> <td style="text-align: center;">DEST</td> <td style="text-align: center;">PORT</td> <td style="text-align: center;">SOURCE</td> </tr> </table>	3	5	3	5	001	DEST	PORT	SOURCE	Gate source contents to destination.		
3	5	3	5									
001	DEST	PORT	SOURCE									
GCD	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> <td style="text-align: center;">8</td> </tr> <tr> <td style="text-align: center;">010</td> <td style="text-align: center;">DEST</td> <td style="text-align: center;">CROM ADDR</td> </tr> </table>	3	5	8	010	DEST	CROM ADDR	Gate the 16-bit constant at 'CROM ADDR' to destination.				
3	5	8										
010	DEST	CROM ADDR										
GID	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> <td style="text-align: center;">8</td> </tr> <tr> <td style="text-align: center;">011</td> <td style="text-align: center;">DEST</td> <td style="text-align: center;">IMM DATA</td> </tr> </table>	3	5	8	011	DEST	IMM DATA	Gate IMM data to destination (Right justified, zero fill).				
3	5	8										
011	DEST	IMM DATA										
—	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> </tr> <tr> <td style="text-align: center;">100</td> </tr> </table>	3	100	Unused.								
3												
100												
SCB	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">1</td> <td style="text-align: center;">4</td> <td style="text-align: center;">3</td> <td style="text-align: center;">5</td> </tr> <tr> <td style="text-align: center;">101</td> <td style="text-align: center;">R</td> <td style="text-align: center;">COND</td> <td style="text-align: center;">PORT</td> <td style="text-align: center;">SOURCE</td> </tr> </table>	3	1	4	3	5	101	R	COND	PORT	SOURCE	Conditional branch to ADDR contained in 'SOURCE' if 'COND' true, and R = 0. BR on 'COND' false if R = 1.
3	1	4	3	5								
101	R	COND	PORT	SOURCE								
ACB	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">1</td> <td style="text-align: center;">4</td> <td style="text-align: center;">8</td> </tr> <tr> <td style="text-align: center;">110</td> <td style="text-align: center;">R</td> <td style="text-align: center;">COND</td> <td style="text-align: center;">ADDR ON PG</td> </tr> </table>	3	1	4	8	110	R	COND	ADDR ON PG	Conditional branch to ADDR on current page if 'COND' true, and R = 0. BR on 'COND' false if R = 1.		
3	1	4	8									
110	R	COND	ADDR ON PG									
UCB	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">3</td> <td style="text-align: center;">1</td> <td style="text-align: center;">12</td> </tr> <tr> <td style="text-align: center;">111</td> <td style="text-align: center;">S</td> <td style="text-align: center;">ADDR</td> </tr> </table>	3	1	12	111	S	ADDR	Unconditional branch to ADDR in current 4K. If S = 0, then 'BRA'. If S = 1, then 'BSR' (UPNT + 1 gated to specific implied 'USRT' REG).				
3	1	12										
111	S	ADDR										

In that case the micro-instruction contains a 12 bit branch address for the next instruction (within the current half of the UROM as specified by the high order bit of UPNT). A special subroutine bit is also encoded in the UCB FORMAT. If it is one, then the processor saves the UPNT register in a specific register in the RAM for use later as a subroutine return.

The ACB and SCB FORMATS are "conditional branches". For these instructions to alter micro-instruction sequencing to their specified addresses, the indicated condition must be true (if R = 0) or false (if R = 1). ACB, if taken, causes the next instruction to be fetched at the address on the current 256 word "page" in the UROM given by the 8-bit "ADDR ON PG" encoded in the ACB micro-instruction. An SCB type BRANCH, if taken, specifies as the branch address the contents of the encoded SOURCE register.

Certain of these basic ALU, GATING, and BRANCH type micro-instructions have an additional "PORT" field. This 3-bit field in these instructions encodes memory initiation and control functions. The specific functions are described in Section 3.3.

3.1 SOURCE AND DEST FIELD CODES

This section describes the encoding of the SOURCE and DEST fields of the 7100 micro-instruction set. The locations of the fields in the micro-instructions are shown in Table 1, while the CODES and MNEMONICS are listed in Table 2.

All the SOURCE and DEST field codes are encoded into 5-bits. If the high order bit is zero, then the SOURCE or DEST is in the RAM registers and the low-order four bits form the "row" address for the RAM register. RAM registers may be either SOURCE's or DEST's, although timing restrictions preclude gating from one RAM register to another RAM register in the same micro-cycle.

If the high order bit of the code is one, then one of the CPU discrete registers or some variation is indicated (See Table 2).

TABLE 2

SOURCE AND DEST FIELD CODES

<u>CODE</u>	<u>SOURCE</u>	<u>DESTINATION</u>
00	REGO	REGO
1	1	1
1	1	1
1	1	1
1	1	1
0F	REGF	REGF
10	X	X
11	PIR	PIR
12	MDR	MDR
13	SMDR	SMDR
14	UPNT	PROT
15	R1I	R1I
16	R2I	R2I
17	SWS	A
18	PIRD	B
19	P	UIR
1A	MAR	MAR
1B	CST	AMAR
1C	UCR	UCR
1D	_____	_____
1E	_____	_____
1F	ZRO	NOP

Certain of the CPU discrete registers may be either micro-code SOURCES or DESTS. These are: X, PIR, MAR, MDR, and UCR. Note that UCR, while it holds 10 bits, has only its left 8 bits gated to/from the bus when used as a SOURCE or DEST. In both cases the 8 bits are left justified (L. J.) on the bus. Also, when UCR is used as a SOURCE, the right-most 8 bits of the bus are zero.

Some CPU registers may only be used as a source. These are UPNT and P. UPNT is used as an implied destination for the branch address during the execution of a SCB type branch. Thus, the Figure 1 shows a data path from the bus into UPNT. However, UPNT may not be directly encoded as a DEST.

The 'SWS' source code is used to read the system control panel data entry switches.

The 'ZRO' source code places all zero's on the bus.

The mnemonic 'PIRD', indicating PIR decode, appears as a SOURCE only. This code is used to gate the DROM decode of the OP field (8 high order bits) of the PIR as a SOURCE.

The mnemonic 'CST' appears as a SOURCE only. This is used as a special function in an otherwise NOP instruction (PORT may be used) to indicate that the P register may be updated to the new value specified by the priority-interrupt system (changes CPU 'state').

Some CPU registers may only be used as destinations. These are PROT, UIR, A, B, and AMAR. AMAR is a special DEST only code which enables the bus data to be simultaneously gated to two destination registers, A and MAR. A UIR DEST OR's the bus with the next fetched micro-instruction.

Note that there is a NOP DEST code. This code disables any gating to destinations.

'SMDR' is a special use of MDR. The right-justified byte on the bus is gated to/from the MDR half indicated by MAR₁₅ (0 = left, 1 = right).

R1I, R2I cause the indirect use of the R1 or R2 fields (3 bits) of the PIR as SOURCE/DEST codes. This enables indirect use of REG0 - REG7 in the RAM according to value of R1 or R2 field in the PIR.

3.2 ALU OPERATIONS (R. Stallman)

The ALU operations consist of three basic types; arithmetic functions; logic functions; and word, byte, nybl manipulations. All of these ops are performed on operands in the A and B registers. The ALU op codes are listed in table 3.

Bits 0-7 of the condition register (UCR) are dedicated to storing the status of the applicable ALU ops. The assignment of each bit location in the UCR (condition register) is diagrammed in table 2.

Arithmetic Functions

Arithmetic operations performed by the ALU are either binary or decimal. These two general categories differ from each other substantially. Binary ops: require one microcycle for their execution; involve two-16 bit operands and require binary numbers to be represented in 2's complement notation.

Decimal ops: require two micro-cycles for their execution; involve 8 bit (two decimal digits) operands and outputs; and require digits to be represented in packed BCD (8421) format.

Binary Arithmetic Ops

Signed binary values are represented by 15 bits of magnitude information with the 16th bit (MSB) representing the sign.

Negative numbers have a 1 in the sign position, and positive numbers, a 0. Numbers are also in 2's complement notation. Negative numbers are formed by deriving the 1's complement of the positive representation and adding a 1 to the least significant bit:

$$+ 17 = 010001$$

$$- 17 = 101111$$

In 2's complement addition, the 16 bit numbers (including sign) are added and the carry from the most significant (sign bit) is always ignored. Subtraction is performed by adding the 1's complement of B to A, and adding a 1 to the least significant bit position. In either addition or subtraction, the resultant answer is in 2's complement form, and no correction need be made. The overflow bit in the UCR is set according to the following:

ADDITION:

$$(BAD+BADX) \left[(A_0=0 \cdot B_0=0 \cdot ALU_0=1) + (A_0=1 \cdot B_0=1 \cdot ALU_0=0) \right]$$

SUBTRACTION

$$(BSB+BSBX) \left[(A_0=1 \cdot B_0=0 \cdot ALU_0=0) + (A_0=0 \cdot B_0=1 \cdot ALU_0=1) \right]$$

A_0 = Bit zero of the A Register (Sign Bit)

B_0 = Bit zero of the B Register (Sign Bit)

ALU_0 = Bit zero of the ALU Register (Sign Bit)

ADD, SUB

ADD and SUB are ops intended for use with address arithmetic. When executed, they do not change the condition register. Carries or borrows from the condition register into the ALU are inhibited during these ops. SUB is a 16-bit binary subtraction of B from A, producing no borrow or overflow in the condition register. ADD is a 16 bit binary addition of A and B, also producing no carry or overflow in the condition register.

BAD, BSB

BAD and BSB are ops intended for use in single precision 16-bit binary addition or subtraction. Carries or borrows into the ALU, from the condition register, are not enabled. Carries or borrows and overflow resulting from these ops are, however, stored in the condition register.

BADX, BSBX

BADX and BSBX are ops intended for use in multiple precision 16-bit binary additions or subtractions. Carries or borrows initially in the condition register are included in the addition or subtraction being performed and carries, borrows, or overflows resulting during the execution are stored in the condition register.

The following examples use 6 bits rather than 16 for simplicity:

Single-Precision ADDITION (ADD,BAD)

$$\begin{array}{r}
 -10 \\
 -3 \\
 \hline
 -13
 \end{array}
 \begin{array}{r}
 +110110 \\
 +111101 \\
 \hline
 1\ 110011
 \end{array}
 \begin{array}{r}
 +10 \\
 -3 \\
 \hline
 +7
 \end{array}
 \begin{array}{r}
 +001010 \\
 +111101 \\
 \hline
 1\ 000111
 \end{array}$$

↑ Carry-store in UCR if BAD
↑ Carry-store in UCR if BAD

Single-Precision SUBTRACTION (SUB,BSB)

$$\begin{array}{r}
 +10 \\
 -(-3) \\
 \hline
 +13
 \end{array}
 \begin{array}{r}
 +001010 \\
 +000010 \leftarrow \text{1's complement of } (-3) \\
 +\quad\quad 1 \leftarrow \text{Forced carry-in} \\
 \hline
 001101
 \end{array}
 \left. \begin{array}{l} \\ \\ \end{array} \right\} \begin{array}{l} \text{Done by} \\ \text{ALU} \end{array}$$

↑ No borrow - 0 in UCR if BSB

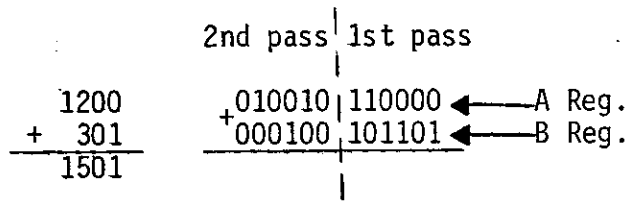
$$\begin{array}{r}
 +10 \\
 -(+3) \\
 \hline
 +7
 \end{array}
 \begin{array}{r}
 +001010 \\
 +111100 \leftarrow \text{1's complement of } (+3) \\
 +\quad\quad 1 \leftarrow \text{Forced carry-in} \\
 \hline
 1\ 000111
 \end{array}
 \left. \begin{array}{l} \\ \\ \end{array} \right\} \begin{array}{l} \text{Done by} \\ \text{ALU} \end{array}$$

↑ Borrow-store in UCR if BSB

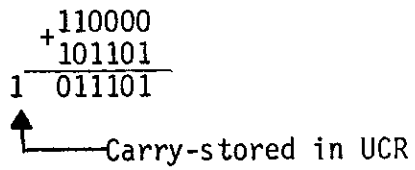
Multiple-precision additions or subtractions involve binary numbers larger than the capacity of the 16-bit A and B registers. If a 32 bit number is to be added to, or subtracted from, another 32 bit number, two passes through the ALU must be made. The first pass would be a BAD or BSB as illustrated, the second pass must include the carry or borrow produced in the first pass (BADX or BSBX). The

first pass (least significant word) would not include a sign bit.
 The following example uses 12 bits instead of 32 for simplicity.

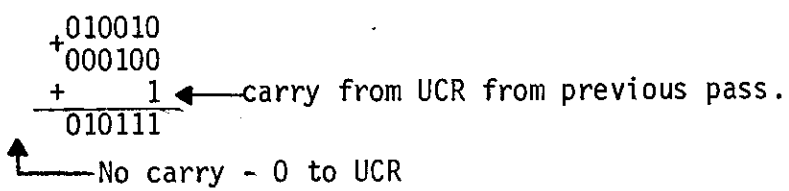
MULTIPLE-PRECISION ADDITION (BAD & BADX)



1st pass through ALU (BAD)

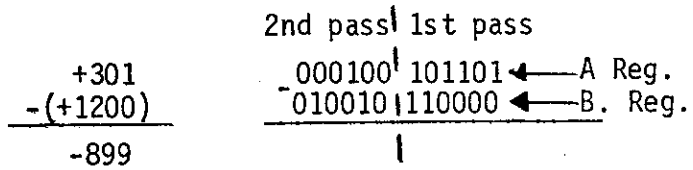


2nd pass through ALU (BADX)

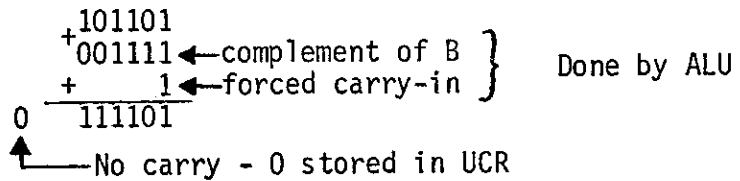


Resultant answer: 010111011101 (1501)

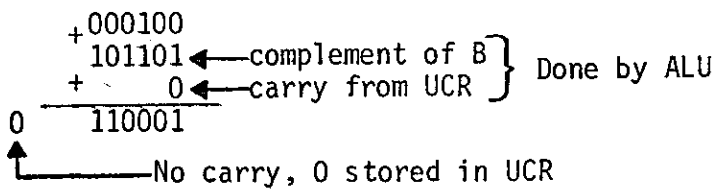
MULTIPLE PRECISION SUBTRACT (BSB,BSBX)



1st pass through ALU (BSB)



2nd pass through ALU



RESULTANT ANSWER: 110001111101 (2's complement of -899)

DECIMAL ARITHMETIC OPS

Decimal ops are performed on a right-justified byte in the A and B register. The data is in packed decimal form.

In the zoned decimal format, a byte is required to hold each digit, 0-9. The zone portion of each byte is unused (except for the sign in the right-most byte). These zoned decimal fields can be packed into a fewer number of bytes if the zones are removed; thus a packed or unzoned decimal format. All numeric fields must have a sign, and in the zoned decimal format the first four bits (0-3) of the right-most byte are used to hold a sign. A hexadecimal C is a plus sign; and a hexadecimal D is a minus sign. In the packed format, the 4-bit sign is moved to the last four bits of the rightmost byte.

The following illustration shows the number 18,634 in a zoned format and a packed (unzoned) format. Note the difference in placement of the sign. For easier interpretation, the actual bit patterns are not shown.

ZONED

F	1	F	8	F	6	F	3	+	4
---	---	---	---	---	---	---	---	---	---

Zone Digit Zone Digit Zone Digit Zone Digit Sign Digit

PACKED

1	8	6	3	4	+
---	---	---	---	---	---

Digit Digit Digit Digit Digit Sign

The decimal digits are represented by 4-bit BCD (8421) code, therefore each pass through the ALU adds two decimal digits to two decimal digits and a possible carry from the UCR. A carry from the high-order digit is stored in the UCR.

Subtraction of decimal representations are performed using 10's complement addition. This method of subtraction by addition is somewhat analogous to using 2's complement addition when performing binary subtraction. When using 2's complement notation, binary numbers are left in 2's complement form in memory. When decimal data is processed, however, it is stored in true form. When subtracting one number from another number, by adding the 10's complement of one to the other, the addition may or may not have produced a carry. The carry, if it occurs, is dropped and the difference is in true form. If no carry occurs, the difference is in 10's complement form and must be corrected to place it in true form. No carry indicates that a larger number was subtracted from a smaller number, and thus a negative difference has resulted. To put a BCD number in 10's complement form, the 9's complement is first derived by subtracting each digit from 9 and adding 1 to the LSB. In a similar fashion, taking the 10's complement of a number in 10's complement form, places it back in true form. Where reference to BCD is made, 8421 code is implied.

DAD,DSB

DAD and DSB are ops intended for use in 8 bit BCD addition (DAD) or subtraction (DSB). Specifically, these ops are to be used for the first pass, of a series, through the ALU. That is, when the two least significant decimal digits of a number are added. Carries or borrows from the condition register are ignored during these ops, however, if a carry or borrow is produced during the execution of the op it is stored in the UCR.

DADX, DSBX

DADX and DSBX are ops used in 8 bit BCD addition (DADX) or subtraction (DSBX). These ops are used after the first pass through the ALU, and include a carry or borrow from the UCR that the first pass may have produced. If a carry or borrow is produced during the execution of these ops, it is stored in the UCR.

The following are examples of the decimal Add ops:

(It should be noted that these are BCD additions, not Binary)

$$\begin{array}{r}
 \text{2nd pass} \quad | \quad \text{1st pass} \\
 \hline
 \begin{array}{r}
 0000 \ 0000 \\
 0001 \ 0010 \\
 \hline
 \end{array}
 \quad | \quad
 \begin{array}{r}
 0101 \ 0110 \\
 0011 \ 0100 \\
 \hline
 \end{array}
 \end{array}$$

1st pass through ALU: (DAD)

$$\begin{array}{r}
 56 \\
 +34 \\
 \hline
 90
 \end{array}
 \quad
 \begin{array}{r}
 0 \leftarrow \text{Carry inhibited by ALU} \\
 +01010110 \\
 +00110100 \\
 \hline
 10010000
 \end{array}
 \quad
 \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{ALU performs BCD addition}$$

0
 0 \leftarrow No carry, 0 stored in UCR

2nd pass through ALU (DADX)

$$\begin{array}{r}
 00 \\
 +12 \\
 \hline
 12
 \end{array}
 \quad
 \begin{array}{r}
 0 \leftarrow \text{Carry from UCR resulting} \\
 \quad \quad \quad \text{from previous pass} \\
 +00000000 \\
 +00010010 \\
 \hline
 00010010
 \end{array}$$

0
 0 \leftarrow No carry, 0 stored in UCR

Resulting Answer: 0001 0010 1001 0000 (1290)

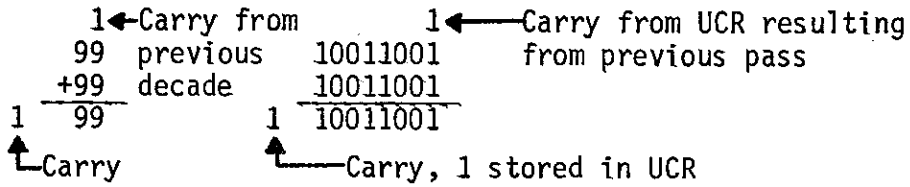
$$\begin{array}{r}
 \text{2nd pass} \quad | \quad \text{1st pass} \\
 \hline
 \begin{array}{r}
 1001 \ 1001 \\
 1001 \ 1001 \\
 \hline
 \end{array}
 \quad | \quad
 \begin{array}{r}
 1001 \ 1001 \\
 1001 \ 1001 \\
 \hline
 \end{array}
 \end{array}$$

1st pass through ALU (DAD)

$$\begin{array}{r}
 99 \\
 +99 \\
 \hline
 98
 \end{array}
 \quad
 \begin{array}{r}
 0 \leftarrow \text{Carry from UCR inhibited} \\
 +10011001 \\
 +10011001 \\
 \hline
 10011000
 \end{array}
 \quad
 \left. \begin{array}{l} \\ \\ \end{array} \right\} \text{BCD add done by ALU}$$

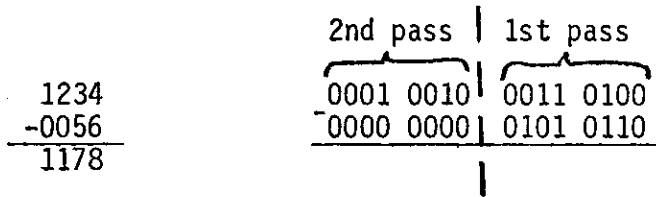
1 \leftarrow Carry to next decade
 1 \leftarrow Carry, 1 stored in UCR

2nd pass through ALU (DADX)

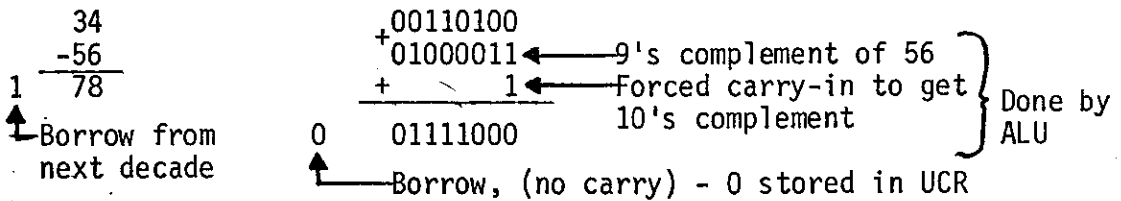


Resulting answer: 1 1001 1001 1001 1000 19998

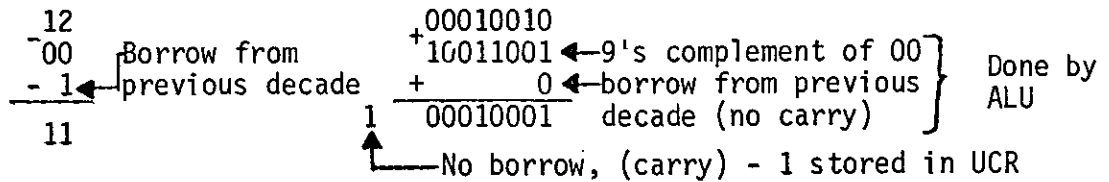
The following are examples of the decimal subtract ops:



1st pass through ALU (DSB)



2nd pass through ALU (DSBX)



Resultant answer: 1 0001 0001 0111 1000 1178

Since there was a carry (no borrow) in this answer, it is to be ignored, and the answer considered to be in true BCD form.

2nd pass	1st pass
0000 0000	0101 0110
-0001 0010	0011 0100
<hr style="border: none; border-top: 1px solid black;"/>	<hr style="border: none; border-top: 1px solid black;"/>

b	0056
	-1234
	<hr style="border: none; border-top: 1px solid black;"/>
	8822

↑ borrow

1st pass through ALU (DSB)

56	+ 01010110	
-34	+ 01100101	← 9's complement of 34
<hr style="border: none; border-top: 1px solid black;"/>	+ 1	← Forced carry-in to get 10's complement
22	<hr style="border: none; border-top: 1px solid black;"/>	00100010

↑ No borrow (carry), 1 stored in UCR

Done by ALU

2nd pass through ALU (DSBX)

b	00
	-12=
	<hr style="border: none; border-top: 1px solid black;"/>
	88

↑ borrow

0	+ 00000000	
<hr style="border: none; border-top: 1px solid black;"/>	+ 10000111	← 9's complement of 12
0	+ 1	← Carry (no borrow) from previous decade
<hr style="border: none; border-top: 1px solid black;"/>	10001000	

↑ Borrow (no carry) 0 stored in UCR

Resultant answer: 0 1000 1000 0010 0010

Since there was a borrow (no carry) in this answer, the answer is negative and is 10's complement form. It must be corrected (recomplemented). This is accomplished by taking its 10's complement, equivalent to subtracting the answer from all zeros with an implied high order borrow available.

Correction (Recomplementing)

b	0000	-8822	0000 0000	0000 0000
	-8822		1000 1000	0010 0010
	-1178			

1st pass through ALU (DSB)

b	00	-22	00000000	01110111	← 9's complement of 22	} Done by ALU
	-22		1	0111 1000	← Forced carry-in to get 10's complement	
	78		1			
1	↑ Borrow		0	↑ Borrow (no carry), 0 stored in UCR		

2nd pass through ALU. (DSBX)

b	00	-88	+00000000	+00010001	← 9's complement of 88
	-88		+ 0	00010001	← Borrow (no carry) from previous carry
	-1		00010001		
	-11				

Resultant recomplemented answer: 0001 0001 0111 1000

It should also be remembered that this answer is negative.

Logic Functions

The logical ops include compare, right and left shifts, inclusive OR, exclusive OR, AND, and invert.

CMP

The execution of an ALU instruction with a CMP op results in a logical and an arithmetic comparison of A to B. The results of this comparison are stored in the condition register. The DEST field of the instructions is not used during this op. Bits 1-3 of the UCR reflect the results for Arithmetic compare (includes sign) of A and B. Bits 5-7 of the UCR indicate the results of a logical (magnitude of 16 bits) compare of A and B. UCR bits 0 and 4 are reset during the execution of this op.

SLO, SRO

The shift ops define 1-bit linked, end-off shifts. SLO is a 1-bit serial left shift through the 32 bits of A and B. The resulting contents of the A register are gated onto the bus. SRO is a 1-bit serial right shift through the 32 bits of A and B. The resulting contents of the B register are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

IOR

IOR is the inclusive OR (logical summation) of A and B. The results of the OR are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

XOR

XOR is the exclusive OR (modulo - 2 sum) of A and B. The results of XOR are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

AND

AND is a logical AND (logical product) of A and B. The results of AND are gated onto the bus. Bits 0-7 of the UCR are not changed during the execution of this op.

INV

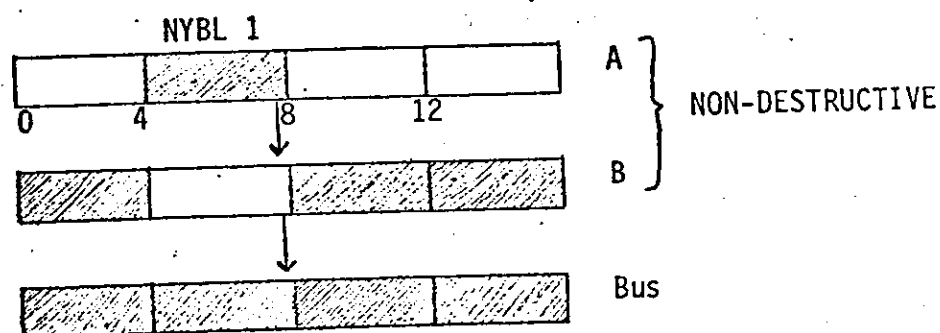
INV, performs a logical complementation. This op, inverts the state of each of the 16 bits of A, onto the bus. The contents of A remain undisturbed. Bits 0-7 of the UCR do not change during the execution of this op.

WORD, BYTE, and NYBL Manipulations:BCB:

Gates onto the bus the combination of the right byte of A and the left byte of B. UCR bits 0-7 are not changed.

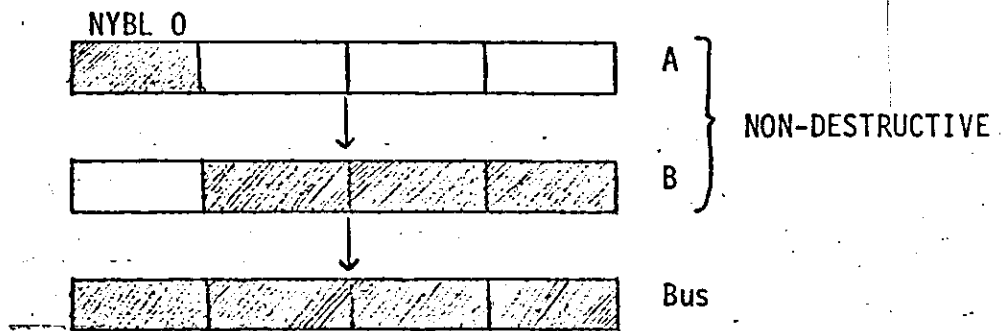
INI:

Gates onto the bus the insertion of NYBL 1 of A into B. UCR bits 0-7 are not changed.



INZ:

Gates onto the bus the insertion of NYBL 0 of A into B. UCR bit 0-7 are not changed.

DBT

Bit test DBT is an op in which any one of the 16 bits in the A register may be tested. The location of the bit to be tested is encoded in the low-order 4 bits of the DEST field of the micro-instruction. The state of the bit tested is placed in UCR bit zero. Bits 1-7 of the UCR are reset. No data is gated on to the bus during the execution of this op.

BBT

Bit test BBT is an op identical to DBT in with the exception of the location of the address of the bit to be tested. The low-order 4 bits of the B register are encoded to hold this address. As with DBT, the results of the test are stored in Bit 0 of the UCR. Bits 1-7 of the UCR are reset. No data is gated onto the bus during the execution of this op.

AOB, BOB

The contents of the A register (AOB) or B register (BOB) are gated onto the bus. UCR bits 0-7 are not changed.

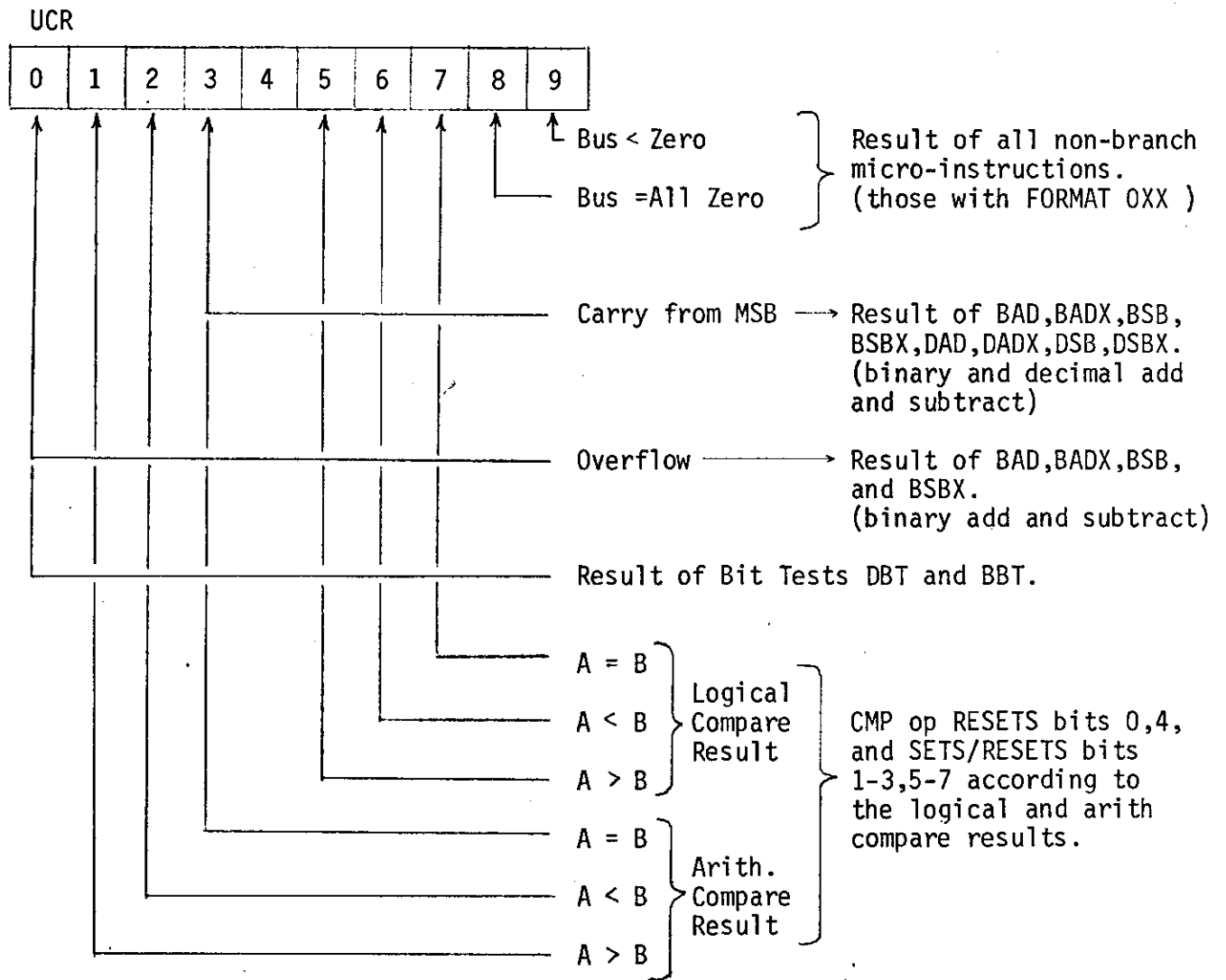
TABLE 3ALU-OP CODES

<u>CODE</u>	<u>MNEM</u>	<u>OPERATION</u>
00	SLO	Shift B to A, left open, 1 bit, A onto bus.
01	DAD	BCD add, right half A to right half B, no CI, CO.
02	BAD X	Add A to B, CI, CO.
03	DAD X	BCD add, right half A to right half B, CI, CO.
04	CMP	Compare.
05	DSB	BCD sub, right half B from right half A, no BI, BO.
06	BSB	Subtract B from A, no BI, BO.
07	BSB X	Subtract B from A, BI, BO.
08	ADD	Add A to B, no change to UCR.
09	BAD	Add A to B, no CI, CO.
0A		
0B		
0C		
0D	DSB X	BCD sub, right half B from right half A, BI, BO.
0E	SUB	Subtract B from A, no change to UCR.
0F	BCB	Combine right byte A, left B, onto bus.
10	INV	Invert A (1's complement), onto bus.
11	SRO	Shift A to B, right open, 1 bit, B onto bus.
12		
13		
14		
15	IN1	Insert 'NYBL1' of A into B onto bus.
16	INZ	Insert 'NYBL 0' of A into B onto bus.
17		
18		
19	XOR	Exclusive OR (A, B)
1A	BOB	Gate B onto bus.
1B	IOR	Inclusive OR (A, B)
1C	DBT	Bit test. (address in DEST field)
1D	BBT	Bit test. (address in B).
1E	AND	AND (A, B)
1F	AOB	Gate A onto bus.

NOTE: CI = carry-in, CO = carry-out, BI = borrow-in, BO = borrow-out.

FIGURE 2

CONDITION REGISTER SETTING



3.3 PORT OPERATIONS

The 3-bit PORT field contained in certain types of micro-instructions is used to encode the initiation and control of CPU-memory data transfers. Table 4 lists the PORT operation codes and mnemonics.

Those PORT ops which initiate memory requests all use the MAR to hold the memory address to be read/written. The MDR holds the data for writes and receives the data during reads. During FULLWORD ops, the high-order 15 bits of MAR select the word address. During byte ops, all 16 bits of MAR are used (see below).

It should be noted that the CPU must compete for memory accesses with other devices of higher priority on the Direct Memory Access bus. Thus, a PORT memory initiation may not result in completion within a fixed number of micro-cycles. The actual completion delay time is a function of the interference on the DMA. Interlocking in the micro-code is accomplished by the use of the HDM port operation. This 'HOLD-ON-MEMORY' op is used following a memory operation to hold up micro-code execution until the memory operation is completed. Refer to section 5.0 for memory operation micro-programming techniques.

FMR, FMW are the port ops for READ and WRITE of a full 16-bit memory word. BMW is the op for a byte memory write with the left/right byte written depending on the low-order bit of MAR (0 = left, 1 = right). FPW and BPW are the same as FMW, BMW except that the protect logic is enabled, and upper(U) and lower(L) bounds are placed on MAR(0-7).

TSR (test read) is a special PORT op which performs the same function as FMR, but additionally raises a control line to the memory system which causes the CPU to have top priority during the following memory cycle. This enables the CPU to perform a READ followed immediately (locking out any interference) by a WRITE, and thus perform a "TEST and SET" programming function.

TABLE 4PORT OPERATIONS

<u>CODE</u>	<u>MNEM</u>	<u>DESCRIPTION</u>
000	NOP	No operation
001	FMR	Full memory word read
010	BMW	Byte memory write. If $MAR_{15} = 0$, left byte. If $MAR_{15} = 1$, right byte.
011	FMW	Full memory word write.
100	HDM	Hold on memory.
101	TSR	Test read.
110	BPW	Protected BMW.
111	FPW	Protected FMW.

TABLE 5BRANCH CONDITION CODES

<u>CODE</u>	<u>MNEMONICS</u>	
	<u>R = 0</u>	<u>R = 1</u>
0	TRU	NTRU
1	I1Z	NI1Z
2	I2Z	NI2Z
3	R1Z	NR1Z
4	R2Z	NR2Z
5	ZOB	NZOB
6	NOB	NNOB
7	ERR	NERR
8	UCZ	NUCZ
9	UC1	NUC1
A	UC2	NUC2
B	UC3	NUC3
C	INT	NINT
D	---	---
E	---	---
F	---	---

TABLE 6

SUMMARY OF MICRO-INSTRUCTION FIELD CODES

<u>CODE</u>	<u>SOURCE</u>	<u>DEST</u>	<u>OP</u>	<u>PORT</u>	<u>COND</u>	<u>COND</u>
0	REG0	REG0	SLO	NOP	TRU	NTRU
1	REG1	REG1	DAD	FMR	I1Z	NI1Z
2	REG2	REG2	BADX	BMW	I2Z	NI2Z
3	REG3	REG3	DADX	FMW	R1Z	NR1Z
4	REG4	REG4	CMP	HDM	R2Z	NR2Z
5	REG5	REG5	DSB	TSR	ZOB	NZOB
6	REG6	REG6	BSB	BPW	NOB	NNOB
7	REG7	REG7	BSBX	FPW	ERR	NERR
8	REG8	REG8	ADD		UCZ	NUCZ
9	REG9	REG9	BAD		UC1	NUC1
A	REGA	REGA	—		UC2	NUC2
B	REGB	REGB	—		UC3	NUC3
C	REGC	REGC	—		INT	NINT
D	REGD	REGD	DSBX		—	—
E	REGE	REGE	SUB		—	—
F	REGF	REGF	BCB		—	—
10	X	X	INV			
11	PIR	PIR	SRO			
12	MDR	MDR	—			
13	SMDR	SMDR	—			
14	UPNT	PROT	—			
15	R1I	R1I	IN1			
16	R2I	R2I	INZ			
17	SWS	A	—			
18	PIRD	B	—			
19	P	UIR	XOR			
1A	MAR	MAR	BOB			
1B	CST	AMAR	IOR			
1C	UCR	UCR	DBT			
1D	—	—	BBT			
1E	—	—	AND			
1F	ZRO	NOP	AOB			

3.4 BRANCH CONDITION CODES

Table 5 lists the condition codes and mnemonics for the BRANCH CONDITIONS encoded in and tested by the SCB and ACB conditional-branch micro-instructions. Mnemonics are listed for both R=0 and R=1 for each condition code. If R=1 (mnem. prefix N), the inverse condition is tested.

I1Z, I2Z, R1Z, R2Z conditions are true if the corresponding field in the PIR=0.

The conditions UCZ, UC1, ---, UC3 are true if the corresponding bit in the UCR=1.

ZOB condition is true if the previous non-branch micro-instruction bused ZERO to a DEST. NOB condition is true if the previous non-branch micro-instruction bused negative data to a DEST.

The INT condition is true if the output of the PRIORITY/INTERRUPT SYSTEM is not equal to the current value of the P register.

The TRU condition always results in a branch.

The ERR condition branches if any error has been detected. For now, only a PROTECT error is detected.

4.0 MICRO-ASSEMBLY LANGUAGE

The 7100 MICRO-ASSEMBLY LANGUAGE is used to write symbolic micro-programs for the MEMOREX 7100 processor. Programs written in this language are processed by the 7100 Micro-Program Assembler to produce listing and object files.

The following sections describe the structure of the language. Many of the mnemonics and symbols used to actually construct symbolic micro-instructions are not listed in these sections. Those mnemonics which referenced actual 7100 CPU registers and operations are listed in the earlier chapters of this manual.

The input file for the Micro-Program Assembler consists of 80 character records, usually punched cards. There are four (4) types of input cards: (1) comment card, (2) assembly control statement card, (3) define constant card, and (4) micro-instruction card.

4.1 COMMENT CARDS

Comment cards have an * in column 1. These cards are listed on the output listing but have no effect on the object code produced. None of the restrictions on the remaining card types apply to comment cards. Columns 2-80 may contain any alphanumeric or special characters.

4.2 GENERAL CARD FORMAT

The following specifications apply to all card types except comment cards.

Blanks are in general ignored, unless some explicit rule is mentioned for blanks for a specific statement type. The scan and processing of a card will stop with the last card column, or with the first semicolon. Any characters on the card past the semicolon will not be processed, but will appear on the output listing. Thus comments may be inserted on any card.

The general card format is:

Columns 1-5:	Label Field
Column 6:	Blank
Columns 7-80:	Dependant on card type.

4.3 LABELS

A label consists of 1 to 5 alphanumeric characters in the label field of a card.

4.4 VALUES

A 'value' is one or more 'value elements' separated by the delimiters + or -. A value element is any of the following:

- 1) Label
- 2) X 'xxxx' a hex constant
- 3) D 'dddd' a decimal constant
- 4) * the present location counter

4.5 ASSEMBLY CONTROL STATEMENTS

There are three assembly control statement ops: (1) ORG, (2) EQU, and (3) END. For these statements card columns 7-9 must contain the op, and columns 10-80 may contain a value field.

The ORG statement controls the setting of the program counter. A label may be used. The program counter is set to the 'value' of the value field. Any labels in the value field must be pre-defined (appear before the ORG statement). A label on the ORG card will be assigned the value of the value field.

The EQU statement is used to define a symbol. A label must be used or the EQU statement will be ignored. The EQU card causes the assignment of the 'value' of its value field to its label. Any labels appearing in the value field must be pre-defined.

~~The~~ END card signals the end of the input deck. The label and value fields are ignored.

4.6 DEFINE CONSTANT STATEMENTS

In addition to the UROM memory which holds micro-instructions, the 7100 CPU has a CROM memory which holds up to 256 16-bit constants which may be referred to by the GCD type micro-instructions.

The output of the assembler, in addition to micro-instruction object code, must therefore contain an object file to initialize the CROM.

This file will be constructed during the assembly process by adding a new constant on each use of a GCD type micro-instruction with a previously not encountered CROM literal value field, and on each use of a DEFINE CONSTANT (DC) statement which specifies a previously not encountered CROM value field (if previously encountered, then an equate of CROM labels is established).

The DEFINE CONSTANT statement contains the DC op in columns 7-8 and a value field in columns 10-80. The DC statement may have a label. The label field symbol, if present, causes the assignment of the assembled CROM address and DC statement value to the label field symbol.

4.7 MICRO-INSTRUCTION STATEMENTS

If the assembler finds that a card is not a comment card, assembly control card, or define constant card, it assumes that it is a micro-instruction statement card.

Each micro-instruction statement results in the generation of one micro-instruction in the output object file. The micro-instruction statement may have a label in columns 1-5, while columns 7-80 are a free form field area into which the micro-instruction is symbolically encoded.

All micro-instruction statements have the same basic free form field structure. The FORMAT type of the instruction is implied by the specific 7100 MNEMONIC symbols used by the instruction. The basic symbolic micro-instruction structure is:

X = Y, modifier;

where X = primitive

or X = primitive.modifier

Following are simple examples of the encoding of micro-instructions of each format type, which will serve to approximately define the language at this time. At a later date when the 7100 architecture is more fixed, a more formal description will be constructed.

The following examples should be sufficient to provide guidelines for the construction of sample emulation programs, etc., and to indicate the level of logic required in the micro-program assembler itself.

<u>FORMAT</u>	<u>EXAMPLES</u>	<u>FUNCTION ENCODED</u>
<u>ALU:</u>	MAR = ADD; 14 = SUB, FMR;	$A + B \rightarrow \text{MAR}$ $A - B \rightarrow \text{REGF, also FMR}$
<u>GSD:</u>	PIR = MDR; 15 = MDR, XRA;	$\text{MDR} \rightarrow \text{PIR}$ $\text{MDR} \rightarrow \text{REGF, with X rather than P used for col. addr.}$
<u>GCD:</u>	MAR = C (X'40'); MAR = C (LABEL);	The constant X'40' from the CROM \rightarrow MAR. The constant value (assgnd to LABEL) from the CROM \rightarrow MAR.
<u>GID:</u>	A = I (X'F1'); MDR = I (CHAR);	$\text{X'F1'} \rightarrow \text{A}$ $\text{VALUE (CHAR)} \rightarrow \text{MDR}$
<u>SCB:</u>	BRA.UCZ = 15, FMW;	Branch if $\text{UCR}_0 = 1$ to UROM ADDR in REGF. Also FMW.
<u>ACB:</u>	BRA.IIZ = *D'3';	Branch if $\text{II} = 0$ to current location plus 3_{10}
<u>UCB:</u>	BRA = TEST; BSR = TEST;	Branch to location given by value of label 'TEST' Same, but store UPNT return into assigned register in RAM.

NOTE: If PORT modifier is used alone, then prefix with a comma. For example, the following symbolic instruction assembles into a GSD format with HDM Port field and NOP DEST field:

, HDM;

5.0 MICRO-PROGRAMMING TECHNIQUES

In order to properly and efficiently micro-program the 7100, certain basic techniques, restrictions, and timing information must be understood. This chapter provides such information.

5.1 SOURCE-DESTINATION RESTRICTIONS

A number of restrictions exist on the use of sources and destinations for ALU, Gating, and Branch type operations.

An important restriction is that any one micro-instruction may make only one access (either SOURCE or DEST) to the RAM register file.

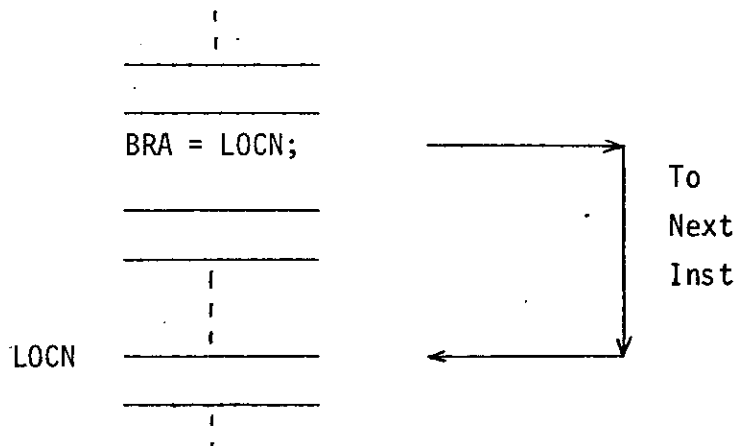
A large class of restrictions is implied by the data flow (Figure 1) and table of source and destination codes (Table 2): a number of the CPU registers may serve as only a SOURCE or DEST but not both. For example UPNT may only be a SOURCE (see Table 2) even though Figure 1 shows an input from the bus into UPNT (that path is used only by the SCB type instructions which have UPNT as an implied DEST).

Caution must be used in gatings from SOURCE's to DEST's when dissimilar width registers are involved. Justifications and unfilled bit positions must be verified (some of these have not yet been specified in this manual).

5.2 SCB BRANCH TIMING

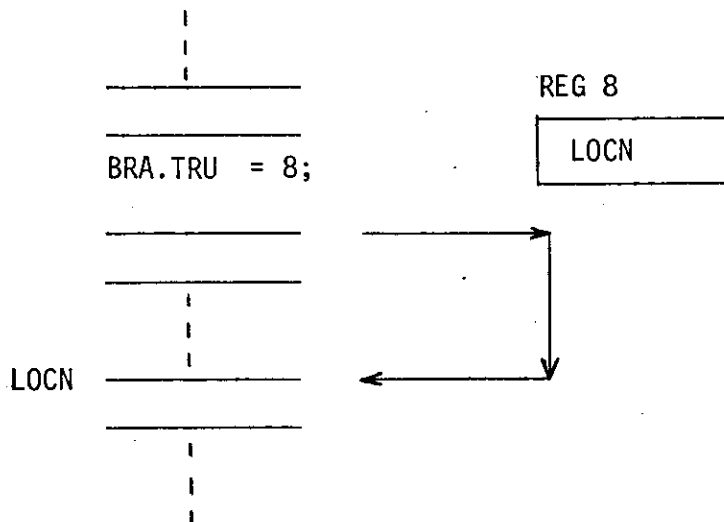
The ACB and UCB format type branch instructions function completely within one micro-instruction cycle. In other words, as expected we find the next instruction (if the branch is taken) is fetched from the specified branch address location in UROM:

EXAMPLE:



However, due to the time required to obtain the branch address from a source, the SCB format type branch actually functions one cycle after the SCB itself. In other words, the instruction following the SCB is always executed. The next instruction after that is at the branch address obtained from the source if the branch is taken:

EXAMPLE:



The SCB op enables branching across the entire 8K UROM span because a 13-bit address is loaded into UPNT from the bus to effect the branch. Note that an SCB branch followed by an ACB (both testing the same condition) enables a conditional branch to an 8-bit address specified by the ACB on the 'page' specified by the SCB.

5.3 MEMORY PORT OPERATION TIMING

Various memory port ops are described in Section 3.3, where it was noted that, due to possible interference on the DMA bus, a memory op initiated by a CPU port op does not necessarily complete within a predetermined time.

Thus the micro-code must interlock memory access initiation with "Hold-on-Memory" port ops so that completion of the access is insured before using the result of a READ or overwriting MAR or MDR for a write.

A "HOLD-ON-MEMORY" (HDM) port op in a micro-instruction causes subsequent micro-instruction execution to be inhibited if a CPU memory request is up and not yet completed. The micro-instruction containing the HDM is always completely executed before the hold takes effect.

The 7100 micro-instruction cycle is 400 ns. The 7100 main memory cycle is 1.2 μ s. Thus there are three (3) micro-cycles per memory cycle.

The concept of the Hold-on-Memory (HDM) interlocking micro-instruction, and the timings which result if interference is present is displayed in Figure 3, EXAMPLE 1.

In this example, the CPU requests a memory read at the same micro-cycle as a higher priority device on the DMA. Thus, when the subsequent CPU HDM micro-instruction is executed, the CPU enters a HOLD state and micro-instruction execution is disabled while the memory request remains up.

In memory cycle 2 of example 1, another higher priority (than CPU) request is serviced and the CPU remains in HOLD.

Finally, in memory cycle 3, the CPU memory read is serviced, and the CPU leaves HOLD and its request line falls.

This example displays the ratio of three (3) micro-cycles per memory cycle.

Note, however, that the memory is not restricted to accesses which are synchronous to 1.2 μ s time boundaries.

The memory may be accessed on any micro-cycle in which it is not busy. This feature is termed "zero-latency" accessing. This feature will be described in more detail later in this section.

5.3.1 WRITE OPS (see FIG.3, EX.2)

A main memory WRITE operation is requested by the CPU when any of the PORT codes FMW, BMW, FPW, FBW are decoded and executed by the CPU.

The MAR and MDR must be loaded with valid data before (or by) the micro-instruction which initiates the write.

Following the write micro-instruction and prior to and including the interlocking HDM, neither MAR nor MDR should be changed.

Following the HDM micro-instruction, the write has completed and MAR and MDR may be changed.

Example: _____, FMW; initiates WRITE
 _____; any OP
 _____, HDM; interlocks till WRITE complete
 _____; MAR, MDR may now be changed

5.3.2 READ OPS (see FIG.3, EX.2)

A main memory READ operation is requested by the CPU when the port codes FMR or TSR are decoded and executed by the CPU.

The MAR must be loaded with valid data before (or by) the FMR or TSR micro-instruction. Both MAR and MDR must be held fixed prior to and including the interlocking HDM micro-instruction.

The new MDR value is available in the instruction following the HDM.

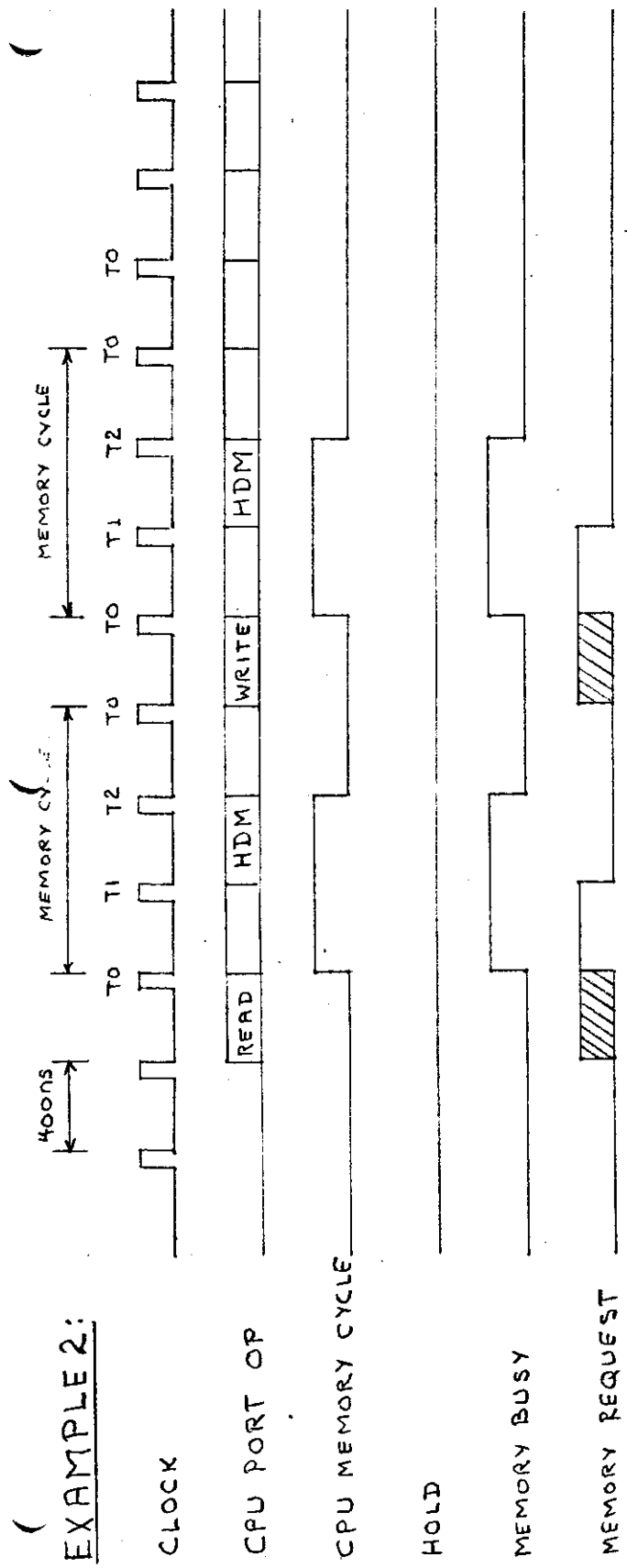
Example: _____, FMR; initiates READ
 _____; any OP
 _____, HDM; interlocks till READ complete
 Y = MDR; new MDR value available for use

5.4 DECIMAL ALU OP TIMING

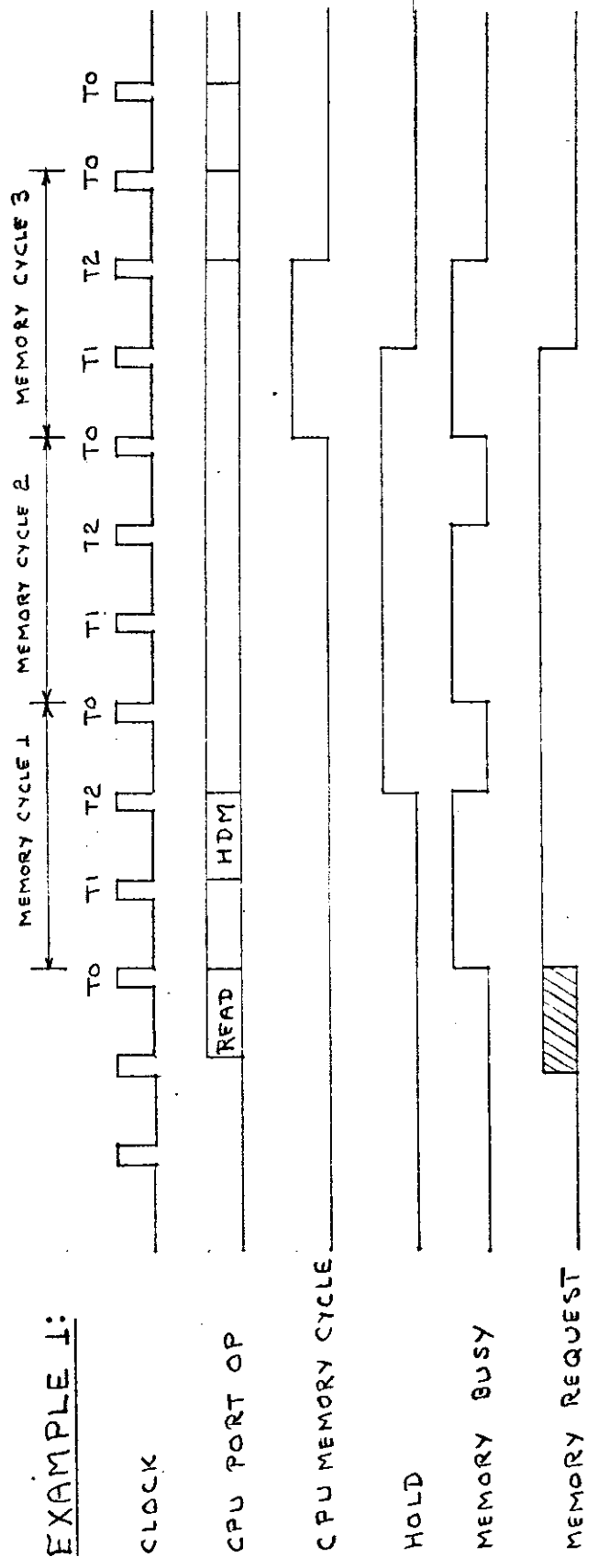
The decimal ALU operations DAD,DSB,DADX,DSBX, require two micro-cycles for their execution (see section 3.2).

The preferred technique for micro-coding these operations to insure two micro-cycle execution timings is to consecutively execute two identical micro-instructions for each decimal ALU op to be encoded. The first micro-instruction of the identical pair will initiate the execution of the op, and the second will place valid data onto the bus to the selected destination.

EXAMPLE 2:

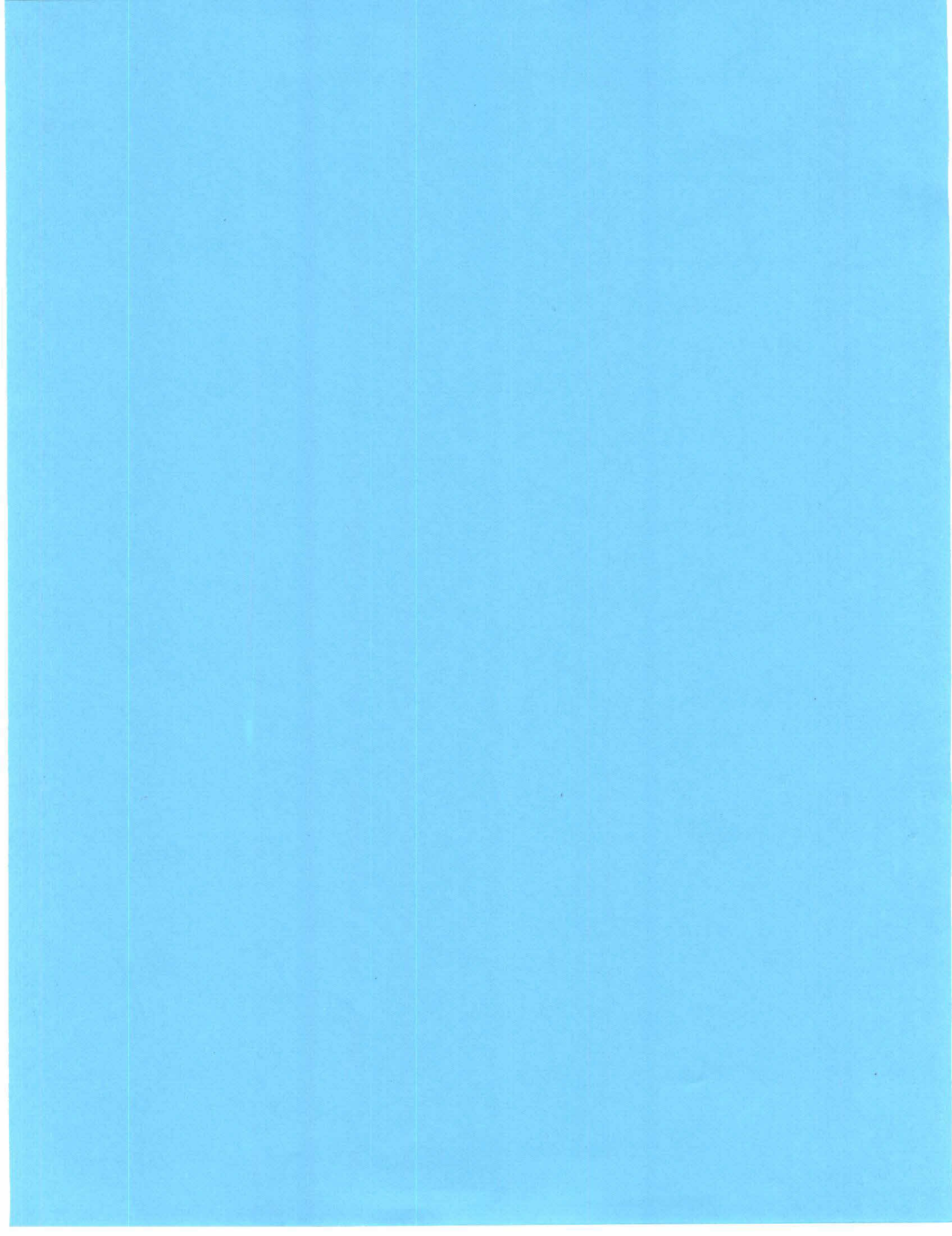


EXAMPLE 1:



2.02 year
G-15-72

FIGURE 3. CPU-MEMORY PORT TIMING



MEMOREX 7100

CPU LOGIC DESIGN

G. YEE

R. STALLMAN

7/19/72

MEMOREX CONFIDENTIAL

CONTENTS

- 1.0 INTRODUCTION
- 2.0 MICRO-INSTRUCTION DECODE
- 3.0 MICRO-ROM ADDRESSING
- 4.0 MICRO-PROGRAM STORE
- 5.0 PROGRAM INSTRUCTION REGISTER
- 6.0 ARITHMETIC LOGIC UNIT
- 7.0 CONDITION REGISTER
- 8.0 REGISTER FILE AND I/O CONTROL
- 9.0 MEMORY INTERFACE

1.0

Introduction

This manual describes the logic design of the Memorex 7100 CPU. It is intended that this manual will provide all the basic information necessary to understand the CPU block diagram and the second-level logic diagrams of the CPU organization.

The implementation discussed is in TTL, and employs MSI elements and ROMS, where applicable, to reduce parts count. The CPU requires a clock frequency of 400 nsec. and a pulse width of approximately 50 nsec. Because of this speed, bipolar ROMS and RAMS were chosen. The design chosen eliminates the need to employ Schottky or high speed elements.

The block diagram has been partitioned for convenient board sizes based on the TTL implementation.

The design, as implemented, is of the architecture detailed in the architecture/microprogram manual.

2.0 Micro-Instruction Decode (Ref. 7100 CPU Block Diagram)

The micro-instructions are loaded into the u-instruction register and held there for one clock cycle. Normally, u-instructions are loaded into the UIR from the UROM, however the UBUS may be used to load the UIR during halt. The UROM output is or'ed with the UBUS when the UIR is specified as a destination by the micro-program. Micro-instruction modification is accomplished with this capability.

The contents of the UIR are preserved during a halt unless the operator loads a new u-instruction from the control panel.

2.1 Micro-op Decode

The micro-op field is 3 bits wide which allows a maximum of 8 micro-instruction formats that may be specified. The micro-op field is in bit positions 0-2 in all 8 micro-instruction formats. Refer to Table 1 of the CPU Architecture/Micro-programming Manual. Each micro-op is decoded and used to enable the field decoding that applies to the particular instruction in the UIR. An example would be one in which the source field is found in micro-ops 1 and 5 only. Thus the source decode is enabled only for these two micro-op formats. When the system is in halt the micro-op decode is disabled.

2.2 Register Source and Destination Decode (Ref Table 2, CPU Arch/Micro-programming Manual)

The micro-instruction format contains a 5 bit field which is the address of the source for information to be placed on the UBUS and a 5 bit field for the destination of information from the UBUS. There are a maximum of 32 possible sources and 32 destinations. The 16 least significant source and destination decodes, address the register file, and the most significant 16 are either discrete registers, not assigned, or special operation codes. A source

or destination of R1I/R2I implies that the source/destination address can be found in the R1I or R2I field of the PIR. The R1I or R2I field can specify only one of the first 8 registers in the register file.

2.3 Memory Port Decode: (Ref. Table 4 of the Architecture/microprogramming manual)

The memory port decode is a three bit field in the ALU, GSD and SCB micro-instructions. There are eight possible commands that can be decoded. All port commands are memory-related. The decoded port field has the capability of providing 8 commands. BMW, TSR, FMW, BPW, FMR, and FPW are decoded in the memory interface control logic. FMR, TSR, and HDM are decoded elsewhere. Since memory operations take more than one cycle the port field is retained in a latch by the CPU memory interface control logic until the memory operation is complete.

2.4 Branch Condition Decoding: (Ref. Table 5 of the Architecture/Microprogramming manual).

The condition field is a 4-bit field which selects a maximum of 16 possible status conditions for a micro-instruction branch. There is also a reverse sense bit that is used in conjunction with the 4 bit condition field for micro-instruction branching on the not-true condition. The conditions are described in the architecture/microprogramming manual. The decoding of the condition field is enabled by UOP 5 and UOP 6, the conditional branch micro-instruction formats.

2.5 ALU Control Decode

The ALU up field of 5 bits allows definition of 32 types of arithmetic/logic functions. The ALU control decode uses these 5 bits along with UOP 0 to generate all A & B register and ALU controls.

2.6 CROM Address

The GCD instruction, UOP 2, has an 8-bit field which is an address for a table of 16-bit constants stored in a read only memory (CROM). The output of the CROM is always sourced onto the bus for this type of micro-instruction. The CROM may contain 256 16-bit constants.

3.0 Micro-ROM Addressing (13 bits)

The micro-instructions stored in the UROM are normally executed in consecutive order. The UROM address is incremented by one and the incremented address is stored in the UPNT register.

There are several ways to branch to a different UROM address. For the unconditional branch the 12 least significant bits of the UIR and the most significant bit of the UPNT are used to generate the UROM address. The 8 LSBs of the ACB micro-instruction are combined with the five MSBs of the UPNT to produce another type of branch address. A branch address can also be loaded into the UPNT from the bus as a result of a SCB type micro-instruction.

The UPNT register is 13-bits wide and has the capability of addressing 8k x 16 bit words of UROM. The UPNT register can be loaded by the control panel when halted or during normal operation from the incrementer.

4.0 Micro-program Store (UROM)

The micro-program store is 8K maximum by 16 bits of ROM. The UROM consists of bipolar ROM elements with a worst case access time of 60 nsec. The UROM is accessed every 400 nsec.

5.0 Program Instruction Register (PIR)

The PIR is a 16 bit register that is used to hold macro-program instructions. It is loaded from the UBUS when selected as a destination. The 8 MSBs are used to address a decode ROM (DROM)

which is 256 x 13 bits and contains branch addresses for the UROM. The 8 LSBs of the macro-instruction field are designated as R1 and R2 fields and are used as register file addresses when specified as a source or destination.

6.0 Arithmetic Logic Unit

6.1 General

Included in the arithmetic-logic unit are the A & B feeder registers; the ALU logic; the control for the ALU logic; the control for setting/resetting bits related to Arithmetic-logic unit functions in the micro-condition register; and logic required to perform word, byte, and NYBL manipulation.

All ALU ops are performed on 8 or 16-bit operands previously loaded into the A & B registers. Results of the ALU operations are gated to the destination specified in the destination field of the micro-instruction. Results of these operations also set/reset related bits in the micro-condition register. In some ALU instruction formats the destination field is unused.

Arithmetic operations performed by the ALU are either binary or decimal. These two general categories differ from each other substantially. Binary ops: require one micro-cycle for their execution; involve two 16-bit operands; and require binary numbers to be represented in 2's complement notation. Decimal ops: require two micro-cycles for their execution; involve 8 bit (two decimal digits) operands and outputs; and require digits to be represented in packed BCD (8421) format.

6.2 Control

When an ALU instruction is decoded, if it is of a format having a destination, the arithmetic-logic unit gates to the bus are enabled. All other micro-instructions disable these outputs to the bus.

The five least-significant bits of the ALU micro-instruction determine the operation to be performed (Ref. Table 3 of the Architecture/Microprogramming manual). It is the decode of these 5 bits that control all functions of the arithmetic-logic unit. (Ref. logic diagram - "ALU op Decode")

6.3 Detailed Functional Description

(Ref. Table 3 of the Architecture/Microprogramming manual)

6.3.1 A & B Registers: (Ref. logic diagram - "A & B Registers")

The A & B Registers are 2-16 bit registers serially linked. Each is composed of 2-8 bit MSI Registers serially linked. They perform; right shift, left shift, parallel load, or do nothing. Control for them is via two mode control lines. Broadside loading of the registers is allowed when either is selected as a destination during execution of a micro-instruction. ALU ops SRO and SLO (shift right or left one bit) are linked (32 bit) end-off shifts. The actual shifting of the data in the A & B registers will not be accomplished until the end of the micro-cycle. Since the shifted data must be present at the destination specified in the micro-instruction prior to the end of the cycle, the outputs of the registers must be scaled one position left or right with gates. The resultant output then appears at the destination shifted. Right shifts are scaled through discrete gates. Left shifts are scaled through the MSI ALU.

6.3.2 WORD, BYTE or NYBL Manipulation: (Ref. logic diagram - "A & B Registers")

ALU, ops IN1, INZ & BCB, insert nybles or bytes from the A register into corresponding fields in the word from the B register. AOB & BOB are ops which allow the A or B register contents to be gated to a destination without modification. These ops are performed with gates for IN1, INZ, BCB, and through the MSI ALU for AOB & BOB. INV gates the 1's complement of the A register onto the bus. This function is performed with the MSI ALU.

6.3.3 Bit Test: (Ref. logic diagram - "A & B Registers")

ALU ops BBT and DBT nondestructively test the state of any one of the 16 bits in the A register and stores the state of that bit in bit zero of the micro-condition register. This function is performed with a

1 out of 16 decoder. The address of the bit to be tested is encoded in the low-order 4 bits of the dest field of the micro-instruction for DBT and in the low-order 4 bits of the B register for BBT. This op has no data destination, therefore, when it is executed, destination decoding must be disabled.

6.3.4 Arithmetic Functions:

ADD, BAD, BADX, SUB, BSB, & BSBX are binary arithmetic ops and are performed on the two 16-bit operands with an MSI ALU. (4-74181, 1-74182). The MSI ALU is controlled with 5 mode control lines. Additional control circuitry is required to handle carries and overflows. Carries may be stored in the micro-condition register for use in multiple-precision additions or subtractions. Algorithms and logic for the control of carries and overflow are included on logic diagram - "UCR & UCR CONTROL". Two's complement notation is used in these operations.

DAD, DADX, DSB, & DSBX are decimal ops performed on a right-justified byte in the A and B register. The data is in packed decimal form. The decimal digits are represented by 4-bit BCD (8421) code, therefore each pass through the ALU adds two decimal digits to two decimal digits and a possible carry from the UCR. A carry from the high-order digit is stored in the UCR.

Subtraction of decimal representations are performed using 10's complement additions. This method of subtraction by addition is somewhat analogous to using 2's complement addition when performing binary subtraction. When using 2's complement notation, binary numbers are left in 2's complement form in memory. When decimal data is processed, however, it is stored in true form. When subtracting one number from another number, by adding the 10's complement of one to the other, the addition may or may not have produced a carry. The carry, if it occurs, is dropped and the difference is in true form. If no carry occurs, the difference is in 10's complement form and must be corrected to place it in true form. No carry also indicates that a larger number was subtracted from a smaller number, and thus a negative difference has resulted. To put a BCD number in 10's complement form, the 9's complement is first derived by subtracting each digit from 9 and adding 1 to the LSB.

In a similar fashion, taking the 10's complement of a number in 10's complement form, places it back in true form. When reference to BCD is made, 8421 code is implied.

DAD, DSB

DAD and DSB are ops intended for use in 8 bit BCD addition (DAD) or subtraction (DSB). Specifically, these ops are to be used for the first pass, of a series, through the ALU. That is, when the two least significant decimal digits of a number are added. Carries or borrows from the condition register are ignored during these ops, however, if a carry or borrow is produced during the execution of the op it is stored in the UCR.

DADX and DSBX are ops used in 8 bit BCD addition (DADX) or subtraction (DSBX). These ops are used after the first pass through the ALU, and include a carry or borrow from the UCR that the first pass may have produced. If a carry or borrow is produced during the execution of these ops, it is stored in the UCR.

The Decimal ops are passed through a separate ALU (in parallel with the MSI ALU) composed of 7483 ADDERS and the necessary control circuitry to place it in the Add or Subtract mode. The basic circuit consists of a BCD adder which corrects for illegal BCD counts (over decimal 9). A 9's complemeter at the input to the BCD adder is enabled during subtraction. With a forced carry into the low-order decade, a 10's complement addition is performed to effect a subtraction.

6.3.5 Remaining Logic Functions:

(Ref. Table 3 of the Architecture/Microprogramming manual).

XOR, IOR, and AND are performed in the MSI ALU. The only control circuitry for these functions is that involved in decoding the 5-bit ALU op field.

6.3.6

Compares:

CMP, the compare op simultaneously does both an arithmetic (signed) and a logical (unsigned) comparison of the A & B register contents.

The results of the comparison are placed in the micro-condition register (Ref. Fig. 2 of the Architecture/Microprogramming manual). Algorithms for the control of the micro-condition register are included on logic diagram - "UCR & UCR Control".

7.0

Condition Register (UCR)

7.1

General: (Refer to Fig. 2 of the Architecture/Microprogramming manual)

The condition register is a 10 bit register used for: storing the carries and overflows resulting during arithmetic ops, holding the results of compare and bit test instructions, and for holding the relative arithmetic (signed) magnitude of the results of all non-branch instructions ($BUS = 0$, $BUS < 0$). These 10 bits form a part of the condition set used by the branch instructions.

7.2

Controls for Clocking the Condition Register: (Ref. logic diagrams - "UCR & UCR Control", "ALU & Control", "UCR & Control")
Bits 0-7 of the UCR are clocked only during ALU instructions with ops requiring bits to be set/reset, or when a micro-instruction specifies the UCR as a destination. The remainder of the instructions and instruction formats do not affect this part of the condition register.

Bits 9 & 10 of the condition register are clocked with all non-branch micro-instructions. These 2 bits are not included as destinations specified in the DEST field of micro-instruction formats.

7.3

Set/Reset Controls for the Condition Register: (Ref. Table 2 of the Architecture/Microprogramming manual)

The algorithms and logic required to set bits 0-9 of the condition register are detailed in logic diagram - "UCR & UCR Control".

General (Ref. Logic diagram - "Reg. File & I/O Interface")

The register file contains 80 16-bit RAM registers as shown in Fig. 1 of the architecture/microprogram manual. Addressing of the register file's 16 rows by 16 columns (256 locations) may select one of the 80 registers as a source or destination of data to/from the bus. Certain of the remaining 256 addresses are assigned to registers or command codes external to the CPU (I/O) while other do not exist. This organization of the register file enables the CPU to communicate with I/O devices in the same manner in which it uses its own registers.

Addressing the register file

There are 256 possible address locations in the partially-populated register file, arranged in a 16 row by 16 column configuration. The MSB of the source or destination field in the micro-instruction determines whether the address is of a register in the register file or is of another register in the CPU. If a register file address is being decoded, the register to be used is selected by a 4-bit row and 4-bit column address.

The 4-bit row address comes from the source or destination field in the micro-instruction or from the R1, R2 fields in the PIR. The selection of the proper address source is done with a 4-input mux. This mux is controlled by a source/dest signal and a decode/R1I, R2I signal. The source/dest signal determines whether the source field or the Dest field from the UIR will be the row address. A decode of 15 or 16 in the UIR source or dest. field indicates that the address is not in the UIR, but will be in the 3-bit R1 or R2 field of the PIR. When the 3-bit field of the PIR is used to address the register, the MSB of the resultant 4-bit address is "0".

The 4-bit column address may originate from one of two sources, the "X" or "P" register. Normal column addressing is done VIA the "P" register. "P" is loaded, by instruction from the CPU, with the output of the priority encoder. The priority encoder encodes data and service requests from 16 I/O Lines. A comparator

compares the existing "P" register contents, with the output of the priority encoder. When a data or service request of higher priority than the one in the "P" register exists, an "interrupt" is routed to the sense demux. As the microprogram checks for "interrupts", "P" may be loaded with a new address if it exhibits higher priority. The other source of column address is from the "x" register. The "x" register may be loaded from the bus when used as a destination in a micro-instruction. The x register is 5 bits wide. Only the four LSB's may be used as a column address. The MSB of the x register is used to select, via a mux, either the x or P register as the column address.

The 4-bit Row and 4-bit column addresses are extended to various I/O devices as previously described.

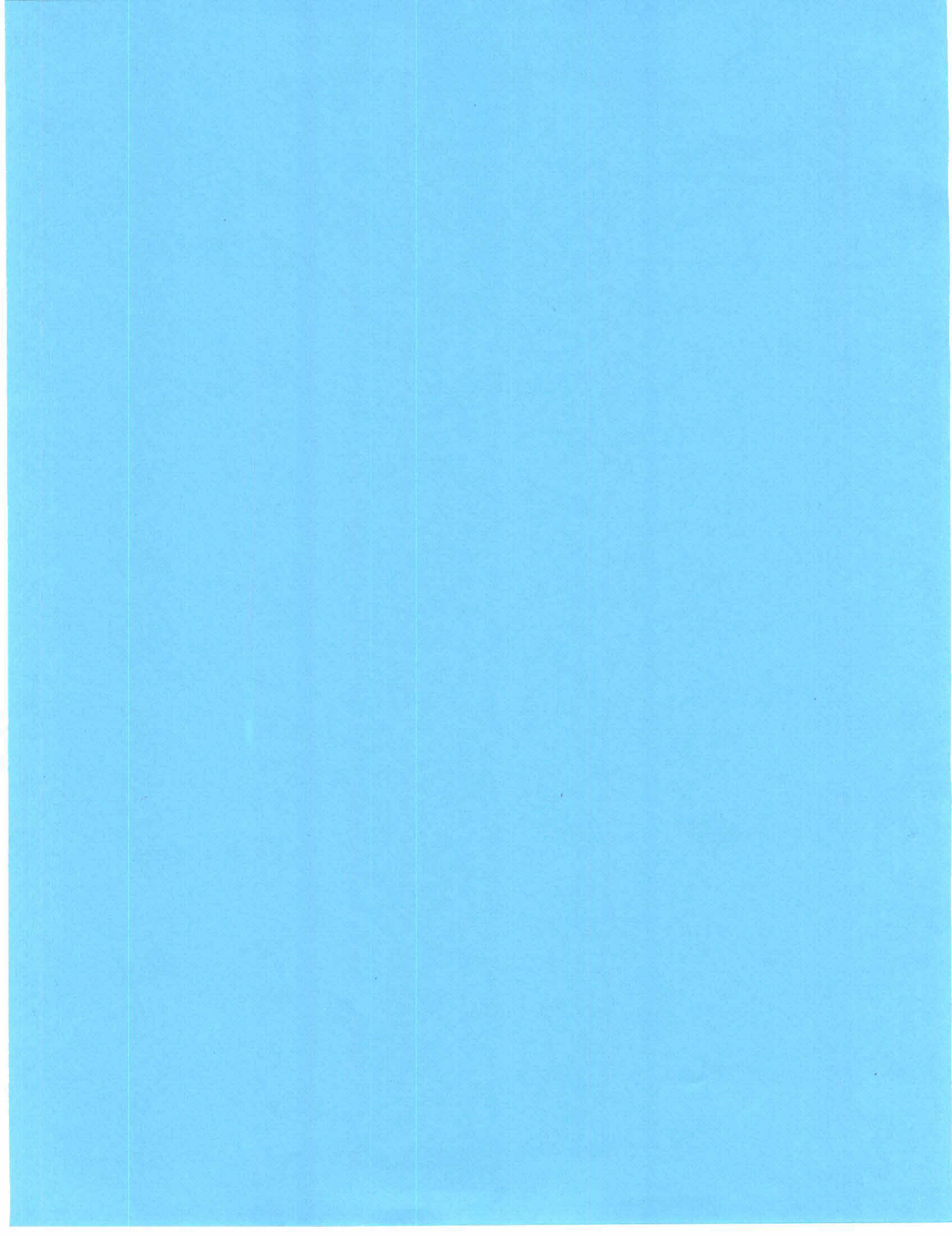
Memory Interface

The CPU may access memory via a micro-instruction port operation. The main memory takes three CPU micro-cycles for one memory cycle. The CPU shares the memory with other devices and each device has a fixed priority. If the CPU is not the highest priority device requesting memory, the port operation must remain latched up until the CPU memory request is satisfied. The memory interface control latches up the port request, decodes the port request, and controls the reading or writing from the memory.

The micro-program has the ability to halt the machine until a CPU memory operation is completed by executing a hold on memory (HDM) port command.

The 16-bit memory address register (MAR) is gated onto the bus by "CPU Memory Enable" at the beginning of the memory cycle. The 16-bit Memory Data Register (MDR) is either gated to, or from the memory on a bi-directional bus depending on whether or not, the operation is a write or read. A CPU memory operation may be initiated from the control panel while the CPU is halted. The MAR & MDR may be loaded or displayed by the control panel. The MDR may be loaded one byte at a time from the 8 LSB's of the UBUS. The LSB of the MAR is used to determine which MDR byte is loaded when the DEST. field is SMDR. The 8 LSB's of the MDR may be sourced either onto the left or right byte of the UBUS depending upon the LSB of the MAR when the source field is SMDR. A full 16 bit word may be loaded into the MDR or placed onto the bus from the MDR when DEST./SOURCE MDR is used.

The memory protect function works as follows: The protect register is loaded from the UBUS and contains two bytes. Each byte represents a page limit of the 8 MSBs of memory addressing. The high order byte of the protect register represents the highest protected page and the low order byte represents the lowest protected page. An error condition is set when the protect control is activated by the proper port operation and may be selected as a branch condition.



MEMOREX 7100

MEMORY SYSTEM

R. PETERSON

7/19/72

MEMOREX CONFIDENTIAL

1.0 SYSTEM MAIN MEMORY

The system main memory (memory) is a MOS RAM data storage device. The memory system utilizes a memory bus configuration, servicing up to eight different devices on the bus. The highest priority on the bus is dedicated to the memory refresh circuitry.

The MOS RAMs that comprise the memory are 8K 9 bit byte modules that can be expanded to a total memory size of 128K bytes. One bit of each byte is for odd parity. Memory access time is 800 nanoseconds. Memory cycle time is 1.2 us. The memory cycle starts on the first system minor cycle after a memory request is raised* by any device on the memory bus.

2.0 MEMORY PRIORITY

Any one of eight different devices can request a memory cycle. However, the highest priority (0) has been dedicated to the memory refresh circuitry, the purpose of refresh is explained in 5.4. The remaining seven devices are assigned priorities as listed in Table 1.

TABLE 1

MEMORY PRIORITY ASSIGNMENTS

PRIORITY	DEVICE
0	Memory Refresh
1	} To Be Determined
2	
3	
4	
5	
6	
7	

*The use of the words "raised" or "goes up" should be interpreted to mean the subject signal or condition becomes true. In order to determine actual logical voltage levels reference must be made to the first level logic diagrams.

The memory priority logic determines the highest priority device with and active memory request and gates its address out on the Select Address Bus. This notifies the requesting device that a memory cycle has been granted and the memory cycle begins.

3.0 MEMORY MODE CONTROL

The two signals WR HI and WR LO are generated by the memory user and places the memory in the proper mode as shown in Table 2. In a byte write mode memory address bit 15 is decoded into WR HI and WR LO. A full memory write (FMW) or a read are also encoded into WR HI and WR LO. This encoding is done by the user. Memory address bit 15 does not go to the memory. WR HI and WR LO are treated as part of the Memory Address Bus (MAB).

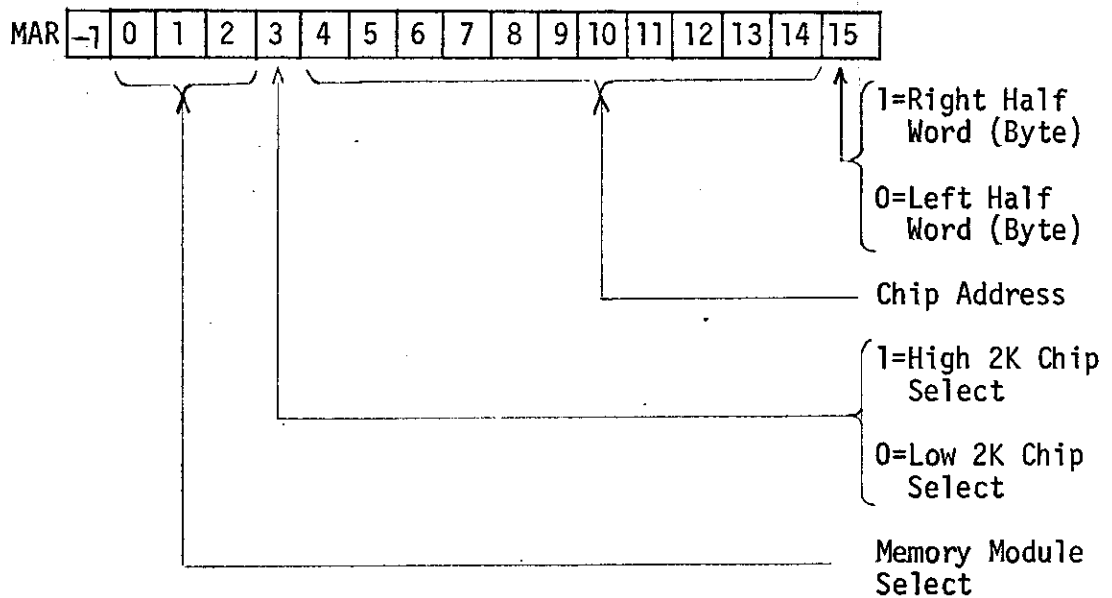
TABLE 2
ENCODING OF WR HI AND WR LO

	MAB 15	RESULT	
		WR HI	WR LO
BYTE WRITE {	0	0	1
	1	1	0
FMW	X	1	1
READ	X	0	0

4.0 MEMORY ADDRESSING

During normal CPU operation, the address of the desired memory location is placed in the Memory Address Register (MAR) of the device requesting a memory cycle. When a device is selected, the contents of its MAR are placed on the Memory Address Bus which goes to all the memory modules and selects the appropriate module and byte. The format of the memory address is shown in Figure 1.

FIGURE 1
MEMORY ADDRESS FORMAT

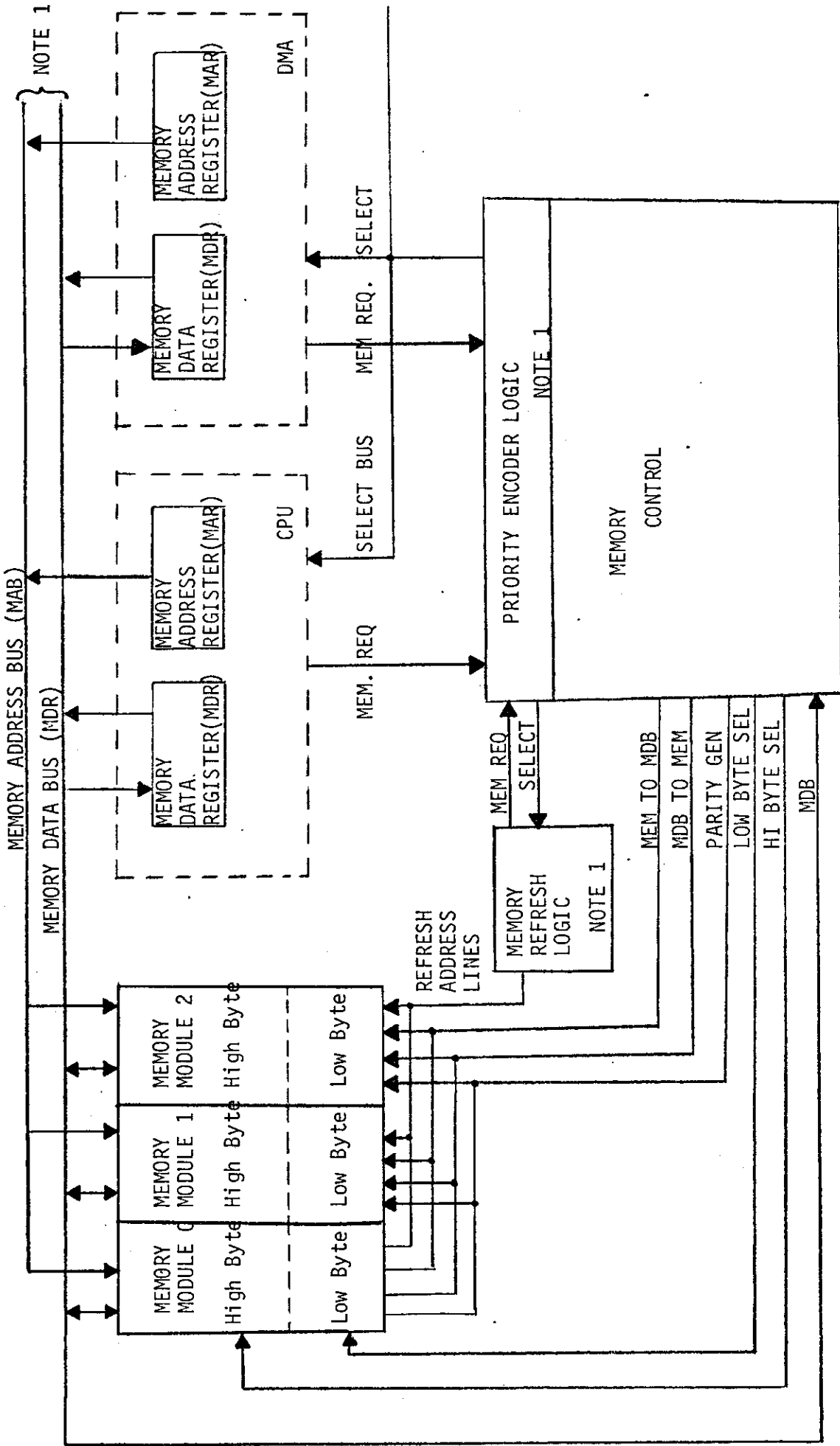


5.0 MEMORY OPERATION

The following paragraphs and Figure 2 provide a functional description of the main memory operation.

5.1 READ

The address of the memory location to be read is placed in the MAR of the device requesting a memory cycle. The requesting device also generates a memory request (MEM REQ), that is sent to the PRIORITY ENCODER LOGIC. The address of the requesting device with highest priority is placed on the Select Address Bus. The select address is decoded by the requesting device and is interpreted to mean that a cycle is granted. After a device is selected, its MAR is gated to the MAB to address the desired memory location. When the Memory is ready to transfer data, the memory control logic generates MEM DATA GATE. The generation of MEM DATA GATE causes the contents of the



NOTES:

1. MEMORY REFRESH is permanently connected to the highest priority (0). Therefore, a maximum of 7 additional devices can be connected to the MAB and MDB.

7100 MEMORY SYSTEM, Simplified Block Diagram

addressed memory location, to be placed on the Memory Data Bus (MDB). Parity is checked by the memory control logic to ensure that the data has correct odd parity. If incorrect parity is detected, READ PARITY ERROR is generated. At the fall of MEMORY DATA GATE the data on the MDB is read into the appropriate device MDR. The memory always reads a full word.

5.2 MEMORY WRITE

After the user's MDR and MAR are loaded, the user generates a memory request (MEM REQ) that is sent to the PRIORITY ENCODER LOGIC. The address of the highest priority requester is placed on the SEL ADR BUS. When the user decodes his address, the appropriate MAR and MDR are gated to the memory buses. The memory control logic then examines the data on the MDB and generates the correct odd parity for one or both bytes, depending on the type of write operation. After the correct parity is generated, MEMORY DATA GATE causes the data on the MDB and generated parity bit(s) to be written into the addressed memory location. When MEMORY DATA GATE returns to the inactive level, the selected device is disconnected from the memory buses and the write operation is complete.

5.3 WRITE HALF WORD

The half word write operation is similar to the full word write except MAR 15 is decoded to determine which half of the memory is to be written. The decoded MAR 15 generates either WR HI or WR LO and the appropriate byte with correct parity is written into memory.

5.4 MEMORY REFRESH

The memory stores a bit by charging or discharging a capacitance. Because these charges leak off the memory must be rewritten or

"refreshed" every 2 ms. To accomplish refresh the 32 low order addresses must be generated and a write cycle with the WRITE signal inactive is executed. This is accomplished by the memory refresh logic. Every 128th cycle, the Refresh Address Flip Flop (REF ADD FF) is set. This generates the highest priority memory request, which is sent to the PRIORITY ENCODER LOGIC.

The address of the refresh requestor will be placed on the Select Address Bus. The column address is generated by the REFRESH ADR CTR and is placed on the low order memory address lines (MAB 10 thru MAB 14). A memory cycle is executed with chip enable and write inactive. T_5 resets the REF flip flop. The trailing edge of REF increments the REF ADR CTR by one, leaving it ready for the next Refresh Cycle in 60 microseconds.

6.0 MEMORY TIMING

Figure 4-3 contains the timing of all memory operation. The memory is able to interface to the various users because it is time bound to them by the system clock shown at the top of Figure 4-3.

PH1, PH2, PH3, and chip sel, are signals which are used for "house-keeping" within the memory chip itself and will not be treated in this document.

When a user desires to access the memory that user raises MEM REQ during the shaded time shown in the figure. All MEM REQ's must be stable 200ns before the next clock pulse. The priority encoder places the address of the highest priority requester on the SEL ADR bus. The 200ns is required to allow SEL ADR to settle and be decoded by the users. At the fall of the next clock pulse the highest priority requestor may begin memory access. The memory

control activates MEM BUSY. MEM BUSY is used internally to disable the priority encoder once a user has been given memory access. A selected user must have a valid address on the MAB and WR HI and WR LO stable 50 ns after selection, this allows 100 ns bus settling time. WR HI and WR LO are explained in Section 3, MEMORY MODE CONTROL.

MEM DATA GATE is used internally to gate the memory to the MDB in the case of memory read or gate the MDB into the memory in the case of memory write. Determination of memory mode (read or write) is explained in Section 5.

The signals labeled DATA IN TO CHIP and DATA OUT OF CHIP are for reference only showing the time relationship of the actual data transfer. The times at which data must be on the MDB for a system concept are noted on the MEM DATA GATE as explained previously in this section.

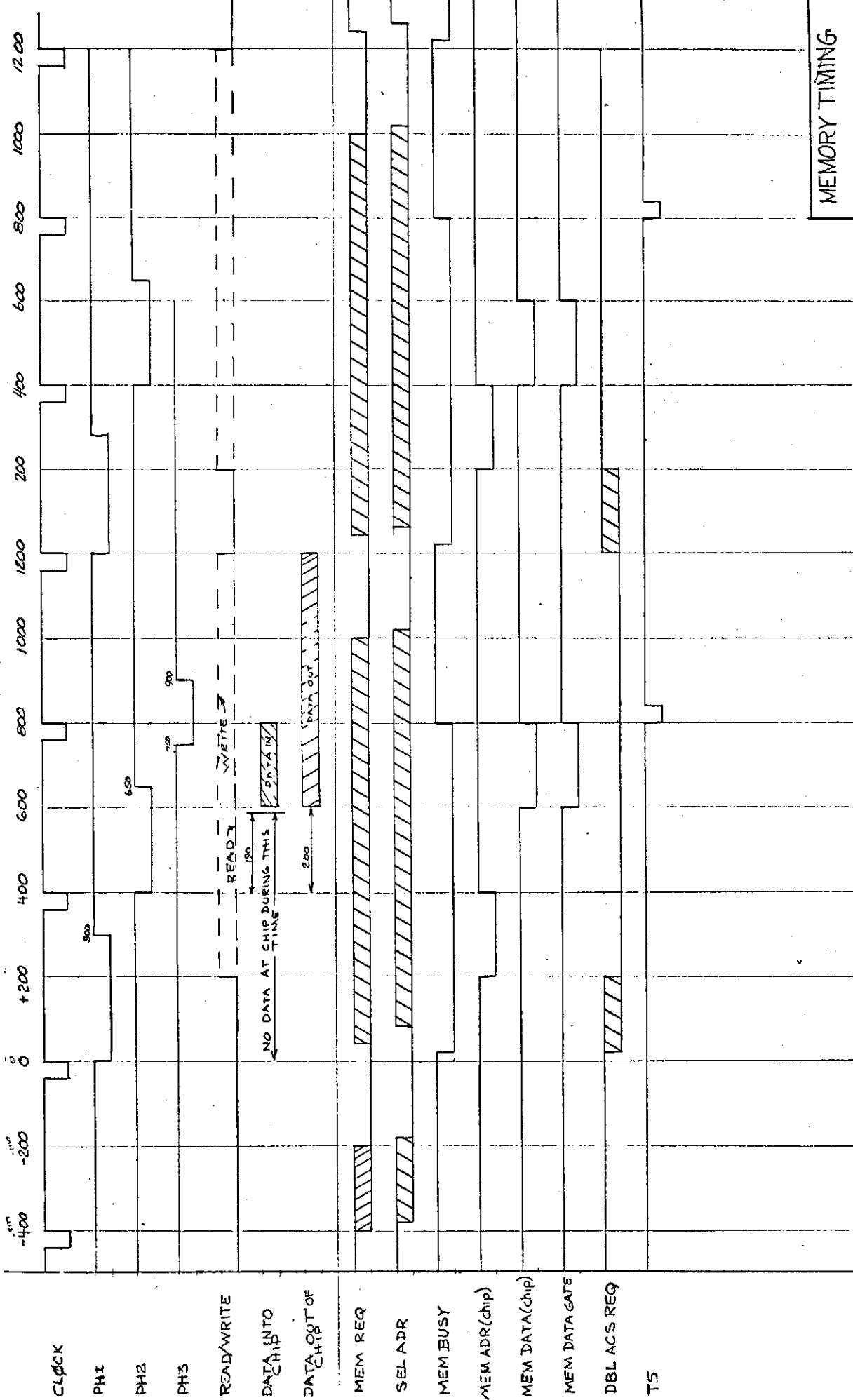
7.0 DOUBLE ACCESS

DOUBLE ACCESS provides two consecutive memory cycles either read or write. DOUBLE ACCESS is initiated when the CPU executes a TEST READ (TSR) port operation. The purpose of DOUBLE ACCESS is to enable micro-code implementation of a TES and SET function. To accomplish DOUBLE ACCESS the user raises double access request (DBL ACS REQ) after selection. DBL ACS REQ must be held up until the MEM BUSY line goes up for the second memory access cycle. The SEL ADR lines will operate normally during the first memory cycle, but no address will appear on the SEL ADR lines for the second cycle. It is not necessary for the user to raise his memory request for the second memory access. Once a double access has been granted the second cycle will usurp any requestor including refresh.

8.0 SECOND LEVEL LOGIC

The following second level logic prints are included:

MEMORY MODULE	40.10
MEMORY CONTROL	40.20

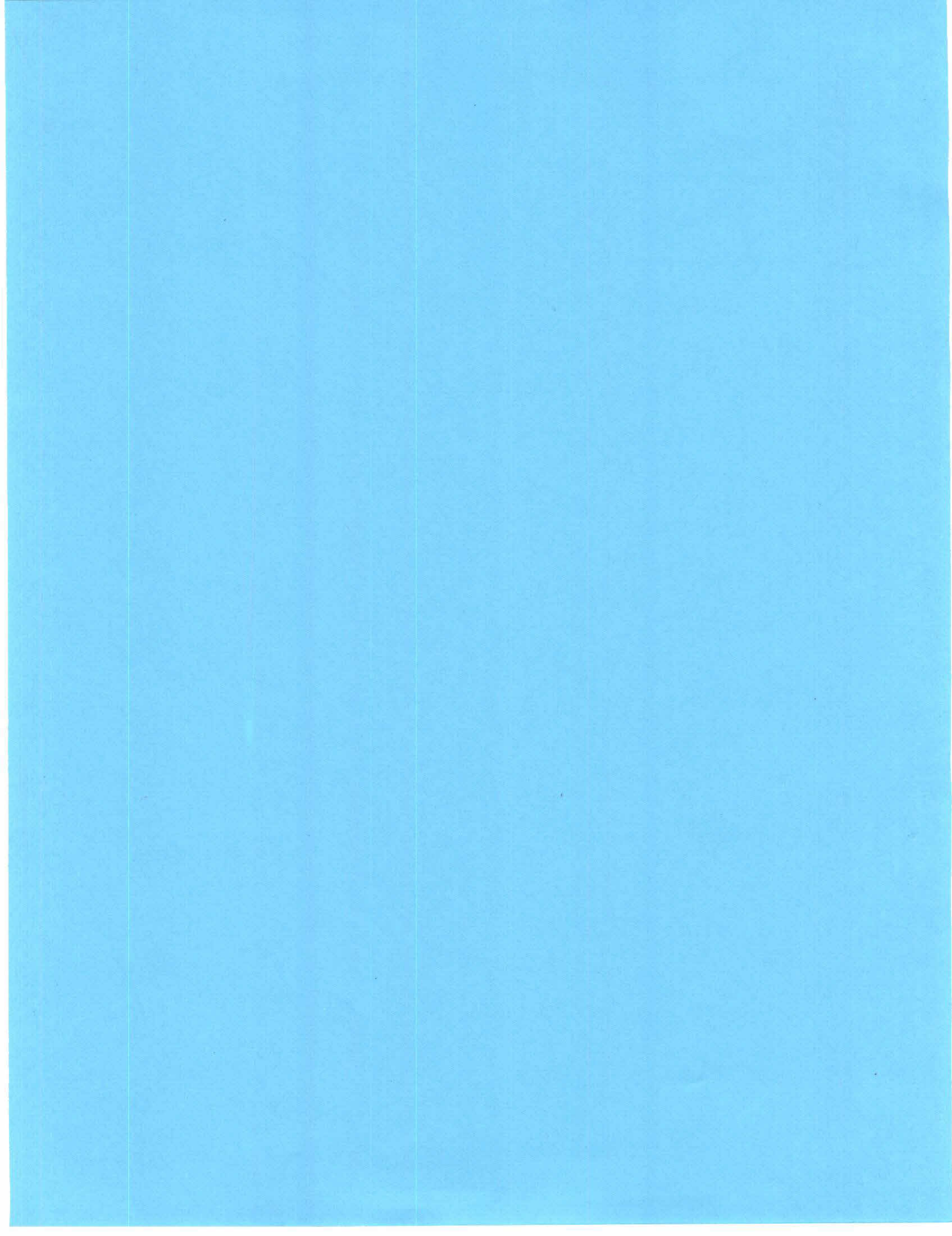


MEMORY TIMING

7-6-72 HAP

FIG.4-3 MEMORY TIMING

- NOTES:
- 1 Down level is always true, unrelated to schematics
 - 2-DIAGONAL LINES INDICATE DON'T CARE REGION



MEMOREX 7100

SYSTEM CONTROL AND DISPLAY PANEL

A. Heme1

7/19/72

MEMOREX CONFIDENTIAL

1.0 INTRODUCTION

The system control and display panel has seven basic functions:

- a) To control the mode in which the CPU operates or else halt it.
- b) To display the contents of various registers on 16 data lites.
- c) To alter the contents of any of four registers with 16 data switches.
- d) To read and write into the main memory.
- e) To control the loading of initial programs for various I/O devices.
- f) To display a parity error condition and allow the operator to reset it and restart the CPU.
- g) To allow the operator to reset the system.

These functions are described in more detail in the subsequent sections.

2.0 MODE ROTARY AND START TOGGLE SWITCHES

- a) The PROCESS position allows the processor to run uninterrupted by the panel as soon as the START switch is actuated.
- b) The MICROADDRESS HALT (MUX) position causes the processor to run as soon as the START switch is actuated. Prior to this, the DATA SWITCHES should be set to the sum of the desired halt micro address plus one. The processor will then halt one minor cycle after the MUX reaches the desired address. This leaves the Micro Instruction Register (UIR) loaded with the micro code of the desired address.

If the START switch is then actuated, the processor will

continue running and again stop one minor cycle after the MUX reaches the desired address i.e., one less than the DATA SWITCH setting. If the DATA SWITCHES are changed while the processor is halted, nothing happens until the START switch is re-actuated. The processor then runs and again stops one minor cycle after the MUX reaches the new desired address which is one less than the new DATA SWITCH setting. If a memory cycle is still in process after the last step of the above run, it will continue until completion, including the loading of the MDR in a read operation.

- c) The MEMORY ADDRESS HALT (MAR) produces the same operation as in 1 b) above except that the MAR replaces the MUX and the DATA SWITCHES are set to the exact halt address rather than one increment higher.
- d) The SINGLE MICRO CYCLE position causes the processor to halt. The START switch gates 1 minor cycle clock to the processor each time that it is actuated. If a memory cycle is started by any one of these single steps, it will continue until completion, including the loading of the MDR in a read operation.
- e) The SINGLE MACRO CYCLE position causes the processor to halt. Actuating the START switch gates as many minor cycle clocks as are required in the microcode to execute the macro instruction. When the processor halts, it leaves the Micro Instruction Register (UIR) loaded with the micro code of the next macro instruction start address. All memory cycles are completed.

3.0 HALT SW

The HALT switch stops the processor until the START or IPL switch is actuated.

4.0 ALTER REG/MEMORY WRITE TOGGLE SWITCH WITH DISPLAY/ALTER ROTARY SWITCH IN 1 OF 4 REGISTER POSITIONS

If the processor is in the halt condition and the DISPLAY/ALTER rotary switch is in any one of the four register positions -- UPNT, UIR, MAR, or MDR -- then

- a) If the ALTER REG/MEMORY WRITE switch is reset, the DISPLAY/ALTER rotary switch reads the contents of the selected register on to the 16 DATA LITES. An 'on' lite indicates binary 1.
- b) If the ALTER REG/MEMORY WRITE switch is actuated, the contents of the 16 DATA SWITCHES are loaded into the register selected by the DISPLAY/ALTER rotary switch.

5.0 ALTER REG/MEMORY WRITE AND MEMORY READ TOGGLE SWITCHES WITH DISPLAY/ALTER ROTARY SWITCH IN MEMORY POSITION

If the processor is in the halt condition and the DISPLAY/ALTER rotary switch is in the MEMORY position, then the DATA LITES will display the source addressed by a GSD or SCB micro instruction in the UIR. If neither of these instructions is present, the lights will be out. No destinations will be enabled.

- a) If the ALTER REG/MEMORY WRITE switch is actuated, the MDR contents will be written into main memory at the MAR address.
- b) If the MEMORY READ switch is actuated, the MDR will be loaded by the contents of the main memory at the MAR address.

6.0 IPL SWITCH

Prior to actuating the IPL switch, the operator is expected to set the DATA SWITCHES to a code corresponding to the selected I/O device. The IPL switch halts and resets the system and then sets the UPNT to a pre-determined starting address for the initial program loading routine. The processor starts running again after actuation of the START switch.

7.0 HALT LITE

The HALT lite (LED) goes on whenever the system is in the halt condition regardless of what caused it.

8.0 PARITY ERROR

A main memory read parity error causes the processor to stop and the MEMORY PARITY CHECK lite (LED) to go on. Actuation of the CHECK RESET switch turns the lite off. The processor continues immediately after the START switch is actuated. During the un-reset parity error period, the MODE, START, ALTER and MEMORY READ, and IPL switches are disabled but the DISPLAY/ALTER SW may be used for display only.

9.0 DATA LITES

Light emitter diodes (LEDs) are used for the 16 DATA LITES. When the processor is in the halt condition, they display various sources, as described in Section 4 a) and 5, above. If the processor is running, the DATA LITES dynamically display the data on the processor source and destination BUS.

10.0 SYSTEM RESET

The SYSTEM RESET switch resets the processor registers, latches and flip flops. The main memory is not reset.

DATA LITES

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15

DATA SWITCHES

-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-

MEMORY PARITY CHECK

HALT

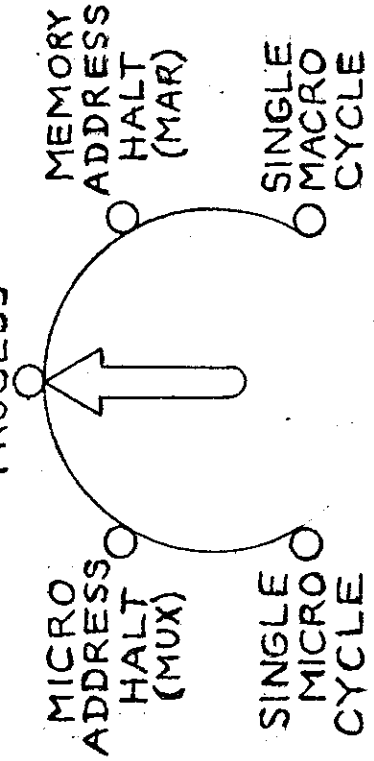
CHECK RESET

SYSTEM RESET

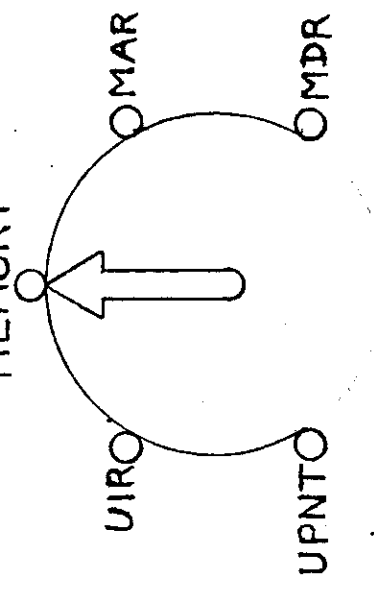
IPL

POWER

MODE SW
PROCESS



DISPLAY/ALTER SW
MEMORY

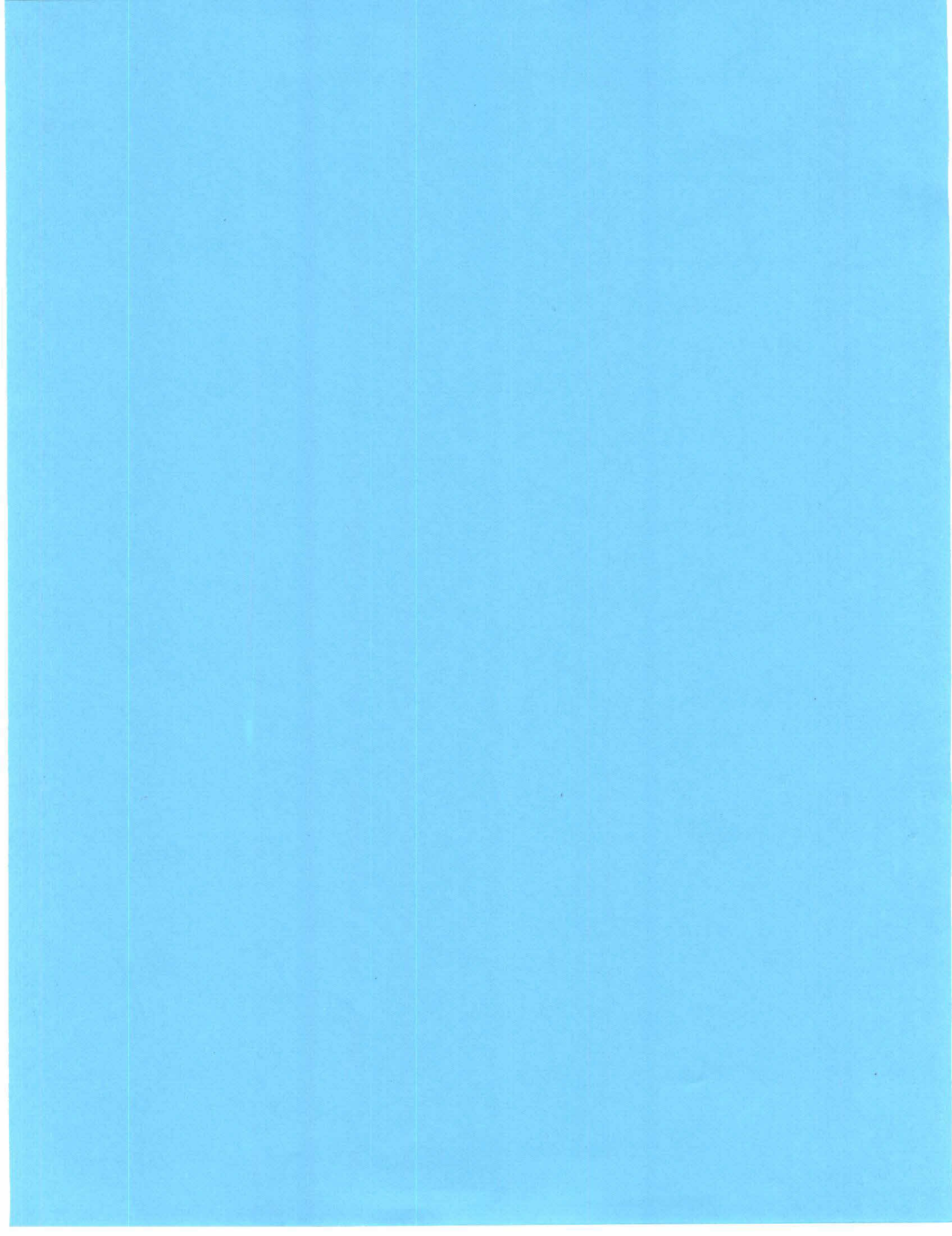


ALTER REG/ MEMORY
MEMORY WRITE READ

START

HALT

7100 CONTROL & DISPLAY PANEL
a.H. 6/22/72



MEMOREX 7100

I/O SYSTEM

R. HOEHNLE

R. HOLLAND

D. PESAVENTO

7/19/72

MEMOREX CONFIDENTIAL

CONTENTS

- 1.0 INTRODUCTION

- 2.0 7100 I/O ARCHITECTURE
 - 2.1 Integrated Adapters
 - 2.2 Micro-Bus Interface
 - 2.3 Memory Bus Interface
 - 2.4 Micro-Processor Control of Adapters
 - 2.5 I/O Timing Considerations

- 3.0 7100 I/O ADAPTER DESCRIPTIONS
 - 3.1 80 Column Card Equipment
 - 3.2 96 Column Card Equipment
 - 3.3 Printer
 - 3.4 Console
 - 3.5 Communications
 - 3.6 Selector Channel

FIGURES

- I Integrated Adapter Interconnection to 7100 System
- II 7100 System Register File
- III Integrated Adapter Block Diagram
- IV Detailed Adapter-CPU Interface

1.0 INTRODUCTION

This section will discuss the 7100 I/O Subsystem. The I/O Subsystem has been designed to make maximum use of CPU resources. The I/O Subsystem consists of peripheral devices that are connected to the CPU and main memory through control units that are integrated into the mainframe design. These integrated control units, or adapters, are directly controlled by a common micro-processor in the CPU.

2.0 7100 I/O ARCHITECTURE

Peripheral devices on the 7100 System connect to the system through integrated adapters (control units). These adapters are controlled by the micro-processor in the CPU and make use of common resources in the CPU such as ALU, address and data registers and a scratch pad memory used as a register file.

Figure I outlines the basic interconnection of integrated adapters to the system. All I/O adapters connect to the micro-bus. The micro-bus provides basic control to the adapters for I/O initiation, termination and data transfer. The exceptions to this general rule are the disc adapter and the selector channel adapter, which, due to their high rate of data transfer, go directly to the memory bus for data transfer.

Each peripheral I/O adapter is assigned a processor state as seen in Figure II. Each processor state is assigned a column in the system register file. The system register file contains registers

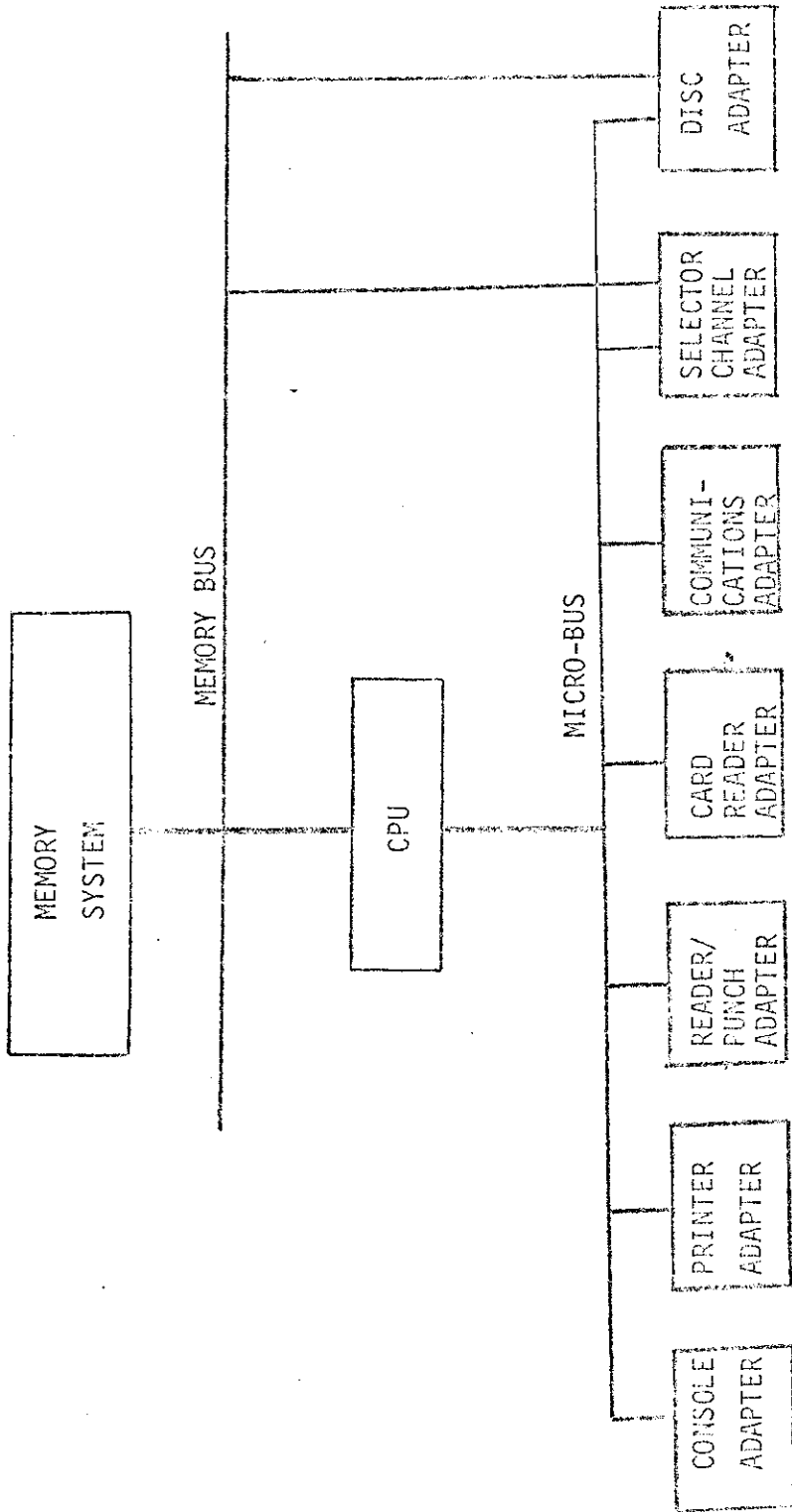


FIGURE I

INTEGRATED ADAPTER INTERCONNECTION TO 7100 SYSTEM

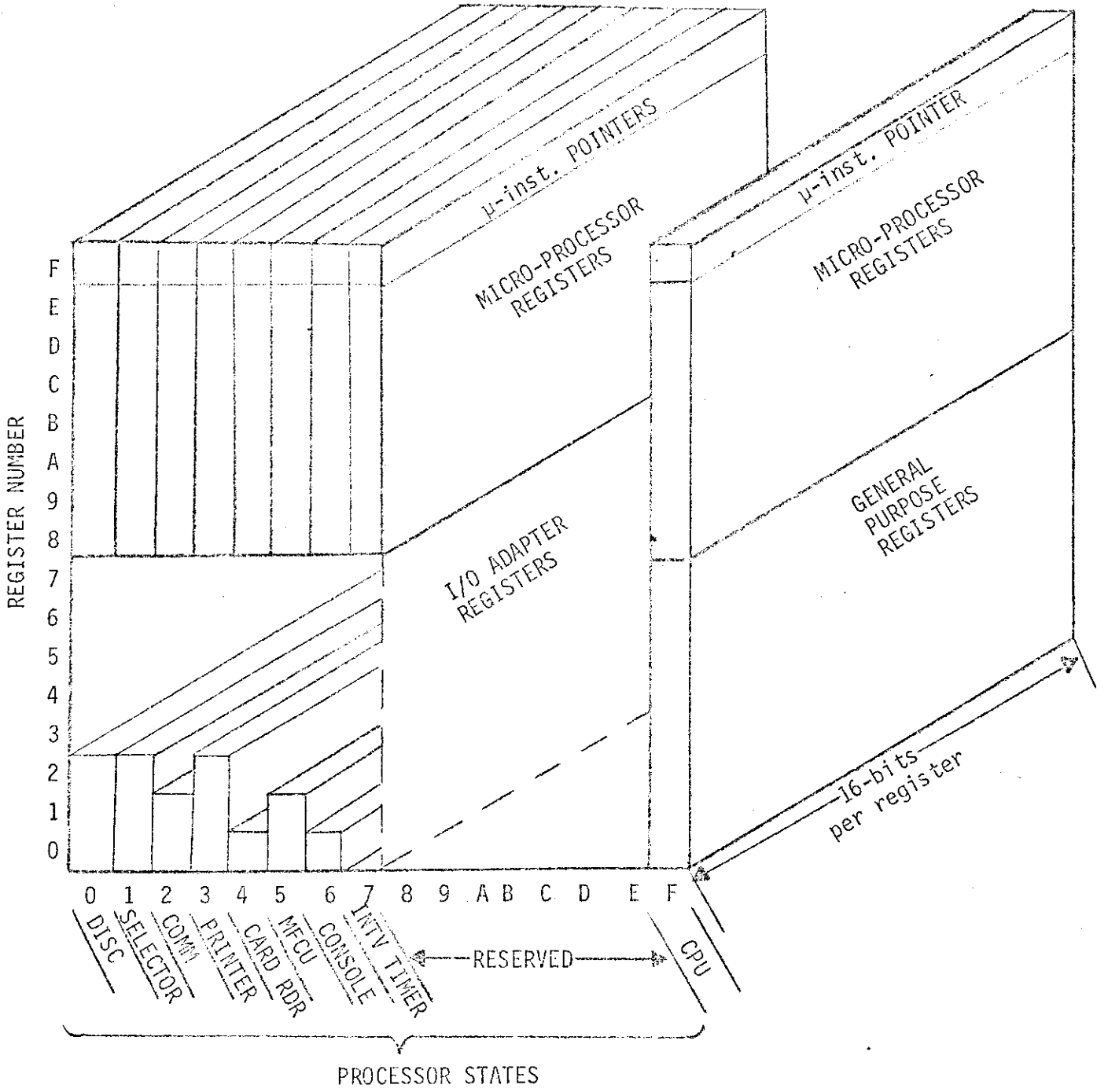


FIGURE II

7100 SYSTEM REGISTER FILE

8-F in Columns 0-7 and 0-F in Column F. Columns 8-E are reserved for future use.

Each I/O adapter has varying numbers (up to 8) of discrete registers located in the adapter. These registers can be addressed by the micro-processor as though they were located in the system register file. Thus, the micro-processor has 8 working registers plus up to 8 discrete adapter registers for each I/O processor state.

Register F is reserved in each processor state for the micro-processor program counter. Thus, each time a processor state change occurs, the entry point in the micro-program is obtained from Register F of the newly selected state.

2.1 Integrated Adapters

Integrated adapters on the 7100 System provide data buffering and control to the peripheral device. Because different types of devices have different characteristics, adapters are designed specifically to meet the needs of each type of peripheral device.

Figure III shows grossly what is required for most types of adapters. Each I/O device requires a certain amount of discrete logic to control it. This, in general, is in the form of latches that set and remember commands or conditions during an I/O operation. During data transfer, buffering of data is required between the I/O device and the CPU or memory. Each adapter must have logic to sequence the work flow through the adapter, (i.e., moving data in and out of the adapter, requesting CPU service, etc.).

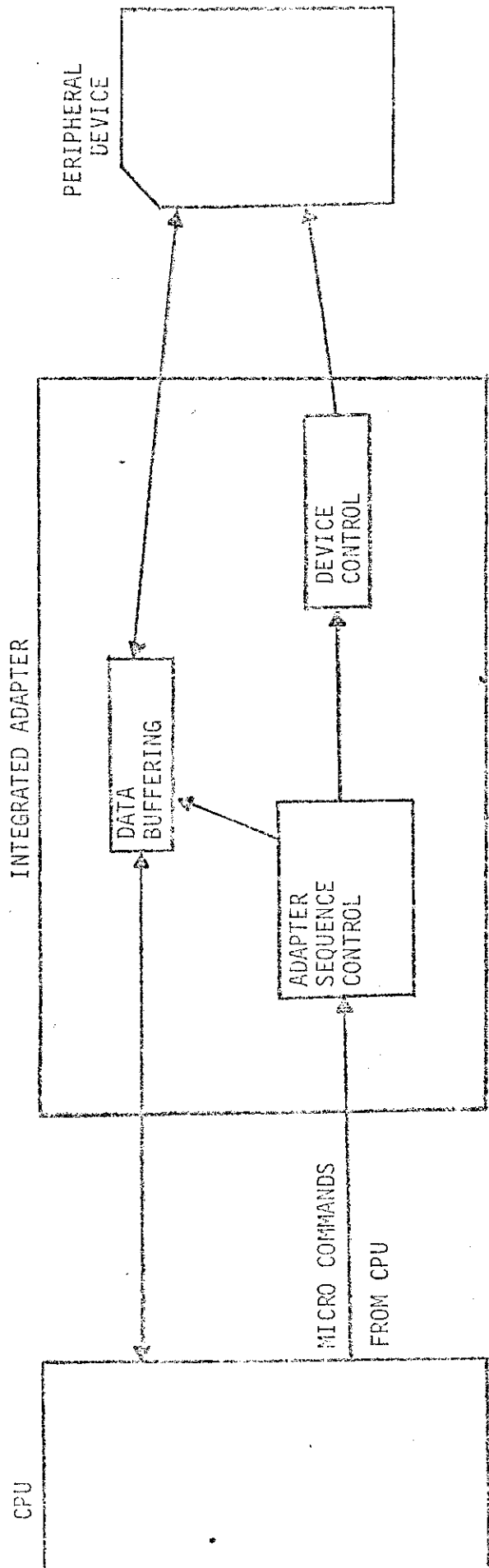


FIGURE III

INTEGRATED ADAPTER BLOCK DIAGRAM

In the 7100 System, the adapters are passive in nature and depend on the micro-processor to control them. In certain cases where the micro-processor is not fast enough to respond to peripheral needs (eg. serialization/deserialization of bit stream data on disc storage devices) discrete logic is provided in the adapter to perform these functions. In general, when a status change takes place in an adapter, the adapter requests service from the micro-processor and waits for the micro-processor to respond. Examples of this will be covered in Section 2.4, Micro-Processor Control of Adapters.

All I/O adapters contain certain common elements such as registers to hold status, device commands and buffer data. These registers are directly accessible by the micro-processor which can read or write these registers. In addition, the micro-processor can issue certain commands that cause the adapter to perform a function, such as, master clear, drop request line, initiate operation to device, etc.

2.2 Micro-Bus Interface

The micro-bus interface provides the vehicle to initiate I/O operations, transfer data and terminate I/O operations.

Figure IV shows a diagram of the micro-bus interface. This interface consists of:

1) Bi-directional, 16 bit Data Bus

This bus interfaces between the registers in the CPU (i.e., MDR, A feeder, B feeder, etc.) and one of eight registers in the adapter. (NOTE: Most adapters only require 2 or 3 registers.)

2) Register Address Line

These lines decode off of the micro-instruction register source and destination fields and select one of eight registers referred to above as Item 1.

3) Read/Write Lines

The read/write line signals the I/O adapter whether the selected register is to be a source or destination. The read line indicates a source, the write line indicates a destination.

4) Adapter Select Lines

These lines select 1 of 8 possible I/O adapters. I/O adapters should respond to signals on the micro-bus interface only when their assigned address is present on the adapter select lines.

5) Service Request Line

Each adapter has a unique service request line. The service request line is used by the adapter to signal the CPU that it needs the services of the micro-processor. At an appropriate time, the CPU will use the service request line, through a priority network to select a new CPU state which will affect the adapter select lines.

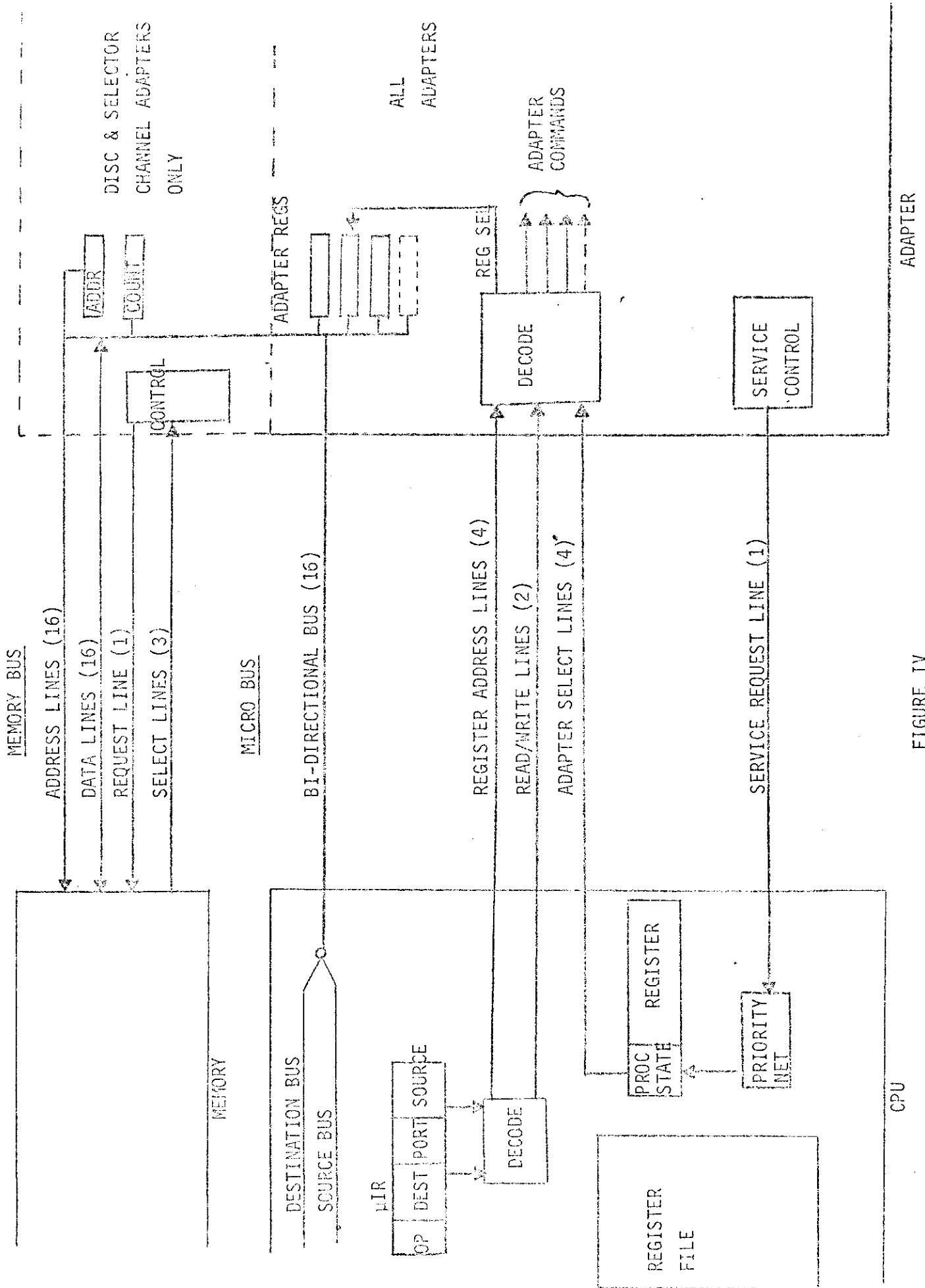


FIGURE IV
DETAILED ADAPTER - CPU INTERFACE

2.3 Memory Bus Interface

The memory bus interface is used by certain adapters that have high speed (> 27K byte) data transfer requirements such as the disc storage unit and the selector channel.

The memory bus interface consists of:

- 1) 16 Address Lines
- 2) 16 Data Lines
- 3) 1 Request Line per Adapter
- 4) 3 Select Lines

Refer to "Memorex 7100 Memory System" description for timing and interfacing requirement.

2.4 Micro-Processor Control of Adapters

The micro-processor, as described in previous sections, can access registers in each of the adapters on the 7100 System. In addition to loading and reading these adapter registers, the micro-processor can instruct the adapters to perform certain programmed sequences.

These instructed sequences from the micro-processor can be interpreted in two ways. One, the act of loading adapter registers can be used by the adapter to initiate certain operations. For example, assume that one register in the adapter is assigned to be a command register for a peripheral device, then, the act of loading the register could signal the adapter to start a procedure with the peripheral device, the exact operation being dependent on the content of the register just loaded.

Another method of instructing the adapter can be through the use of register address decodes of registers not present in the adapter. For example, assume that a particular adapter has 3 discrete registers. Since each adapter has an addressable range of 8 registers, 5 register addresses would normally be unused. If these unused addresses were decoded and interpreted by the adapter as commands then 10 unique (5 read and 5 write) instructions would be decoded through normal micro-program access to adapter registers. Refer to adapter decode block in Figure IV.

During a normal I/O sequence at the point of I/O initiation the system will switch to the processor state associated with the pertinent adapter (see Figure II).

The micro-processor will access memory to obtain the "command word" which describes the I/O operation that is to take place. The micro-processor will make any necessary error checks for format and content of the "command word." If no errors are detected, the micro-processor will begin an I/O command sequence with the adapter.

Once the operation is underway, the micro-processor will relinquish control to another processor state until data transfer is required (this time varies with the peripheral device).

When the adapter detects that an access to memory is eminent, it will raise the service request line to the micro-processor priority network.

In due time, the micro-processor will respond by again switching to the processor state for the requesting adapter. Because the micro-program initiated the I/O operation, it usually knows what type of response will be required to an ensuing request and will have the micro-program counter pointing to the proper (i.e., read or write) micro-program routine.

Once the switch has taken place, the micro-program will access memory and send the data to the adapter data buffer (write sequence) or access the adapter data buffer and send the data to memory (read sequence). In either case, the micro-program will update the memory address pointer and the data block counter. The micro-program will detect for end of block.

If the end of block specified by the "command word" and the end of record specified by the peripheral device do not coincide an error status condition will occur.

After each byte of data is transferred, the micro-processor relinquishes control to another processor state.

When all of the requested data has been transferred, the micro-program goes through a clean up procedure to terminate the I/O operation. If command chaining was requested by the "command word" in memory, the micro-program will access the next "command word" and the procedure starts all over again.

2.5 I/O Timing Consideration

1) Adapter Selection Address Lines

The 4 adapter selection address lines indicate to the I/O adapter which block of extended registers the processor wishes to communicate with. The 4 lines make it possible to address 16 separate register blocks. These address lines must be stable 130 n sec. after the trailing edge of the system clock for a write request and 90 n sec. after the trailing edge of the system clock for a read request.

2) Register Address Lines

The 4 register address lines indicate to the I/O adapter which register in the block of extended registers the processor wishes the address. The 4 lines make it possible to address 16 different registers. The address line must be stable 170 n sec. after the trailing edge of the system clock for a write request and 130 n sec. after the trailing edge of the system clock for a read request.

3) I/O Write Line

The I/O write line indicates to the I/O adapter that the processor wishes to perform a register write function.

The write line must be stable 130 n sec. after the trailing edge of system clock.

4) I/O Read Line

The I/O read line indicates to the I/O adapter that the processor wishes to read the contents of a register.

The read line must be stable 130 n sec. after the trailing edge of system clock.

5) Micro Bus

The 16 line micro bus is a bi-directional communication link between the processor and the I/O adapters. In a write operation, the I/O adapter must have the bus stable 170 n sec. after the trailing edge of system clock. In a read operation, the bus will be stable 230 n sec. after the trailing edge of system clock.

3.1 EIGHTY COLUMN CARD INTEGRATED READER PUNCH ADAPTER (IRPA80)

3.1.1 GENERAL

The IRPA80 is the Integrated Adapter/Controller which contains the hardware interface and control logic required to operate the MRX 8205 Card Reader Punch (Control Data Corp. Model 9280). The IRPA80 may also be used as the Adapter/Controller for MRX 8010 models 1, 2, and 3 (Documation Inc., models M300L, M600L, and M1000L).

3.1.2 PERIPHERAL DEVICE DESCRIPTION

3.1.2.1.1 The CDC Model 9280 will both read and punch eighty column cards in one pass. It is capable of reading at 500 cpm or punching at 100 cpm. The Model 9280 consists of a column oriented, card read-punch unit complete with Hopper, Stacker, internal Control Unit, and Power supply.

3.1.2.1.2 The Model 9280 uses fiber optics to accomplish the photo-electric reading. The read system is resynchronized on each punched column to insure reliable reading of "old" or mutilated cards. An added feature is the punch "Skip-Out" capability which allows a card to be fed at transport rates after the final desired column is punched. It also has a stacker card "Offset" capability.

3.1.2.2.1 The Documation M Series readers will read eighty column cards, column-by-column. The M300 L reads at a rate of 300 cpm, the M600L reads at 600 cpm, and the M1000L reads at 1000 cpm.

3.1.2.2.2 The M Series Mechanisms consist of a Hopper, Read Station, and stacker. All units use long-life LEDs to accomplish the photo-electric reading.

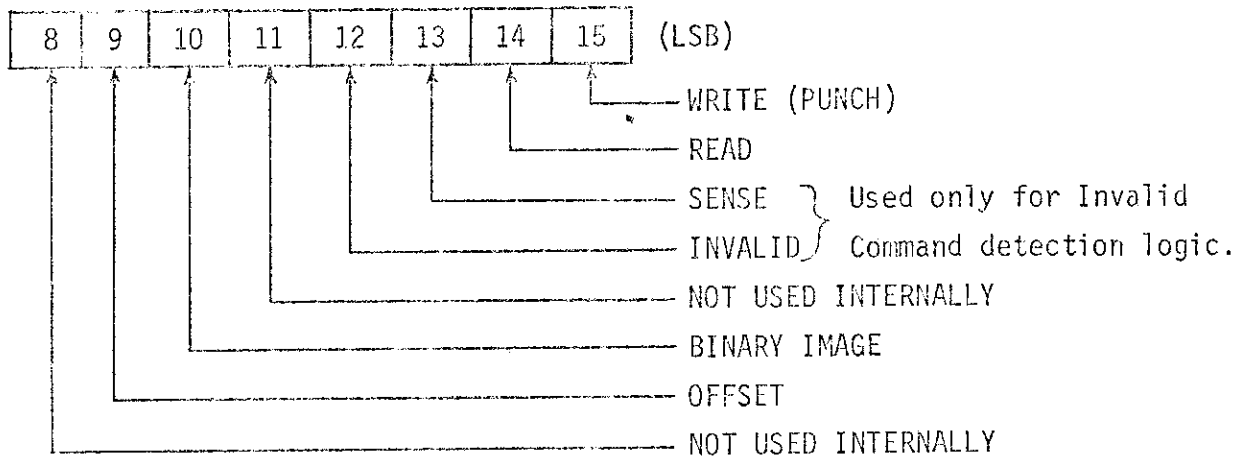
3.1.3 IRPA80 DESCRIPTION

3.1.3.1 The Eighty Column Integrated Reader-Punch adapter logic is contained on two plug-in boards. This logic is divided between a "Register" board and a "Control" board. The IRPA interfaces to the micro-processor via the I/O micro-bus, the row and column address lines, the read/write control lines, and the service request line. For additional information on the I/O interface, refer to Section 2.0.

3.1.3.2 IRPA80 Register Board

The IRPA80 Register Board contains the Command Register, Invalid Command detection logic, Punch Data register, Input Data register, Hollerith conversion logic, and Hollerith Validity check logic. (Refer to IRPA80 logic diagram or Block Diagram 3.1.4)

3.1.3.2.1 The Command Register is 8 bits in length. During the I/O initiation procedure the micro-processor fetches the Command Byte from the SI/O instruction and loads it into this register. The Command Byte is then available to the micro-processor for testing and, normally, will remain in the command register for the duration of one I/O operation. The bits in this register correspond to bits 8 thru 15 of the micro-bus and their meanings are assigned as follows:



3.1.3.2.2 The Invalid Command Detection logic tests the four least significant bits of the Command register. Three of the sixteen possible combinations will be sensed as an invalid command. These are:

- 1) 0000 Invalid
- 2) 1000 Transfer in Channel
- 3) 1100 Read Backward

The Invalid Command term is one of the inputs to the Command Reject logic on the Control board. However, the Invalid Command term will not cause a "Command Reject" to occur unless it is present when the micro-processor executes a "Start I/O" (FEED) control function.

3.1.3.2.3 The Punch Data Register is the outbound data buffer during Card Punch operations. It is 12 bits in length, and they correspond to bits 4 thru 15 of the micro-bus. This register is loaded by the Write Decoder "Load Punch Data" term. The register outputs are connected to the inputs of the EBCDIC/Hollerith ROM converter. A punch multiplexer is used to select either the converted hollerith character or the Punch register outputs (bypassing the converter logic) and to present the resultant Punch data to the peripheral interface. The punch multiplexer data selection is controlled by the Binary image bit of the Command register. Either an "Image" character or a "converted" character is always present on the Punch Data lines. This Punch Data is also available to the Input Multiplexer, thus providing the capability for diagnostic testing of the outbound data paths.

3.1.3.2.4 The Input Data Register is the inbound data buffer during Card Read operations. It is 12 bits in length, and they correspond to bits 4 thru 15 of the micro-bus. Normally, this register is loaded with data read from one column of the card in the device read station, and is loaded on the leading edge of the Busy signal (data index) from the peripheral device. This register may also be loaded from the micro-bus, thus providing the capability for diagnostic testing of the inbound data paths. The register outputs are connected to the Hollerith/EBCDIC conversion logic. A Binary Read multiplexer is used to select the converted EBCDIC character or the Input Data register (bypassing the converter logic) and to present the resultant input data to the Input Multiplexer.

3.1.3.2.5 A Hollerith Validity Check is performed on inbound data if the IRPA is not in Binary Image mode. The Validity check logic inputs are connected to the seven least significant bits of the Input Data register. If more than one of these seven bits is set, the Hollerith Valid term will be not true. If the Hollerith character is not valid, and the IRPA is not in Binary Image mode, the Data Check flag will be set.

3.1.3.3 IRPA80 Control Board

The IRPA80 Control Board contains the Read Control and Write Control decoders, the Input Multiplexer, the Data Transfer Control Logic, Data Check logic, Adapter Status logic, and the Peripheral Device Status Logic.

3.1.3.3.1 In order to perform an I/O operation related to the IRPA80 or the attached peripheral device, the micro-processor must first establish the correct Column Address to select this Adapter. The micro-processor must then execute an I/O micro-instruction: which will establish the I/O Row Address for the register, or control function, desired; and which must raise either the I/O Read signal to specify that the IRPA80 is the information source, or the I/O Write signal to specify the IRPA80 for a control function or as an information destination.

3.1.3.3.2 The I/O Row Address is presented to the Read Decoder, the Write Decoder, and the Input Multiplexer. The Input Multiplexer selects the information specified by the I/O Row Address and, if it is a Read operation, the output drivers are enabled, placing the information on the I/O micro-bus. The selected information is one of the following:

- 1) Input Data (sourced from the Input Data register or the Hollerith/EBCDIC converter)
- 2) Punch Data (sourced from the Punch Data line)
- 3) Command Byte (sourced from the Command register)
- 4) Status (Attention, Busy, Device End, Unit Check, Request In)
- 5) Sense (Command Reject, Intervention, Data Check, Overrun, Not Ready, Device Status, CPU Data Request)

3.1.3.3.3 The Write Decoder is enabled by the I/O write term and, depending on the Row Address contents, decodes one of the following Register Load or Control Function terms:

- 1) Load Input Data (register load)
- 2) Load Command Byte (register load)
- 3) Load Punch Data (register load)
- 4) Start I/O (control function)
- 5) Halt I/O (control function)

- 6) Clear Status (control function)
- 7) Reset Device (control function)

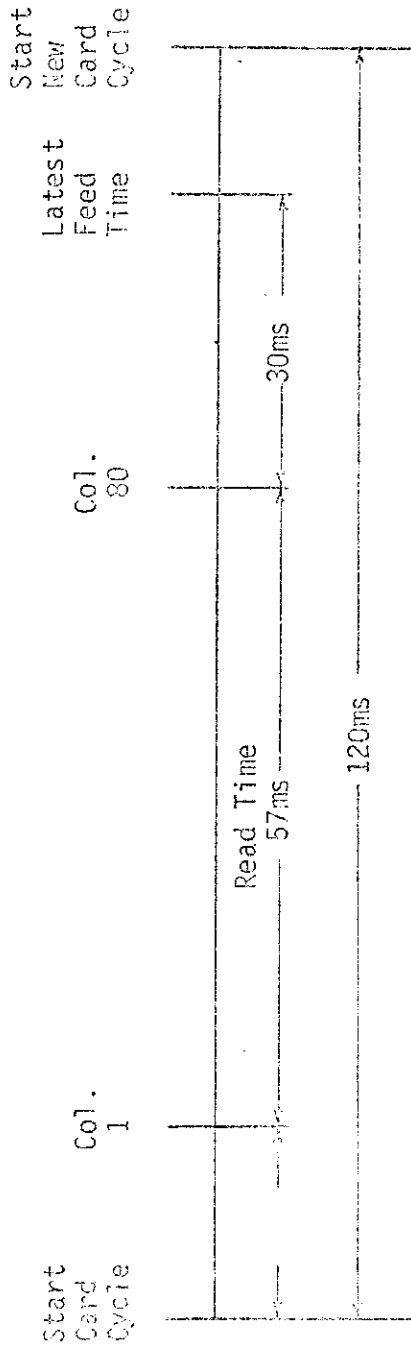
3.1.3.3.4 There is one Write address function and three Read address functions not used by the IRPA80. These are interpreted as illegal addresses and, should the micro-processor attempt to execute one of these when the Column Address is pointing to the IRPA80, the Command Reject Flag will be set to indicate an error condition.

3.1.3.3.5 The Active Flip Flop is set if the peripheral device is "Ready" and the micro-code issues a Start I/O control function to the IRPA80. It is reset by the Halt I/O control function. When the Active FF is reset, all service requests are inhibited except "Attention". Also, all data transfers between the IRPA80 and the peripheral device are inhibited. However, all Control functions and data transfers between the IRPA80 and the micro-processor are allowed in order to provide for diagnostics, adapter testing, and I/O initiation/termination procedures.

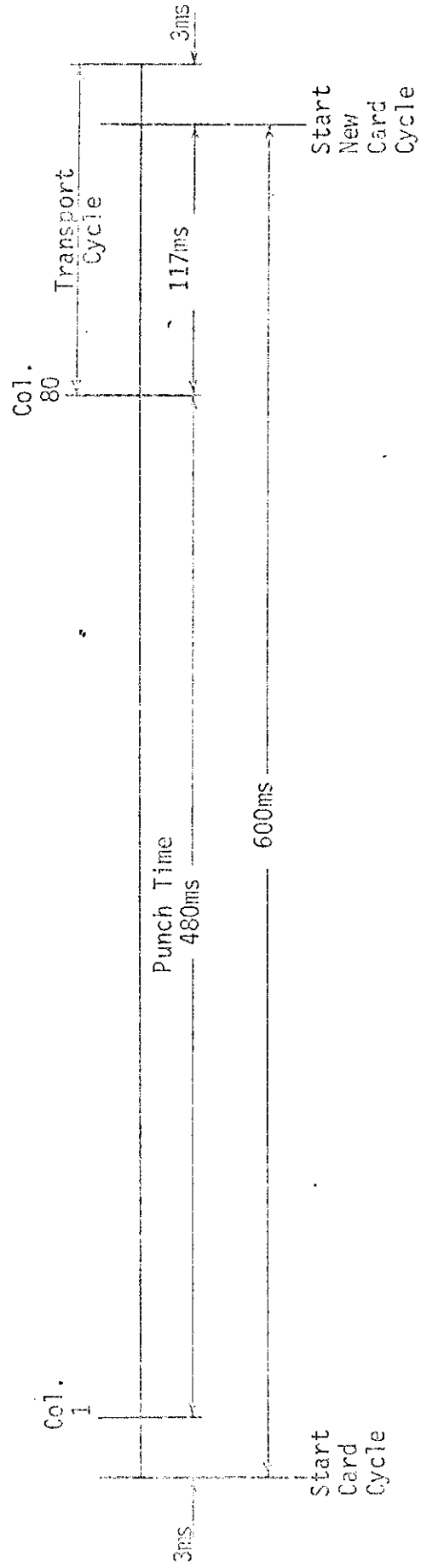
3.1.3.3.6 If the peripheral device is "Ready", and an Invalid Command is not present in the Command register, execution of a Start I/O control function will set the Feed Flip Flop. This FF provides the Feed Command to the peripheral device to begin a new card cycle. A Start I/O control function must be executed for each card. The Feed FF is reset if the Active FF is reset, or upon detection of the trailing edge of the End of Card signal from the peripheral device. For card cycle timing, refer to Figure 3.1.1.

3.1.3.3.7 Control of Inbound Data Transfers is as follows: The peripheral device pulses the Busy line once for each card column as it is read and the column data is stable on the twelve Card Data lines. The Busy line leading edge is detected and used to strobe the data into the Input Data register. This same strobe is used to set the CPU Data Request FF and the Register Full FF. The Register Full FF is reset when the micro-processor reads the Input Data Register, or when it executes a Clear Status control function. Refer to Timing Diagram 3.1.2 for inbound data transfer timing. In the event that the Busy line is pulsed for a new data transfer and the Register Full FF is already set, loading of the Input

500 cpm Read Cycle

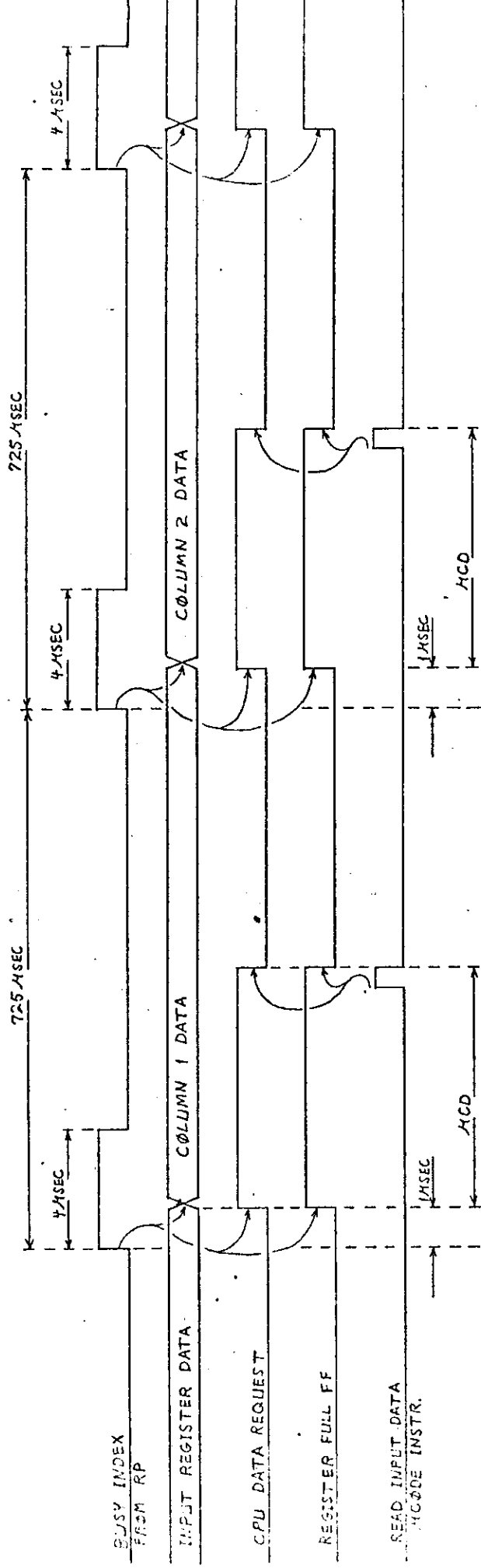


120 cpm Punch Cycle



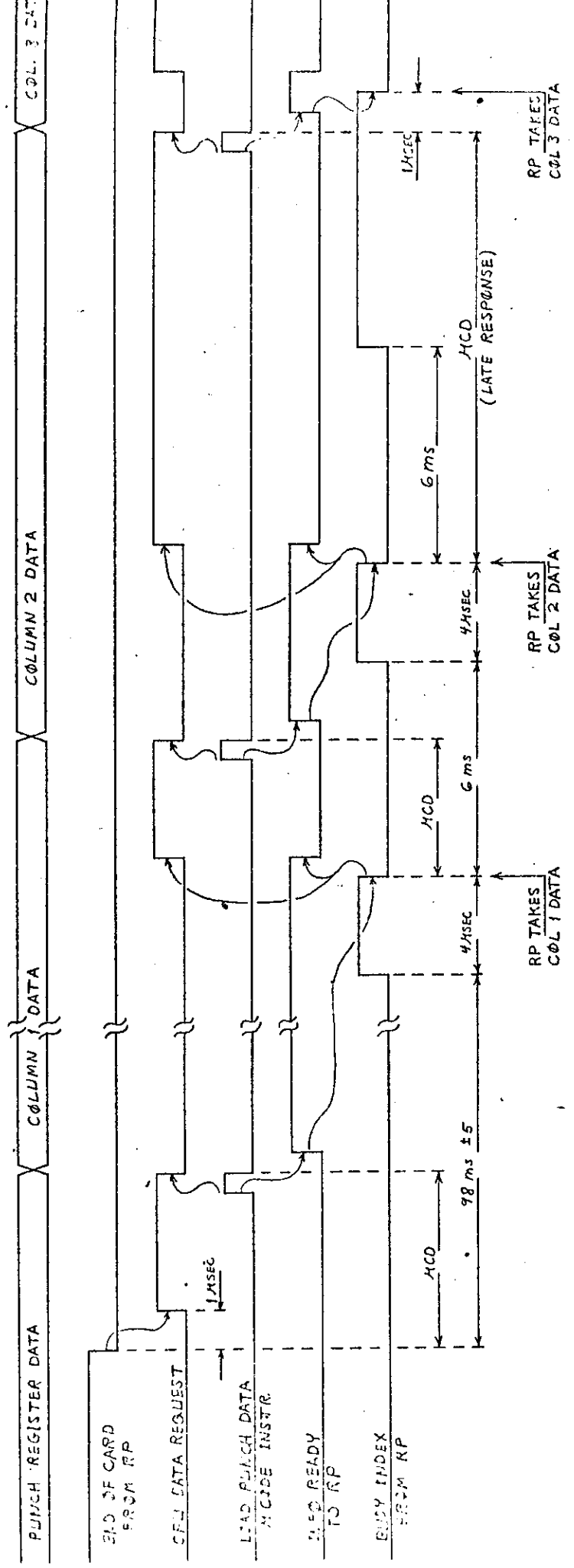
CARD CYCLE TIMING

Figure 3.1.1



NOTE: 1 μCD = MICRO CODE DEPENDENT

TIMING DIAGRAM 3.1.2
 INPUT DATA CONTROL TIMING
 MRX 30 READER/PUNCH ADAPTER
 FOR CONTROL DATA MODEL 9280
 (80 COLUMN)
 (PRELIMINARY) 6-15-72



NOTE: HCD = MICRO CODE DEPENDENT

TIMING DIAGRAM 31.3
 OUTPUT DATA CONTROL TIMING
 MAX 30 READER/PUNCH ADAPTER
 FOR CONTROL DATA MODEL 9220
 (80 COLUMN)
 (PRELIMINARY) G-15-72

Data register will be inhibited and the Overrun FF will be set, signaling a data loss error. The Overrun FF can be reset only by execution of a Clear Status control function.

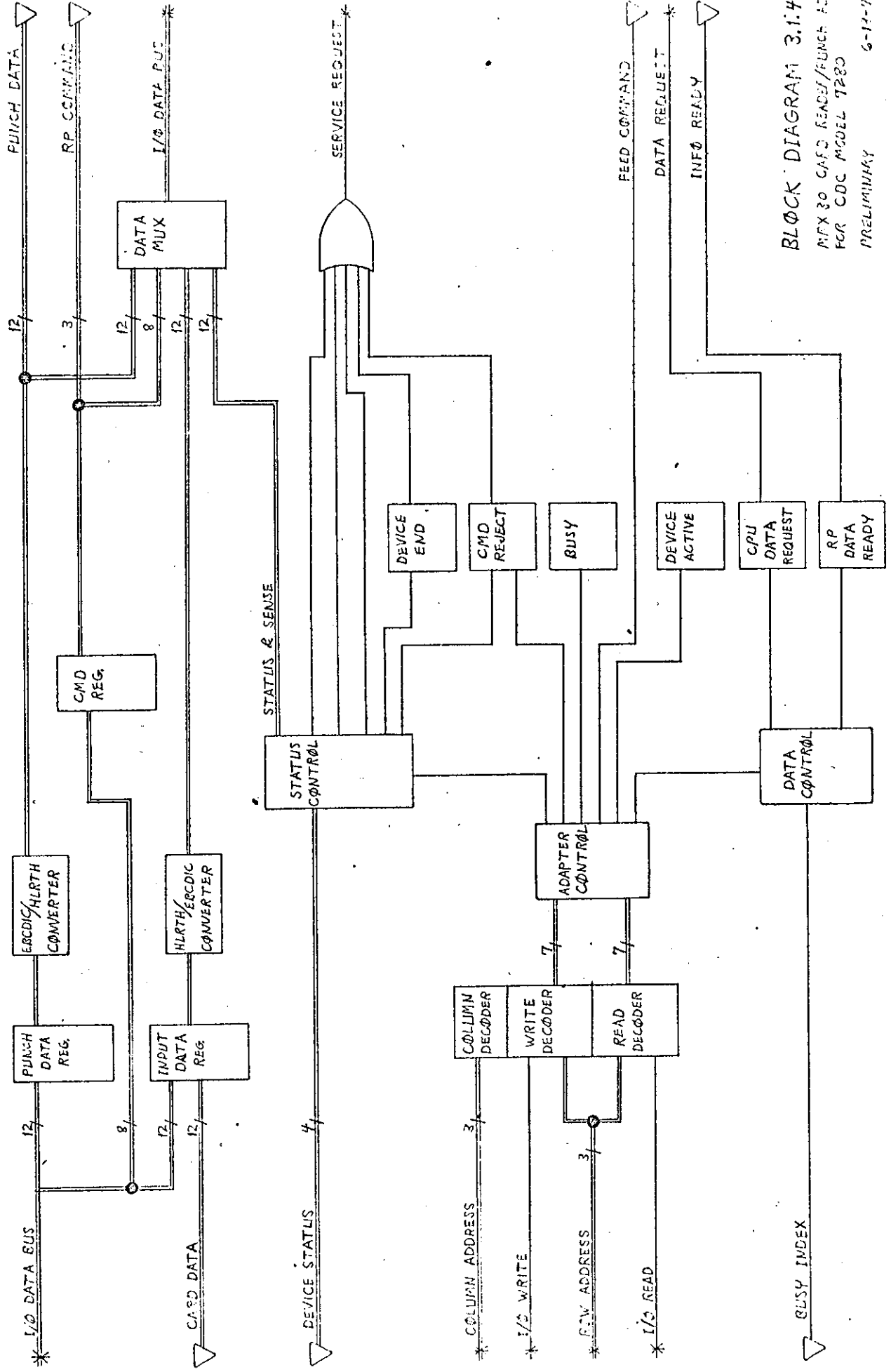
3.1.3.3.8 Control of Outbound Data Transfers is as follows: The peripheral device will set the CPU Data Request FF upon recognition of the trailing edge of the End of Card signal. The micro-processor will respond by loading the first character into the Punch Data register. The Load Punch Data strobe also resets the CPU Data Request FF and sets the Info Ready FF. The Info Ready term signals the peripheral device that Punch Data is available. The peripheral device signals the IRPABO that it has taken the Punch Data by pulsing the Busy line. The trailing edge of this pulse is detected and is used to reset the Info Ready FF, and to again set the CPU Data Request FF. The micro-processor then responds with new data and the sequence is repeated until all punch data for one card has been transferred. (Refer to Timing Diagram 3.1.3 for outbound data transfer timing.) This data transfer sequence may be terminated earlier than 80 columns, if desired, by execution of a new Start I/O (Feed) control function. Overrun cannot occur on outbound data transfers.

3.1.3.3.9 The Busy FF denotes that a peripheral device card cycle is in process. It is set by the Feed Command and is reset by detection of the End of Card (EOC) leading edge, or upon resetting of the Active FF.

3.1.3.3.10 The peripheral device raises the Ready line if it is on-line and there are no error conditions. The device raises the Status line if it is Ready, the motor is on, and cards are present in the Read Ready and Punch Ready stations. If both of these lines are true, the Ready Latch will be set. If either line is false, the Ready Latch will be reset.

3.1.3.3.11 The leading edge of the Ready Latch is detected and is used to set the Attention FF. The Attention FF is reset by the Clear Status control function. The attention signal raises the Service Request line to signal the micro-processor that the peripheral device is On-line and ready for use.

- 3.1.3.3.12 The Device End FF is set by the leading edge of the Ready Latch, or by the leading edge of the End of Card signal. It is reset by the Clear Status control function. The Device End signal denotes that the peripheral device has completed its card cycle and is available for a new Feed Command or, in combination with the Attention signal, that the device has come On-Line and is ready for use.
- 3.1.3.3.13 The peripheral device will pulse the Error Strobe if it detects a "light or dark probe error" during read mode, or if it detects an "echo check error" during punch mode. This Error Strobe line will, if pulsed, set the Data Check FF. If the IRPA80 is not in Binary Image mode this flip flop will also be set by the Hollerith Not Valid signal. The Data Check FF is reset by the Clear Status control function.
- 3.1.3.3.14 The Command Reject FF will be set by an illegal Read or Write Row address, or if a Start I/O is executed when the peripheral device is not ready, or if a Start I/O is executed when the Command Register contains an invalid command. It is reset by the Clear Status control function. The Command Reject signal inhibits the Feed Command logic and the CPU Data Request logic, and signals the micro-processor that an operational error condition exists.
- 3.1.3.3.15 The Service Request line provides the IRPA80 with the capability to signal the micro-processor that service is required. This line is raised by setting the Attention FF. Also, if the Active FF is set, this line will be raised by setting of the CPU Data Request FF, Device End FF, Overrun FF, Command Reject FF, Data Check FF, or by the Ready latch being reset.



BLOCK DIAGRAM 3.1.4
 MFX 30 GAFD READER/FUNCH READER
 FOR CDC MODEL 7280
 PRELIMINARY 6-11-72

3.2 NINETY SIX COLUMN CARD INTEGRATED READER-PUNCH ADAPTER (IRPA96)

3.2.1 GENERAL

The IRPA96 is the Integrated Adapter/Controller which contains the hardware interface and control logic required to operate the Decision Data Corp. 96 column card equipment. The models which this adapter will control are:

8610 Data Recorder	(DDC 9601)
8611 Data Recorder/Printer	(DDC 9610)
8630 1200 cpm Reader	(DDC 9630)
8633 300 cpm Reader	(DDC 9625)
8643 300/120 cpm Reader/Punch	(DDC 9635)
8653 300/120 cpm Reader/Punch/Printer	(DDC 9645)
8655 500/240 cpm Reader/Punch/Printer	(DDC 9640)
8660 1000/500 cpm MFCU	(DDC 9650)

3.2.2 DEVICE DESCRIPTION

3.2.2.1 The functions, interface, and control of all DDC models are compatible sub-sets of the DDC 9650 MFCU. Therefore, all peripheral equipment references in Section 3.2 will be in respect to this MFCU.

3.2.2.2 The DDC 9650 MFCU is a 96 column Multifunction Card Unit which consists of two input card hoppers, two read stations, two pre-punch wait stations, a collating juncture, one punch station, one print station, six output stackers, and associated mechanical and electronic control. Also, included are three "full card" buffers for the read, punch, and print stations. The MFCU interface signals are DTL/TTL compatible.

3.2.3 IRPA96 DESCRIPTION

3.2.3.1 The Ninety Six Column Integrated Reader-Punch Adapter logic is contained on two plug-in boards. This logic is divided between a "Register" board and a "Control" board. The IRPA96 interfaces to the micro-processor via the I/O micro-bus, the row and column address lines, the read/write control lines, and the service

request line. For additional information on the I/O interface, refer to Section 2.0.

3.2.3.2 IRPA96 Register Board

The Register Board contains the Command Register, Punch Data Register, 96 column/ EBCDIC logic, Read Control and Write Control decoders, and the Data Transfer Control logic.

3.2.3.2.1 The Command Register is 11 bits in length. During the I/O initiation procedure the micro-processor fetches the Command Bytes from the SIO instruction and loads it into this register. The Command is then available to the micro-processor for testing, and, normally, will remain in the Command register for the duration of one I/O operation. The bits in this register correspond to bits 5 thru 15 of the micro-bus and their meanings are: Punch, Read, Print, Print Separate Data, Upper Hopper Select, Inhibit Input Feed, Stacker Mode Control, and stacker select (three lines).

3.2.3.2.2 The Punch Data Register is the outbound data buffer during Card Punch or Print operations. It is 6 bits in length. This register is loaded by the Write Decoder "Load Punch Data" term. The 8 bit EBCDIC character is taken from bits 8 thru 15 of the micro-bus, presented to the "EBCDIC to 96 column format" converter, and the converted 6 bits are strobed into the register. The outputs of this register are connected to the peripheral interface Punch Data lines. This Punch data is also available to the Input Multiplexer, thus providing the capability for diagnostic testing of the outbound data path.

3.2.3.2.3 The peripheral device continuously maintains a character on the Input Data interface lines. Therefore buffering in the Adapter is not required. This data is present at the inputs to the "96 column format to EBCDIC" converter, and the output of this converter is available to the Input Multiplexer.

3.2.3.2.4 The Write Decoder is enabled by the I/O write term and, depending on the Row Address contents, decodes one of the following Register Load or Control terms:

- 1) Load Command Byte (register load)
- 2) Load Punch Data (register load)
- 3) Start I/O (control function)
- 4) Halt I/O (control function)
- 5) Clear Status (control function)
- 6) Reset Device (control function)
- 7) Clear Input Buffer (control function)

3.2.3.2.5 There is one Write address function and one Read address function not used by the IRPA96. These are interpreted as illegal addresses and, should the micro-processor attempt to execute one of these when the Column Address is pointing to the IRPA96, the Command Reject flag will set to indicate an error condition.

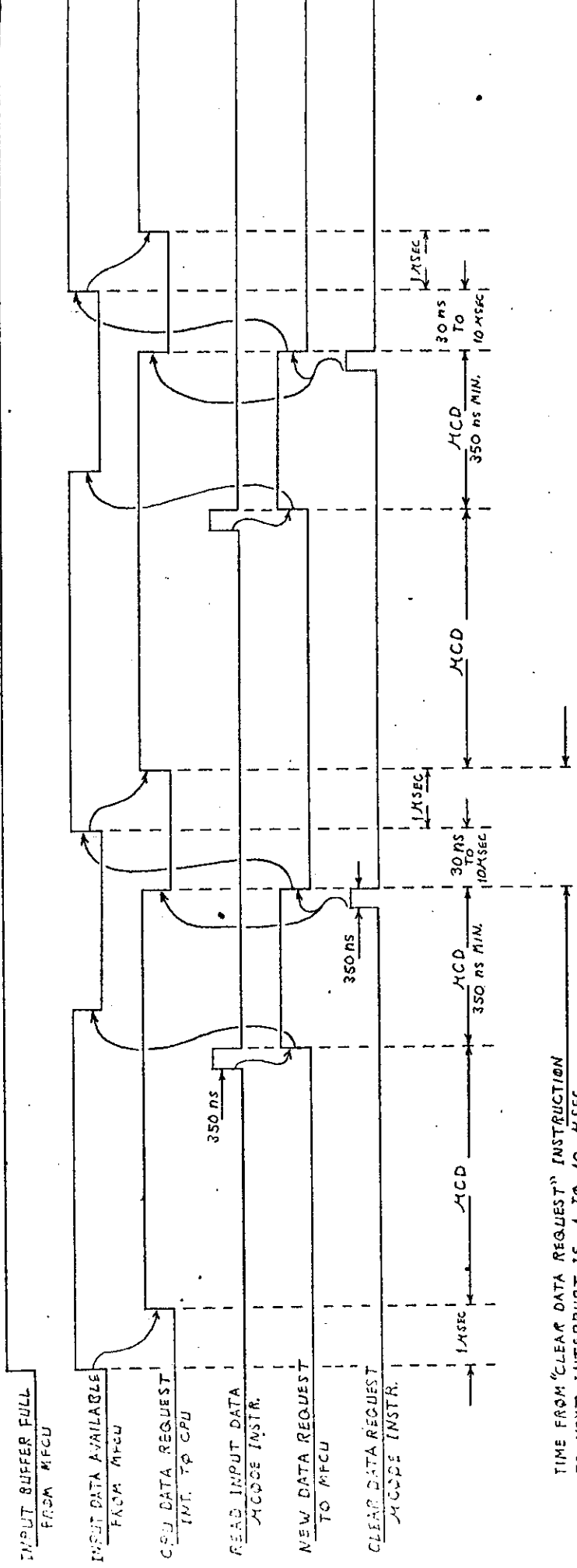
3.2.3.2.6 Control of Inbound Data Transfers is as follows: The peripheral device will raise the Input Data Available line for each character, as it becomes available. The leading edge of this signal is detected and the CPU Data Request FF is set to signal the micro-processor that data is available. When the micro-processor executes a Read Input Data micro-instruction, the CPU Data Request FF will reset and the MFCU Data Request FF will set. The MFCU Data Request signal will cause the peripheral device buffer to shift one character and, when the new character is stable, the device will again raise the Input Data Available line. This sequence will continue until all 96 characters are transferred or until the micro-processor executes a Clear Input Buffer control function. Refer to Timing Diagram 3.2:1.

3.2.3.2.7 Control of Outbound Data Transfers is as follows: The peripheral device will raise the Output Buffer Available line as it becomes ready to receive each output character. The leading edge of this signal is detected and used to set the CPU Data Request FF. When the micro-processor responds by loading the Punch register, the Load Output Buffer line will be strobed by the Adapter to signal that data is available. The peripheral device will accept the data and again raise the Output Buffer Available line. This

COLUMN 3

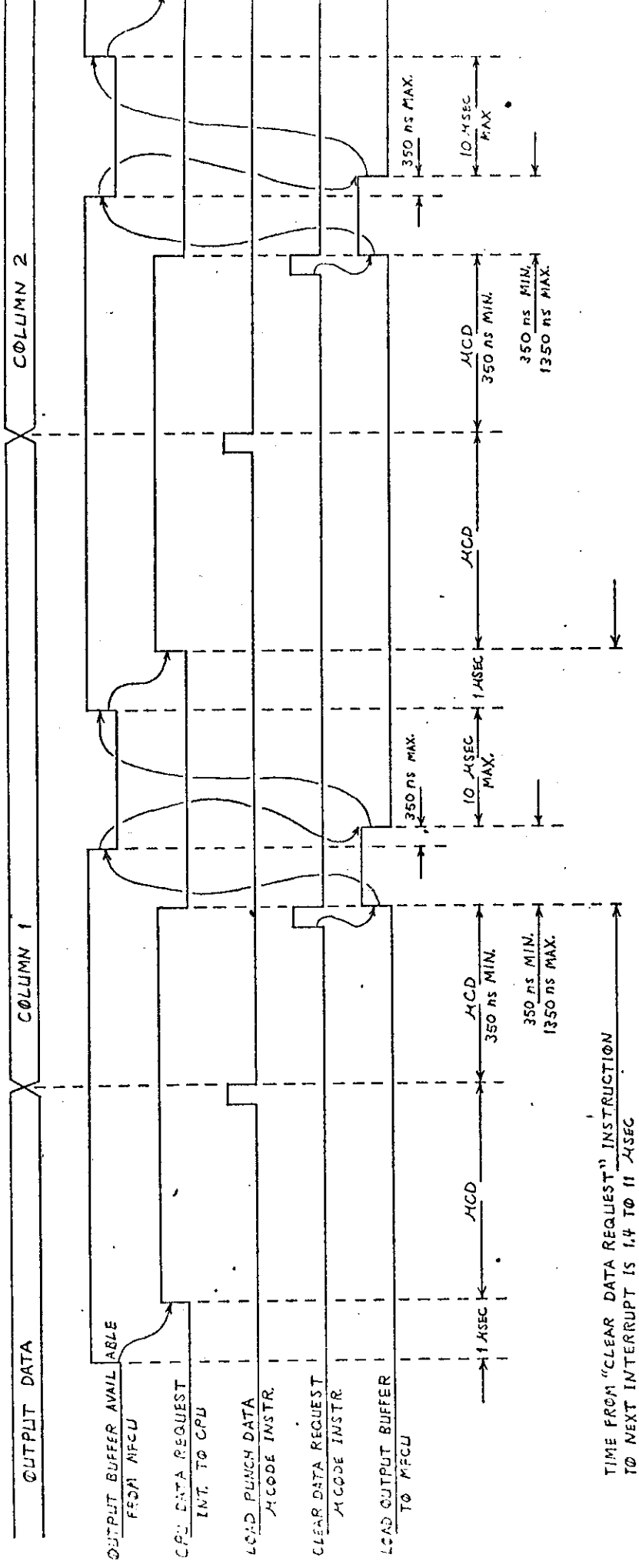
COLUMN 2

COLUMN 1



TIME FROM "CLEAR DATA REQUEST" INSTRUCTION
TO NEXT INTERRUPT IS 1 TO 10 μSEC.

NOTE: MCD = MICRO CODE DEPENDENT



TIME FROM "CLEAR DATA REQUEST" INSTRUCTION TO NEXT INTERRUPT IS 1.4 TO 11 μSEC

NOTE: MCD = MICRO CODE DEPENDENT

sequence will continue until 96 characters have been transferred, or the micro-processor executes a new Start I/O (Feed) control function. Refer to Timing Diagram 3.2.2.

3.2.3.3 IRPA96 Control Board

The IRPA96 Control Board contains the Input Multiplexer, the Data Check logic, Adapter Status logic, and the Peripheral Device Status logic.

3.2.3.3.1 The Input Multiplexer selects the information specified by the I/O Row Address and, if it is a Read operation, the output drivers are enabled, placing the information on the I/O micro-bus. The selected information is one of the following:

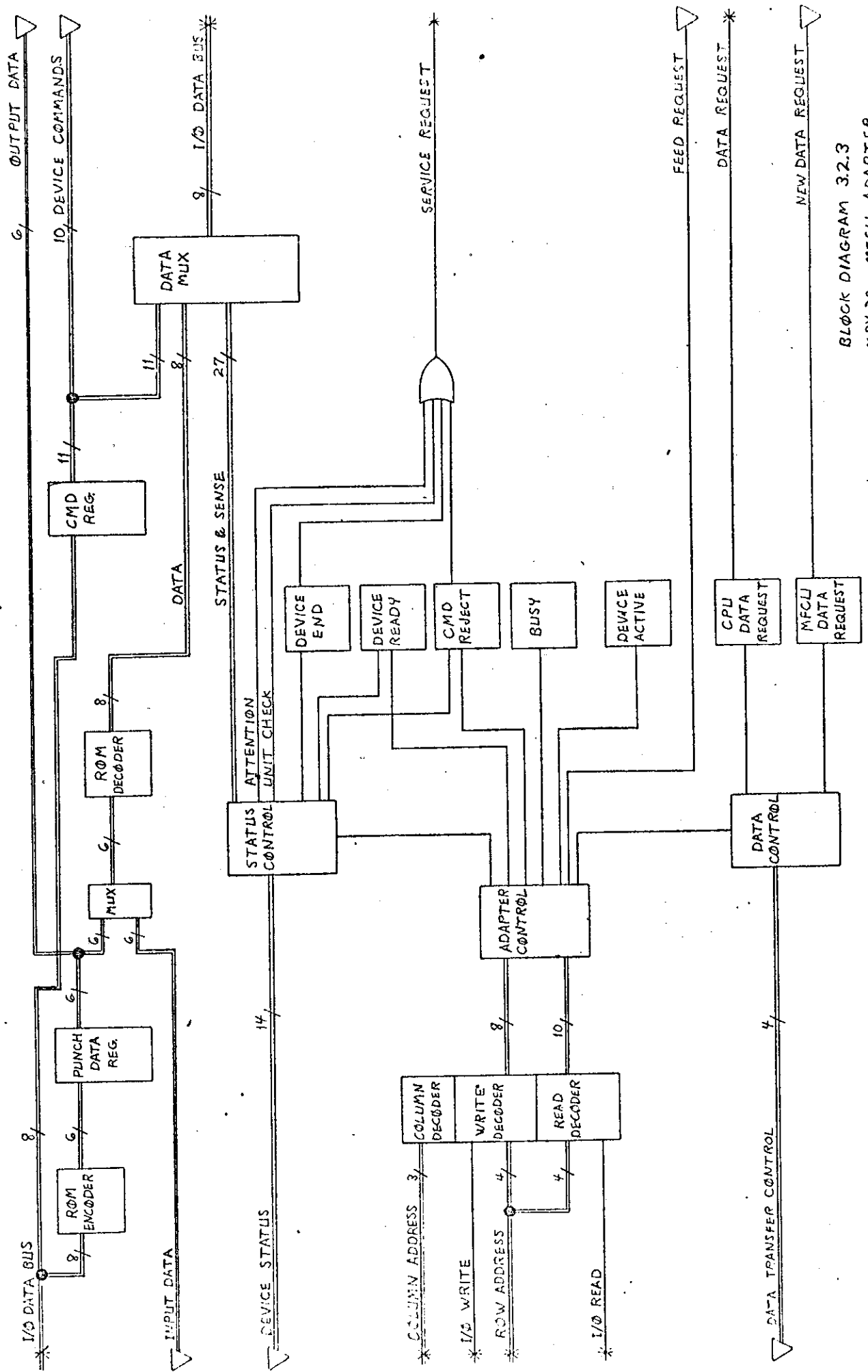
- 1) Input Data (sourced from the input converter)
- 2) Punch Data (sourced from the output data lines)
- 3) Command Byte (sourced from the Command register)
- 4) Status (Attention, Busy, Device End, Unit Check, Request In)
- 5) Sense 1 (Command Reject, Intervention, Data Check, Not Ready, Upper Input Check, Lower Input Check, Output Check, Stacker full, Lower Hopper Empty, Upper Hopper Empty, CPU Data Request)
- 6) Sense 2 (Input Data Available, Output Buffer Available, Print Buffer Available, Card In Lower Read Wait Station, Card In Lower Pre-punch Wait Station, Ready for Upper Command, Ready for Lower Command, End of File, Punch Data Check, Read Data Check)

3.2.3.3.2 The Active FF is set if the peripheral device is "Ready" and the micro-processor executes a Start I/O to the IRPA96. It is reset by a Halt I/O. When the Active FF is reset, all service requests are inhibited except "Attention". Also, all data transfers between the IRPA96 and the peripheral device are inhibited. However, all control functions and data transfers between the IRPA96 and the micro-processor are allowed in order to provide for diagnostics, adapter testing, and I/O initiation/termination procedures.

- 3.2.3.3.3 If the peripheral device is "Ready", execution of a Start I/O will set the Feed FF. This flip flop provides the Feed Request signal to the peripheral device to begin a new card cycle. A Start I/O must be executed for each card. The Feed FF is reset by the Input Buffer Full or Output Buffer Available signals.
- 3.2.3.3.4 The Busy FF denotes that a peripheral device card cycle is in process. It is set by the Feed Request signal and is reset by a Halt I/O or by the trailing edge of the Ready for Command signal.
- 3.2.3.3.5 The peripheral device raises the Ready line if it is on-line and there are no error conditions. If this line is true, the Ready Latch will be set. If it is false, the Ready latch will be reset.
- 3.2.3.3.6 The leading edge of the Ready latch is detected and is used to set the Attention FF. The Attention FF is reset by the Clear Status control function. The Attention signal raises the Service Request line to signal the micro-processor that the peripheral device is On-Line and ready for use.
- 3.2.3.3.7 The Device End FF is set by the trailing edge of the Busy signal, or by the leading edge of the Ready latch. It is reset by the Clear Status control function. The Device End signal denotes that the peripheral device has completed its card cycle and is available for a new Feed Command or, in combination with the Attention signal, that the device has come on-line and is ready for use.
- 3.2.3.3.8 The Data Check signal will go true during a Read operation, if the peripheral device detects a read error. It will go true during a Punch operation, if the peripheral device detects a difference between the punch data and the data read at the Post-Punch Read station.

3.2.3.3.9 The Command Reject FF will be set by an illegal Read or Write Row address. It will also be set by the Start I/O if one or more of the following conditions exist: The Command register contains an illegal command, or upper hopper is selected and the device is not Ready for Upper Command, or upper hopper is not selected and the device is not Ready for Lower Command, or the device is busy, or the device is not ready. The Command Reject FF is reset by the Clear Status control function.

3.2.3.3.10 The Service Request line provides the IRPA96 with the capability to signal the micro-processor that service is required. This line is raised by setting the Attention FF. Also, if the Active FF is set, this line will be raised by setting of the CPU Data Request FF, Device End FF, Command Reject FF, Data Check signal, both hoppers empty, or by the Ready latch being reset.



BLOCK DIAGRAM 3.2.3

MRX 30 MFCLU ADAPTER

PRELIMINARY

5-16-72

3.3 INTEGRATED LINE PRINTER ADAPTER (ILPA)

3.3.1 GENERAL

The ILPA is the Integrated Adapter/Controller which contains the hardware interface and control logic required to operate Data Products Models 2440 and 2470 Line Printers.

3.3.2 DEVICE DESCRIPTION

3.3.2.1 The Model 2440/2470 Line Printers contain the electronics and mechanical parts required to print lines of 132 characters on multiple part fanfold paper and advance the paper to the next line to be printed. The control logic required to operate the mechanism electronics and a 132 character buffer register are included. The printing speed (with the standard 64 characters) is 1200 LPM with 68 characters per line, or 670 LPM with the full 132 characters per line. The interface to these printers is DTL/TTL compatible.

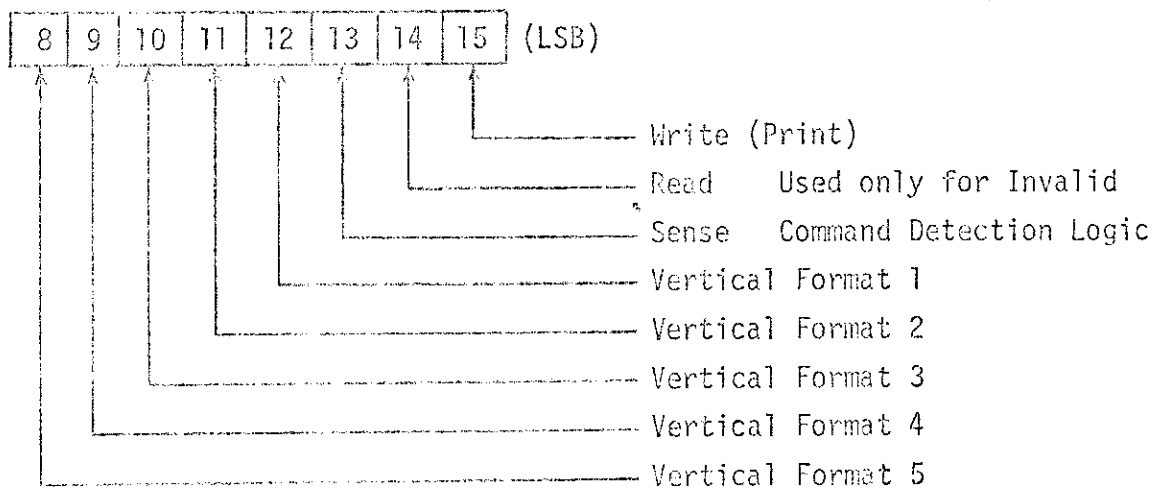
3.3.3 ILPA DESCRIPTION

3.3.3.1 The Integrated Line Printer Adapter logic is contained on two plug-in boards. This logic is divided between a "Register" board and a "Control" board. The ILPA interfaces to the micro-processor via the I/O micro-bus, the row and column address lines, the read/write control lines, and the service request line. For additional information on the I/O interface, refer to Section 2.0

3.3.3.2 ILPA Register Board

The ILPA Register Board contains the Command Register, Invalid Command Detection Logic, Print Data Register, Data Transfer Control Logic, Vertical Format Control Logic, and the Peripheral Device Status Logic. Refer to Logic Diagram ILPA or Block Diagram 3.3.2.

3.3.3.2.1 The Command Register is 8 bits in length. During the I/O initiation procedure, the micro-processor fetches the command byte from the SIO instruction and loads it into this register. The command byte is then available to the micro-processor for testing and, normally, will remain in this register for the duration of one I/O operation. The bits in this register correspond to bits 8 through 15 of the micro-bus and their meanings are assigned as follows:



3.3.3.2.2 Each of the Vertical Format Control Codes required for these line printers is one less, in binary value, than the corresponding codes used in standard software. The vertical format bits from the command register outputs are applied to VF code decrement logic which decrements the binary value, thus correcting this discrepancy. The output of this logic are available to the LP Data Multiplexer.

3.3.3.2.3 The Invalid Command Detection logic tests the four least significant bits of the command register. Three of the sixteen possible combinations will be sensed as an invalid command. These are:

1. 0000 Invalid
2. 1000 Transfer in Channel
3. 1100 Read Backward

The invalid command term is one of the inputs to the command reject logic on the control board. However, the invalid command term will not cause a "Command Reject" to occur unless it is present when the micro-processor issues an "Execute Vertical Format" control function.

3.3.3.2.4 The Print Data Register is a single character buffer for the print data. It is 8 bits in length, and they correspond to bits 8 through 15 of the micro-bus. The outputs of this register are multiplexed with the vertical format lines of the command register. The contents of one of the registers is always present on the LP data lines to the peripheral device. This resultant data is also available to the Input Multiplexor, thus providing the capability for diagnostic testing of the data path.

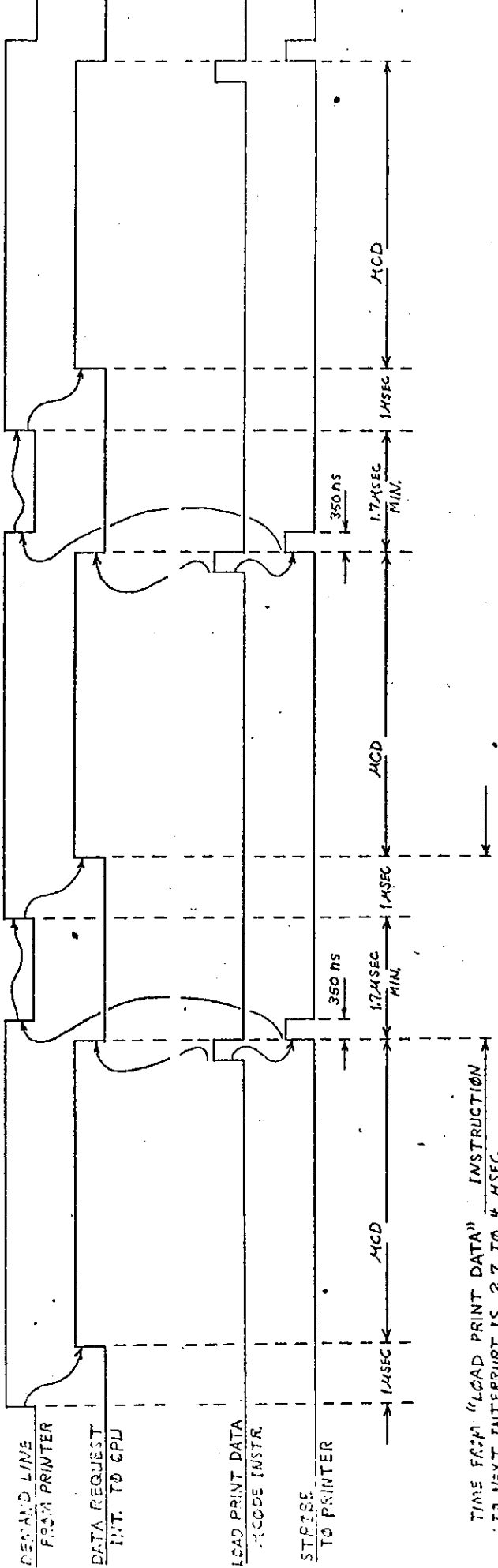
3.3.3.2.5 Control of Data Transfers is as follows. The peripheral device raises the demand line to signal that its buffer is ready for a character. The demand line sets the CPU Data Request FF. The micro-processor will respond by loading a character into the print data register. The load print data strobe resets the CPU Data Request FF and sets the data strobe latch. The

PRINT DATA

CHAR. 1

CHAR. 2

CHAR. 3



TIME FROM "LOAD PRINT DATA" INSTRUCTION TO NEXT INTERRUPT IS 2.7 TO 4 μSEC.

NOTE: μCD = MICRO-CODE DEPENDENT

TIMING DIAGRAM 3.3.1
OUTPUT DATA CONTROL TIMING

MRX 30 LINE PRINTER ADAPTER
DATA PRODUCTS MODELS 2440 & 2470
PRELIMINARY 6-5-72

data strobe signals the peripheral device that the character is on the LP data lines. The device loads the character into its buffer and again raises the demand line. This sequence is repeated until all characters for one printer line have been transferred, or an "Execute Vertical Format" control function is issued. (Refer to Timing Diagram 3.3.1.)

3.3.3.2.6 Vertical Format control of the peripheral device is accomplished as follows. The micro-processor issues an "Execute Vertical Format" control function to the ILPA. This sets the Vertical Format FF which, in turn, switches the LP Multiplexor and raises the Vertical Format Control line. This line signals the peripheral device that the character on the LP Data Lines is a Vertical Format Control character from the command register. The actual character transfer follows the same sequence as that described in Paragraph 3.3.3.2.5. The peripheral device stores the Vertical Format Control character but completes the printing of all data characters previously received, prior to acting upon the control character.

3.3.3.2.7 The Busy signal denotes that the peripheral device is in the process of either printing a line of characters, or moving the paper under Vertical Format Control. The Busy signal is TRUE if the demand latch is set, or the Vertical Format FF is set.

3.3.3.2.8 The Ready latch is set if the peripheral device is "Ready" and "On-line." It is reset if either of these is not true, or if the device detects a forms jam, end of forms, or format tape missing.

3.3.3.2.9 The leading edge of the Ready latch is detected and is used to set the Attention FF. The Attention FF is reset by the clear status control function. The device end signal denotes that the peripheral device has completed its print and paper advance cycle or, in combination with the attention signal, that the device has come on-line and is ready for use.

3.3.3.3 ILPA Control Board

The ILPA Control Board contains the write control decoder, the input multiplexor, and the active and command reject FF's.

3.3.3.3.1 The I/O Row Address is presented to the write decoder and the Input Multiplexor. The Input Multiplexor selects the information specified by the I/O Row Address and, if it is a read operation, the output drives are enabled, placing the information on the I/O micro-bus. The selected information is one of the following:

1. LP Data (sourced from the LP Data mux.)
2. Command Byte (sourced from the Command Register.)
3. Status (Attention, Busy, Device End, Unit Check, Request In)
4. Sense (Command Reject, Intervention, Data Check, Not Ready, On-line, Forms Jam, End of Forms, Format Tape Missing, Vertical Format, Demand, CPU Data Request)

3.3.3.3.2 The Write Decoder is enabled by the I/O Write Term and, depending on the row address contents, decodes one of the following register load or control function terms.

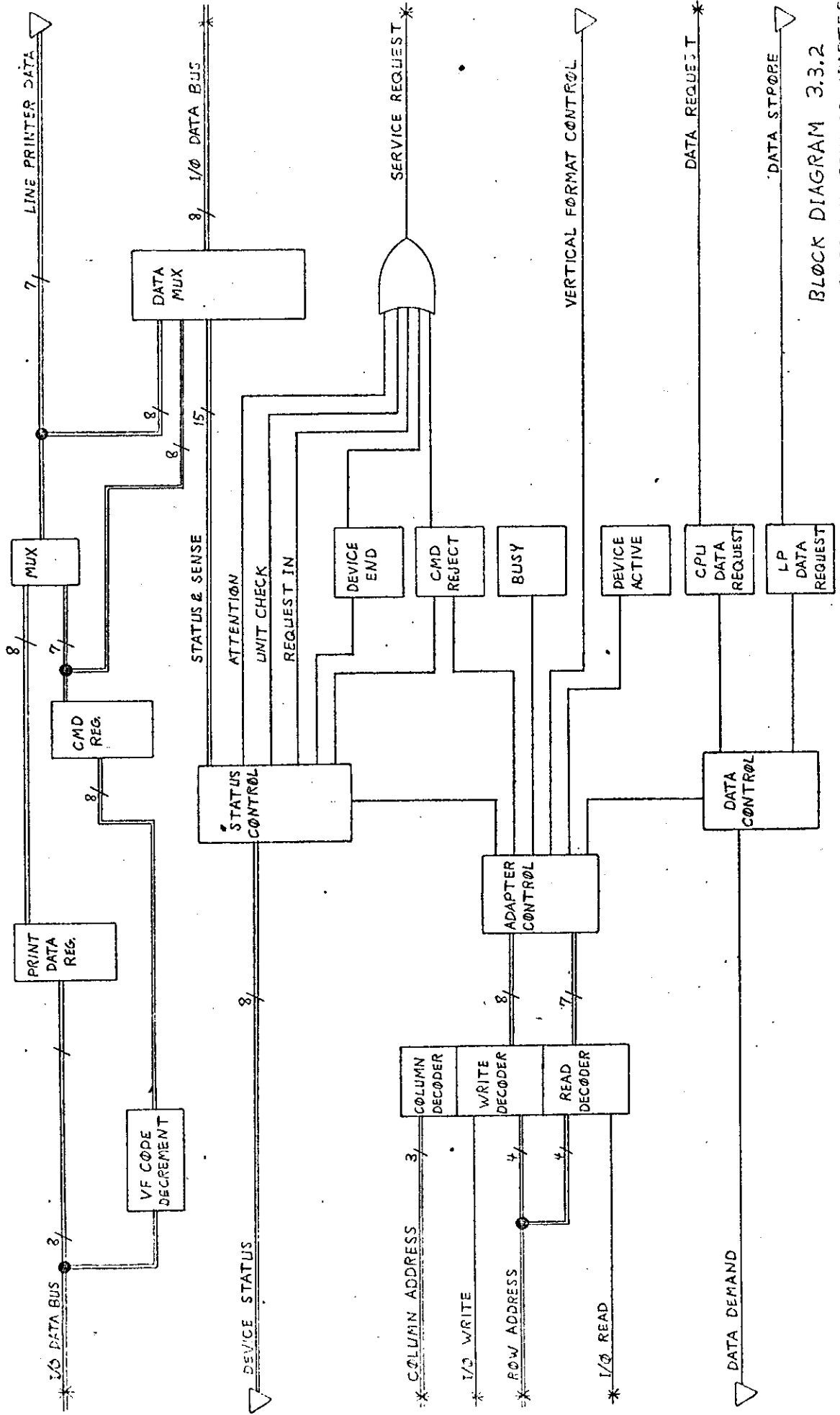
1. Load Print Data (register load)
2. Load Command Byte (register load)
3. Start I/O (control function)
4. Halt I/O (control function)
5. Clear Status (control function)
6. Reset Device (control function)
7. Execute Vertical Format (control function)

3.3.3.3.3 There is one write address function and four read address functions not used by the ILPA. These are interpreted as illegal addresses and should the micro-processor attempt to execute one of these when the column address is pointing to the ILPA, the command reject flag will be set to indicate an error condition.

3.3.3.3.4 The Active FF is set if the peripheral device is "Ready" and the micro-processor issues a Start I/O control function to the ILPA. It is reset by the Halt I/O control function. When the Active FF is reset, all service requests are inhibited except "Attention." Also, all data transfers between the ILPA and the peripheral device are inhibited. However, all control functions and data transfers between the ILPA and the micro-processor are

allowed in order to provide for diagnostics, adapter testing, and I/O initiation/termination procedures.

- 3.3.3.3.5 The Command Reject FF will be set by an illegal read or write row address. It will also be set by an "Execute Vertical Format" control function if it occurs when the command register contains an invalid command, or by a "Load Print Data" if this occurs when the peripheral device is either busy or not ready. The Command Reject FF is reset by the clear status control function. The command reject signal inhibits the CPU data request logic, and signals the micro-processor that an operational error condition exists.
- 3.3.3.3.5 The Service Request line provides the ILPA with the capability to signal the micro-processor that service is required. This line is raised by setting the Attention FF. Also, if the Active FF is set, this line will be raised by setting of the CPU Data Request FF, Device End FF, Command Reject FF or by the ready latch being reset.



BLOCK DIAGRAM 3.3.2
 MRX 30 LINE PRINTER ADAPTER
 FOR MODELS 2440 & 2470
 PRELIMINARY 5-24-72

3.4 CONSOLE ADAPTER

The console adapter provides the interface between the main processor and the console CRT Display Terminal. The console is the means by which the operator can control the operation of the CPU.

3.4.1 DEVICE CONFIGURATION

The adapter will interface with a CRT Terminal at a speed of 9600 BPS or a 1240 Terminal at a speed of 1200 BPS.

3.4.2 THEORY OF OPERATION

The console adapter receives all data and command and transmits data and status from and to the CPU via the I/O micro-bus. The adapter receives characters for the CPU in parallel and disassembles them and sends them serial to the console. The console adapter generates the bit clock for receiving and transmitting 10 bit start/stop characters.

3.4.3 REGISTER USAGE

3.4.3.1 Register Addressing

Address	Read	Write
0	Data	Data
1	Status	Command
2		
3		
4		
5		
6		
7		

3.4.3.2 Data Read

Bit

0

1

2

3

4

5

6

7

8 MS Data Bit

9

10

11

12

13

14

15 LS Data Bit



3.4.3.3 Data Write

. Bit

0

1

2

3

4

5

6

7

8 MS Data Bit

9

10

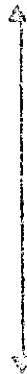
11

12

13

14

15 LS Data Bit



3.4.3.4 Status

0

1

2

3

4

5

6

7

8

3.4.3.4 (Continued)

9 Data Set Ready

10 Carrier Detect

11 Clear to Send

12 VRC Error

13 Stop Bit Error

14 Lost Data

15 Receive Break

3.4.3.5 Command

Bit

0

1

2

3

4

5

6

7

8

9

10

11 Data Terminal Ready

12 Request to Send

13 Write Request

14 Pad

15 Break

3.5 INTEGRATED COMMUNICATIONS ADAPTER

The Integrated Communications Adapter (ICA) is a 16-line multiplexed adapter which transmits and receives information from and to the main processor via the I/O micro-bus. The information flow between the ICA and the terminals is accomplished by leased common-carriers private line facilities, common-carrier switched facilities or equivalent privately owned communications facilities. The ICA is capable of handling any combination of start/stop (asynchronous) lines at speeds up to 1200 BPS, binary synchronous lines (BSC) at speeds up to 9600 BPS and data entry keyboard/display at speeds up to 1200 BPS or one 50KB binary synchronous line.

3.5.1 DEVICE CONFIGURATIONS

The communication terminals that the ICA will support are:

- Memorex 1240/1280 Communication Terminal at speed of 110 BPS, 300 BPS, 150 BPS, 600 BPS, and 1200 BPS.
- IBM 2740 Communication Terminal at 134.5 BPS and 600 BPS.
- IBM 2741 Communication Terminal at a speed of 134.5 BPS.
- Teletypewriter Terminal (Model 33/35/37) at speeds of 110 BPS and 150 BPS.
- Binary synchronous communications terminals at speeds up to 9600 BPS.
- Data entry keyboard at speeds of 110 BPS, 150 BPS, 300 BPS, 600 BPS, and 1200 BPS.

3.5.2 THEORY OF OPERATION

The ICA is divided up into four blocks. They are the control, CRC/LRC/character decode, clock and the line adapter.

The control interfaces with the I/O micro-bus and the line adapters. It also controls the CRC/LRC and character decode. Whenever a line adapter has status or data, a priority flag will be set. This flag is coded by the control to indicate a line adapter address. The control will then set a request for the main processor. The processor will read the address register to determine the line adapter address along with the type of interrupt. Once this is determined the processor will read the data or status register.

When the processor has data or a command for a line adapter, the address is written into the address register, then the data or command is written into the appropriate line adapter. The CRC/LRC/character decode unit accumulates the block check characters or character from the data being sent or received. It also decodes all the data link control characters.

The clock generates all the async bit timing for the LA's.

There are four types of line adapters:

- Asynchronous
- Binary Synchronous
- 50 K B binary synchronous
- Data Entry

All the line adapters have EIA interfaces.

3.5.3 REGISTER USAGE


3.5.3.1 Register Addressing

Address	Read	Write
0	Data/Status	Data/cmd
1	Address	Address
2		
3		
4		
5		
6		
7		

3.5.3.2 Write Address

Bit

0	cmd = 0	Data = 1
1	cmd 1 = 0	cmd 2 = 1
2	Data Character = 0	Control Character = 1
3	Dial Digit = 1	
4		
5		
6		
7		
8		
9		
10	MS Address Bit	
11		
12		
13		
14		
15	LS Address Bit	



3.5.3.3 Read Address

Bit

0 Data = 0 Status = 1

1 sol. status = 0 unsol. status = 1

2

3

4

5

6

7

8

9

10 MS Address Bit

11

12

13

14

15 LS Address Bit



3.5.3.4 Data/Command Word

Bit

0

1

2

3

4

5

6

7

8

9

10

11

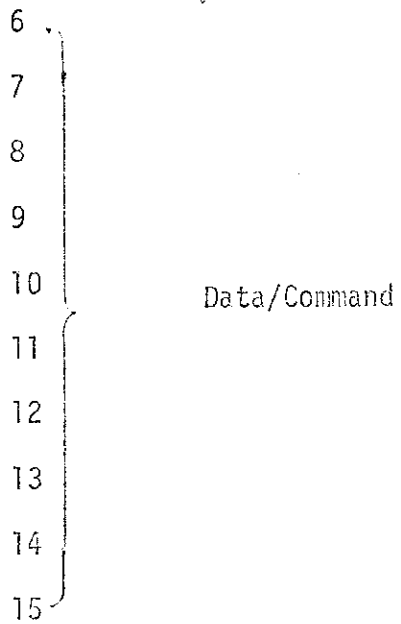
12

13

14

15

Data/Command



3.5.3.4.1 Data Bit Configuration

Bit

8 MS Data Bit

9

10

11

12

13

14

15 LS Data Bit



3.5.3.4.2 Command 1 Bit Configuration

Bit

1

2

3

4

5

6

7

8

9 Data Terminal Ready

10 Request to Send

11 Reset Sync

12 Write Request

13 Call Request

14 Pad

15 Break

3.5.3.4.3 Command 2 Bit Configuration

Bit

1

2

3

4

5 EOT Search

6 Return Unsolicited Status

7 Return Solicited Status

8 Clear Link Command

9 Odd/Error Parity

10 Parity

11 MSB

12

13 LSD

Character Speed

14 MSB

15 LSD

Character Length

3.5.3.4.3.1 Character Speed and Character Length

Character Speed

Character Length

0

0

1 1200 BPS

1 9 Bits

2 600 BPS

2 10 Bits

3 300 BPS

3 11 Bits

4 150 BPS

5 134.5 BPS

6 110 BPS

7

3.5.3.5 Data/Status Word

Bit

0

1

2

3

4

5

6

7

8

9

10

11


12

13

14

15

Data/Status



3.5.3.5.1 Data Bit Configuration

Bit

8 MS Data Bit

9

10

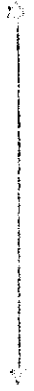
11

12

13

14

15 LS Data Bit



3.5 .3.5.2 Unsolicited Status Bit Configuration

Bit

0

1

2

3

4

5

6

7 Data Set Ready

8 Carrier Detect

9 Clear to Send

10 VRC Error

11 Stop Bit Error

12 Lost Data

13 Ring

14 Abandon Call

15 Receive Break

3.5.3.5.3 Solicited Status Bit Configuration

Bit

0

1

2

3

4

5

6

7

8

9

10

11 Transparent Mode

12 Data Terminal Ready

13 Request to Send

14 EOT Search

15 Synchronization Search

3.6 SELECTOR CHANNEL

The 7100 Selector Channel Adapter is identical to the IBM 360/370 Selector Channel. Transfer rate for data is an excess of 60 K BPS which is accomplished by the direct access to memory. The adaptor design uses the micro sequencer approach. This means that all hand shaking such as raising tag lines, monitoring lines and comparing addresses is achieved by the hardware.

3.6.1 DEVICE CONFIGURATION

The channel can interface with:

a. Line Printers

1. 5120 Mod 6, 600 LPM Printer
2. 5120 Mod 12, 1200 LPM Printer

b. Tape Drives

1. 3237-11 800 BPI, 30 KB Controller Drive, 1 max.
2. 3237-13 1600 BPI, 60 KB Controller Drive, 1 max.
3. 3237-21 800 BPI 30 KB Drive, 3 max. w/3237-11
4. 3237-22 1600 BPI 60 KB Drive, 3 max. w/3237-12

3.6.2 THEORY OF OPERATION

The adapter has a 24 bit micro control word which is used to control tag and data lines and sense for sequencer status, channel status and tag conditions. In order to initiate a sequence the main processor loads the command and address register. If the sequence is a data transfer, the processor will load the memory address register and word count register.

The adapter takes over from there. The next time the main processor gets a request from the selector adapter it will mean either the sequence is complete or it received bad status from the channel or the adapter received a tag or bus error.

The processor will read the status register to identify the status.

The micro sequencer consists of 256 words by 24 bits of ROM.

3.6.3 MICRO SEQUENCER WORD

2	2	3	3	5	1	8
TAG	CS	C	PORT	SENSE	R	BRANCH

3.6.3.1 Bus Out

- 0 HOP
- 1 Set Hold Out/Select Out
- 2 Reset Hold Out/Select Out
- 3 Set Suppress Out
- 4 Reset Suppress Out
- 5
- 6
- 7

3.6.3.2 Bus On Gating

- 0 Gate Status to Reg.
- 1 Gate Address to Reg.
- 2 Gate Data to Upper DMA Reg.
- 3 Gate Data to Lower DMA Reg.
- 4 Set Request Service
- 5
- 6
- 7

3.6.3.3 DMA In

- 0 Store SRB
- 1 Gate SRB
- 2 Decrement Byte Count
- 3
- 4
- 5
- 6
- 7

3.6.3.4 DMA Out

- 0 Data
- 1 Address
- 2 Command
- 3 Gate DMA Data to Bus Out Reg.
- 4
- 5
- 6
- 7

3.6.3.5 Tag

- 0 NOP
- 1 Address Out
- 2 Command Out
- 3 Service Out

3.6.3.6 Port

- 0 NOP
- 1 Set Operational Out
- 2 Reset Operational Out
- 3
- 4 Read
- 5 Low Write
- 6 High Write
- 7 Full Write

3.6.3.7 Sense

- 0 Count Complete
- 1 Tag Error
- 2 Address Compare
- 3 Parity Error
- 4 Status In
- 5 Address In
- 6 Service In
- 7 Suppress Out
- 8 Chaining
- 9 Write
- 10 Operational In
- 11 Select In
- 12
- 13
- 14
- 15
- 16

3.6.4 REGISTER ADDRESSING

	Write	Read
0	Line Address Out	Line Address In
1	Command Memory Load	Status
2	Memory Count Load	Memory Count Read
3	DMA Memory Address	
4		
5		
6		
7		

3.6.4.1 DMA Memory Address

Bit

0 MS DMA Mem Add

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15 LS DMA Mem Add



3.6.4.2 Mem Count Load/Read

Bit

0 MS Count Bit

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15 LS Count Bit



3.6.4.3 Status Register

· Bit

0 MS Status Bit

1

2

3

4

5

6

7 LS Status Bit

8 Initial Selection Seq. Error

9 Incorrect Length Transferred

10 Channel Failure

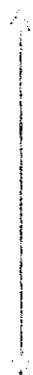
11 Not Used

12 Control Check

13 Transmission Check

14 Invalid Command

15 Not Used



3.6.4.4 Command Register

Bit

0 Chaining = 1

1 Suppress Length Indicator = 1

2 No Data Transfer = 1

3

4

5

6

7

8 MS Command Bit

9

10

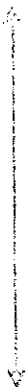
11

12

13

14

15 LS Command Bit



3.6.4.5 Line Address Out

Bit

0 Write = 1

1

2

3

4

5

6

7

Not Used

8 MS Address Bit

9

10

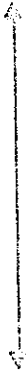
11

12

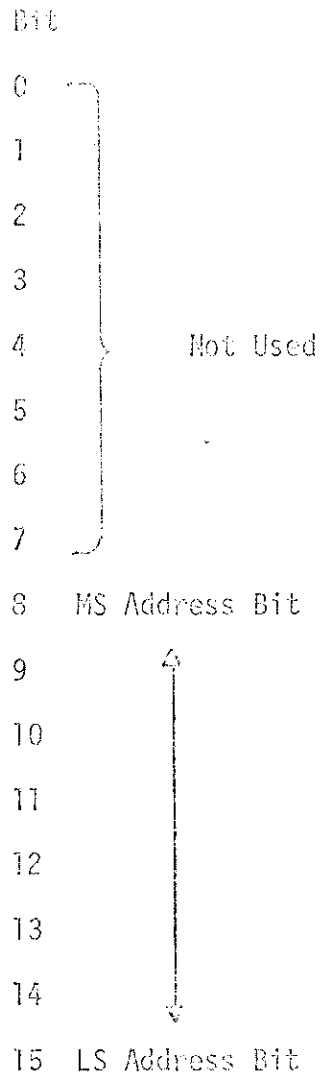
13

14

15 LS Address Bit



3.6.4.6 Line Address In



MEMOREX 7100

PHYSICAL DESCRIPTION

M. GREGORY

7/19/72

MEMOREX CONFIDENTIAL

PHYSICAL DESCRIPTION

This section will describe the physical attributes of the 7100 processor in general without regard to its applications other than as a programmable processor or a non-programmable controller.

The 7100 processor is available as a programmable processor or as a controller. Externally the two systems look very similar. Figure 1 shows the 7100 in a controller configuration while Figure 2 shows it in a processor configuration. The major external difference is in the removal of the desk, keyboard and CRT screen from the programmable processor to make it a controller.

The major physical attributes of the 7100 processor are listed in Table 1.

The 7100 processor dimensions are illustrated in Figure 3.

TABLE 1 7100 PHYSICAL CHARACTERISTICS

Dimensions (Fig 3) - inches	
width	30
depth	43
height	48
Service clearance - inches	
front	30
rear	30
right	30
left	30
Maximum weight *	2000 lb.
Maximum heat Output/hr *	15,000 BTU
Air Flow	500 CFM
Electrical requirements	
Voltage	208/230 \pm 10%
Frequency	60 \pm 0.5 Hz
KVA*	2.5
Phases	3
Operating Environment	
Temperature	60-100 $^{\circ}$ F
Relative Humidity	8-80%
Maximum Wet Bulb	78 $^{\circ}$ F

* Values for system consisting of a 7100 processor, 300 LPM printer, 8660 MFCU and one Trident file.

Non-Operating Environment

Temperature

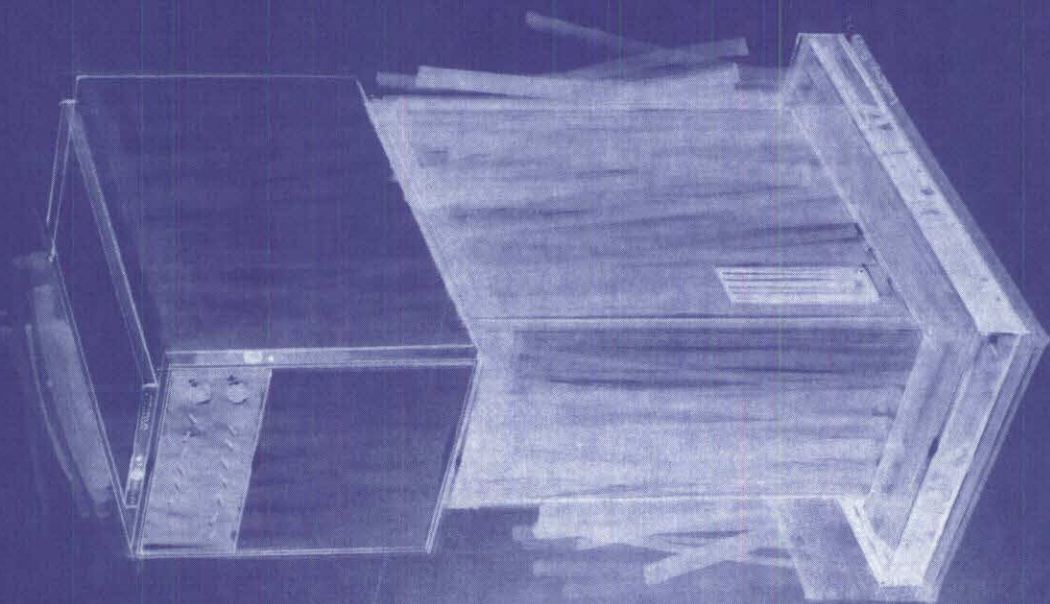
50-110^o

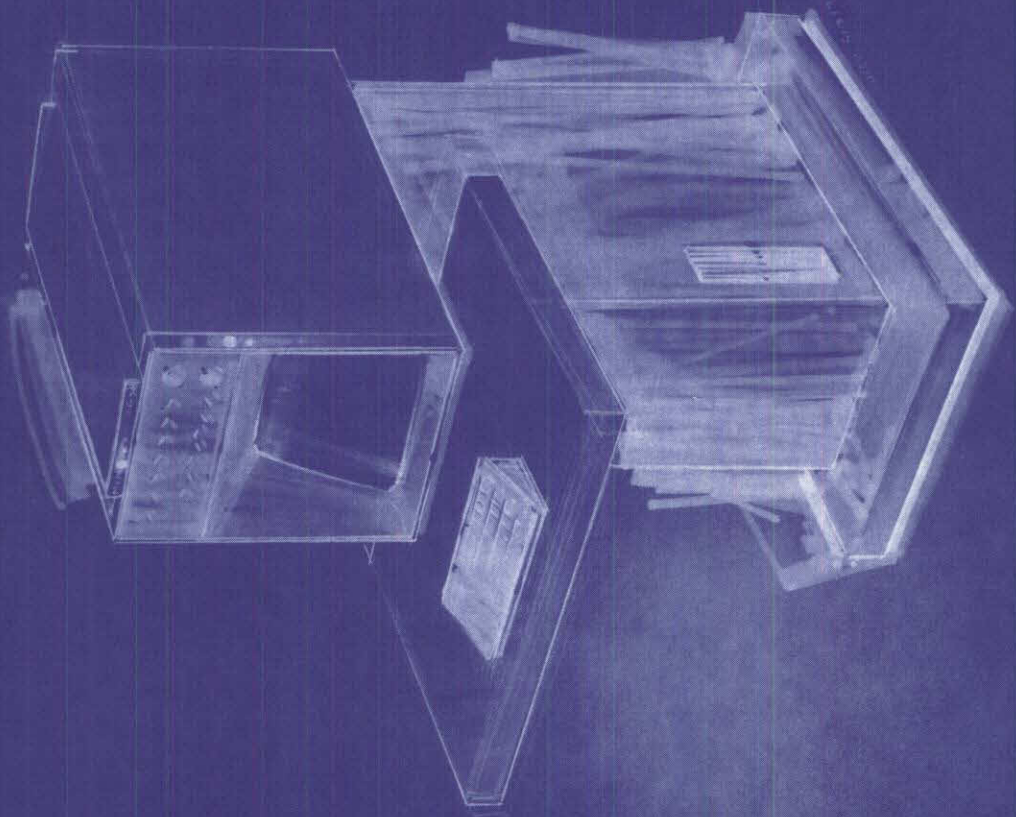
Relative Humidity

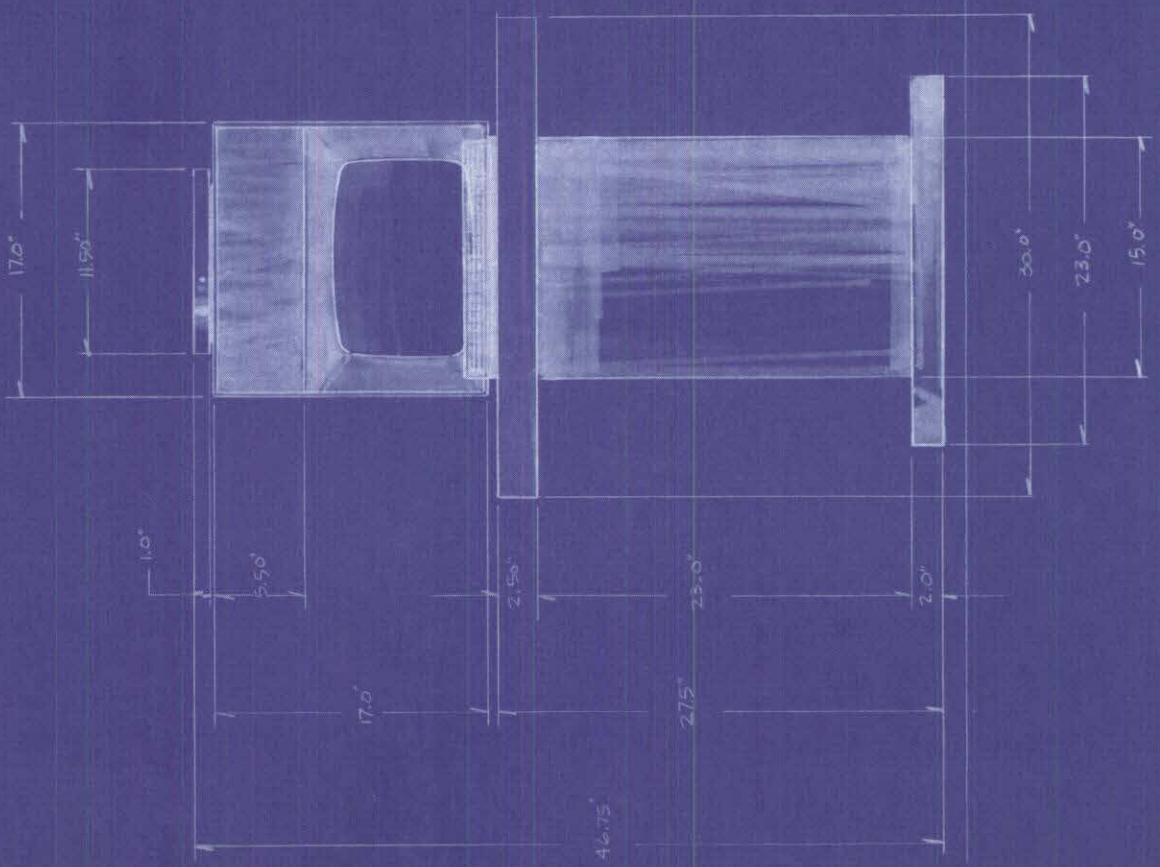
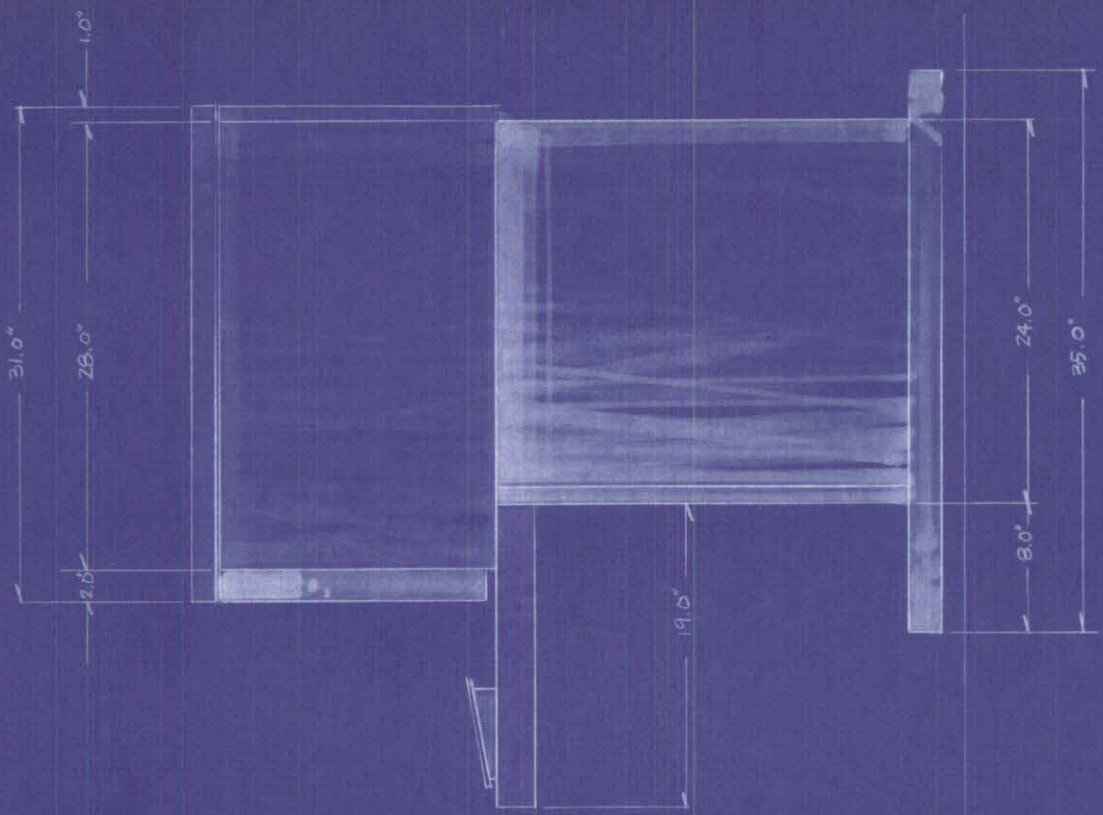
8-80%

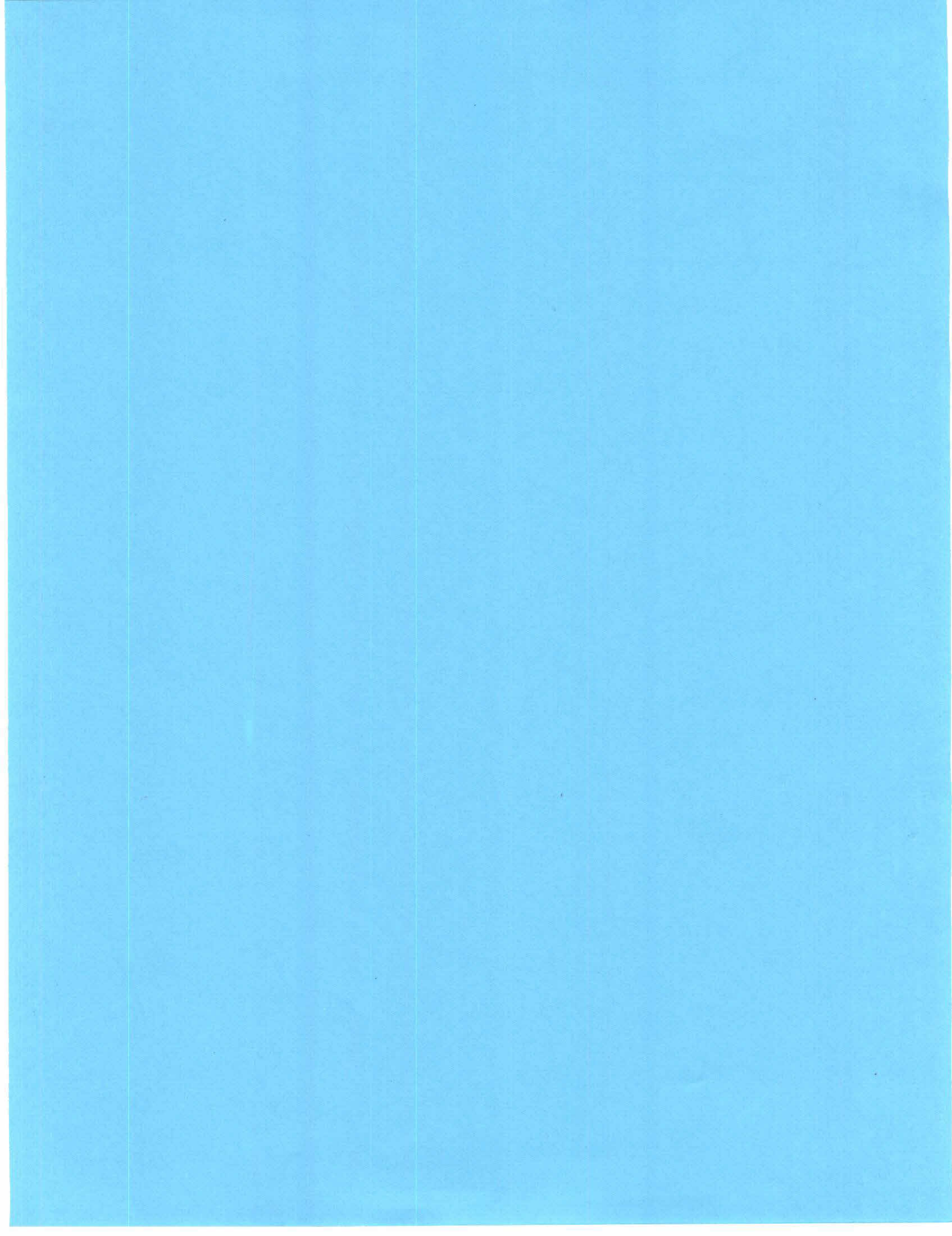
Maximum Wet Bulb

80^oF









MEMOREX 7100

POWER SUPPLY

G. EWART

7/19/72

MEMOREX CONFIDENTIAL

The power system designed for the MRX30 is based on the recently developed switching type of power supply circuit. This type of circuit has many advantages over the "conventional" type of power supply. First; it runs cool because the regulation is controlled with storage elements rather than dissipative elements. Second; it is smaller because the high frequencies involved allow much smaller components to be used. Third; it is more easily repaired in the field because the system lends itself to replaceable plug in packages. Fourth; it is low cost: even through the system is more complex, the components are cheaper and the overall cost is equal to or less than that of a conventional system.

The power supply has overcurrent protection on all supply voltages and shuts down automatically for any over voltage or under voltage condition. Over voltage crowbars also protect circuit boards from momentary high voltage surges. The power sequencing unit allows the MRX30 to be used as a peripheral device for a larger computing system where it must go through a remote controlled power up sequence. On small system configurations cost can be reduced by removing one of the 5V modules. For large system configurations, power can be extended by adding 5V modules, at a rate of 50 amps per module.

MRX30 Power Supply Specifications

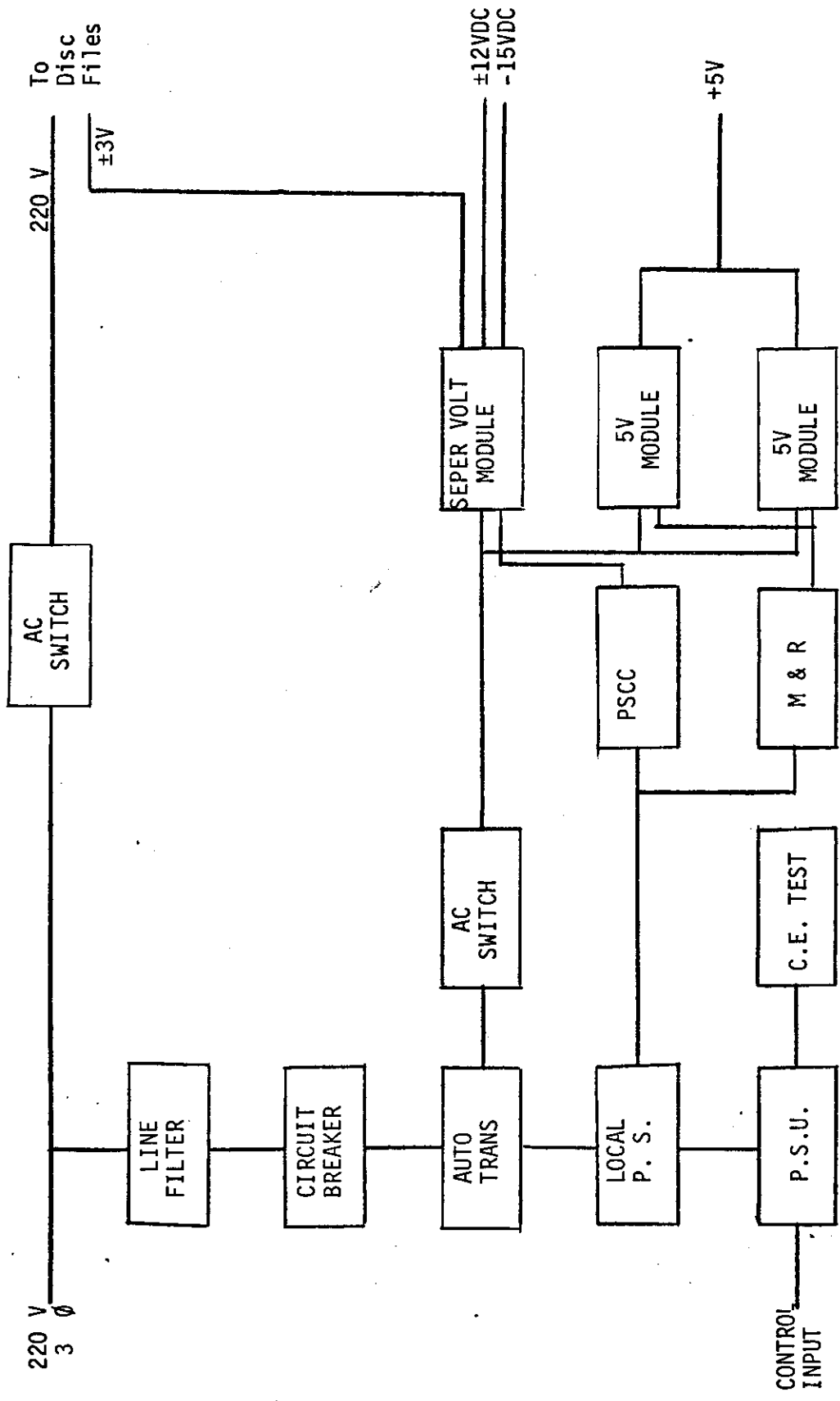
Supply Voltages

Voltage	Tolerance	Current
+ 5	± 5%	100 amp
±12	± 4%	±6 amp
+ 3	± 2%	2 amp
- 3	± 2%	5 amp
-15	±10%	2 amp

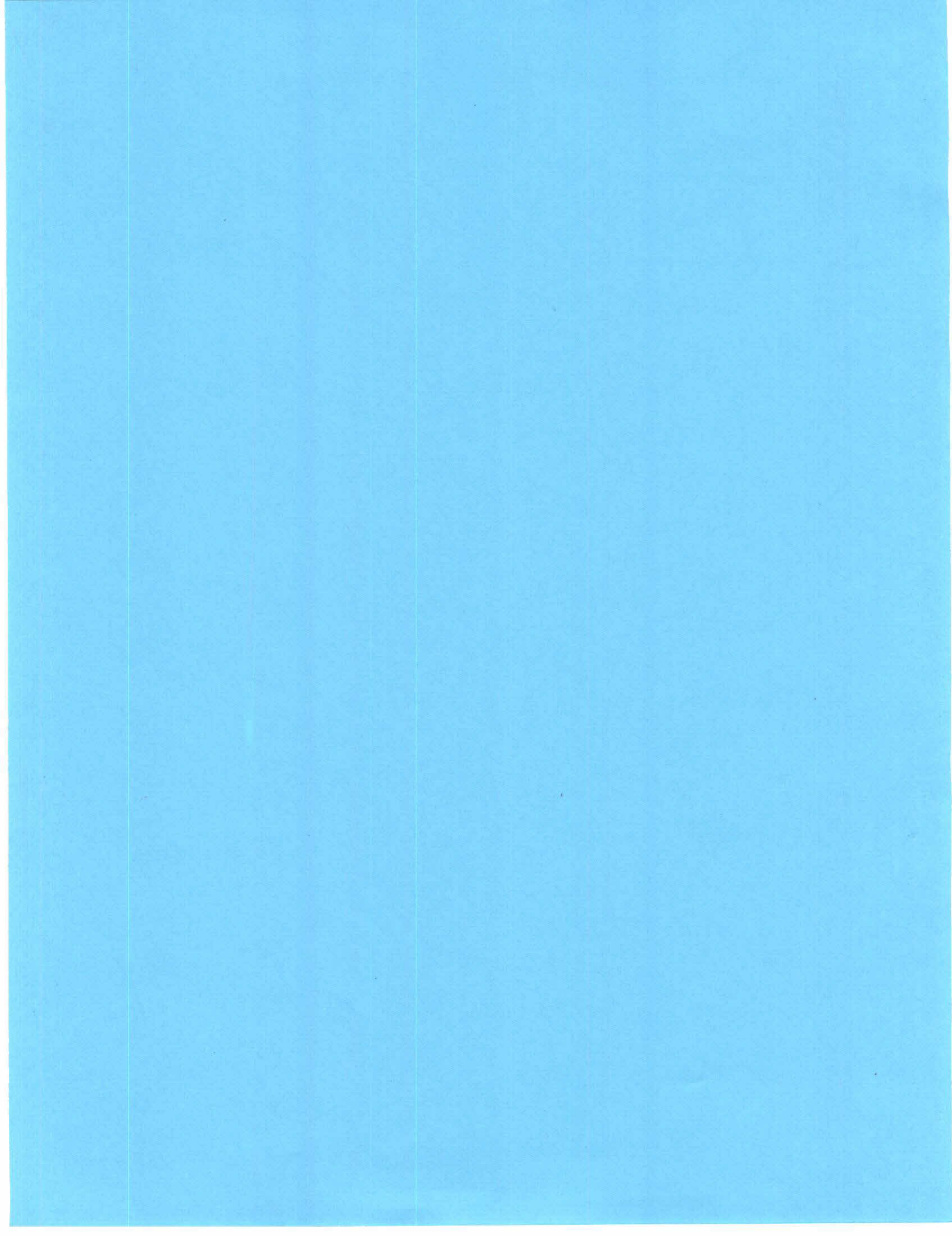
Line Voltage

220 V

3 phase



MEMOREX 30 POWER SUPPLY



MEMOREX 7100

OPSYS1 EMULATION PACKAGE

R. CHUEH

R. HOEHNLE

7/19/72

MEMOREX CONFIDENTIAL

CONTENTS

- 1.0 Introduction
 - 1.1 The 7300 CPU
 - 2.0 Priority Structure of the 7100 CPU
 - 2.1 Memory Accessing Priority
 - 2.2 CPU Processor - State Assignment
 - 2.3 CPU Processor - State Switching
 - 2.4 The Process in State F
 - 3.0 The Busy and Active Bits of 7300 CPU
 - 4.0 Organization of the Emulation Package
 - 5.0 The Tie-Breaker Register and Process Scheduling
- Figure 1. 7100 System Register File
- Figure 2. Organization of the Emulation Package

1.0 INTRODUCTION

This section will outline the 7100 Emulation Package. The Emulation Package has been designed so that OPSYS1 can be executed on the 7100 CPU with minimum modification. The OPSYS1 has been designed for the 7300 CPU which has a time-slicing multi-processor structure. The emulation of the multi-processor states is the main concern of this section.

1.1 THE 7300 CPU

The 7300 CPU is organized into eight (8) semi-independent processor-states. Each processor-state is assigned a particular processing job and is competing with each other for shared CPU sources and central memory access. The CPU is time sliced into time slots of major cycles, (0.9 μ s or 1 μ s). At the end of the cycle a new processor state is scheduled for the next cycle. The active processor states are scheduled in a "round-robin" algorithm supplemented by dynamic priority logics. Switching from one processor-state to another is hardware controlled and is overlapped with the normal instruction execution. For detailed description of 7300 CPU organization, please refer to its Production Description.

2.0 Priority Structures of the 7100 CPU

7100 has three levels of priority structures. The Central Memory assigns fixed priorities to its accessing ports from the highest refresh port to the lowest CPU port. Within the CPU, fixed priorities are also assigned to its processor-states associated with columns of the register-file. The higher priority processor can "interrupt" any of the lower priority processors at the "break" points. Each I/O adaptor is assigned to a processor-state. The majority of the non-time-dependent

data processing jobs are scheduled on the processor state F, the lowest priority one. To emulate OPSYS/I within the processor-state F, software will simulate the 8-state assignments of MRX/50 for non-critical-time-dependent jobs.

2.1 Memory Accessing Priority

Accessing priorities to the Central Memory have been fixed in the following order:

Refresh

DMA 1 (Assigned to DISC data transfer)

DMA 2 (Reserved for Selector Channel data)

DMA 3 (Reserved)

(SPARE)

(SPARE)

(SPARE)

CPU

Switching from one port to another:

Switching Quantum = Memory Cycle = 1.2 us

Switching (Transition) Time = 0

2.2 CPU Processor-State Assignments

The Register-file in CPU is partitioned into columns with each column associated to a processor-state. At any instant, only one of the processor-states is active.

The P-register identifies the current active processor-state and so the associated column of the register-file. Normal addressing to the register-file is restricted to the column of the active processor-state. (Address to other columns can be specified through the use of the X-register).

Fig. 1 depicts the organization of the Register File. There are 16 columns but only 9 columns have registers, the columns from 8 to E are empty. In the first 8 columns (processor-state 0-7), only half of the 16 registers per column are presented in the register file, the rest are located in the various I/O adaptors. These "Virtual I/O Registers" can be addressed as if they were part of the register-file, the meaning and interpretation of a fetch or store to these "virtual registers", however, are part of the particular I/O adaptor organization.

The 9 processor-states are assigned as:

<u>Processor State</u>	<u>Assignment</u>
0	DISC Commands
1	Selector Channel Commands
2	Communication Adaptor
3	Printer Adaptor
4	Card Reader Adaptor
5	MFCU
6	Console
7	Timer
F	General Data PProcessing

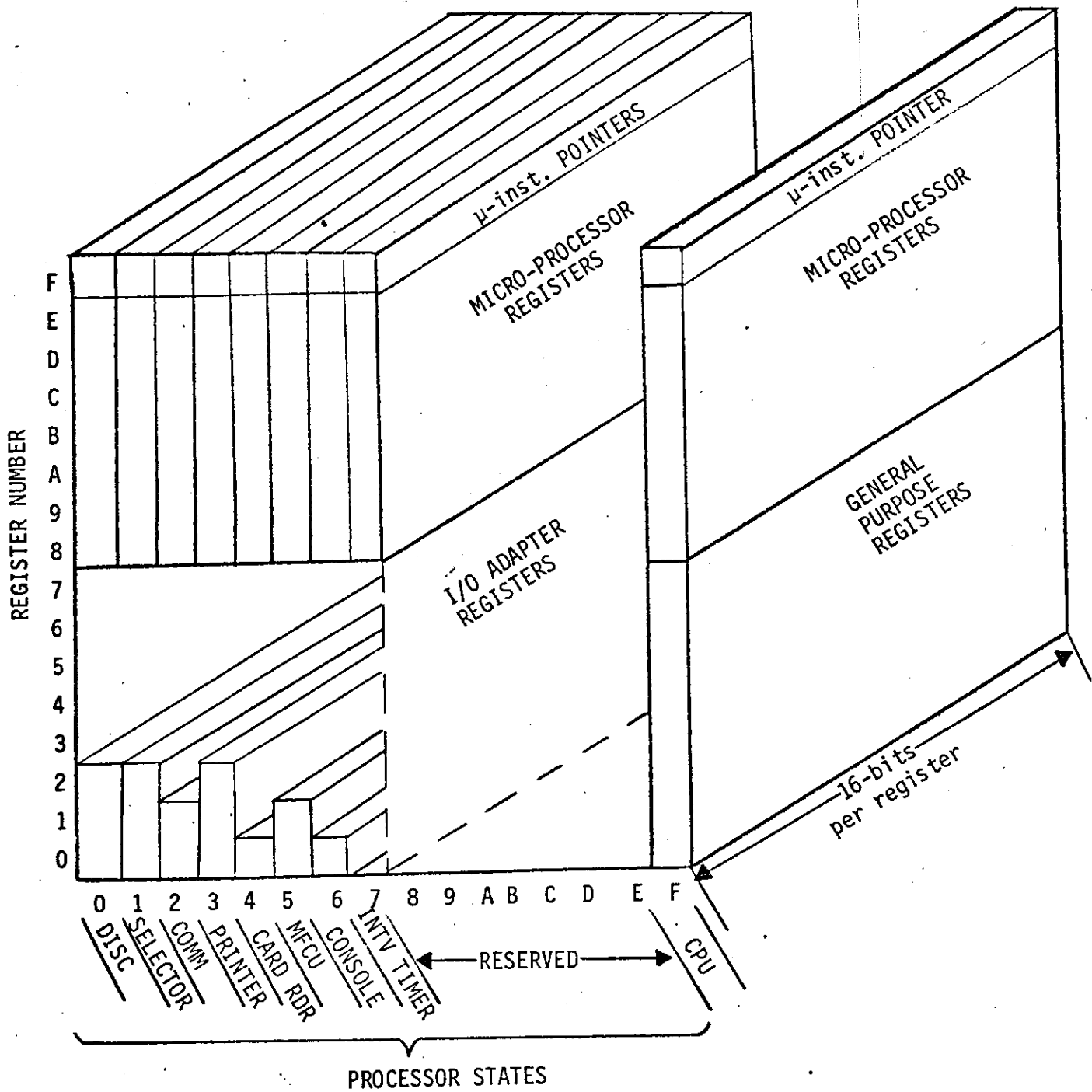


FIGURE I.

7100 SYSTEM REGISTER FILE

2.3 CPU Processor-State Switching

The process of interrupt and switch to higher priority procedures in 7100 CPU is accomplished with a priority network and a four-instruction BREAK subroutine. A processor-state can request service by raising the Service Request Line of the processor (usually by hardware logic in response to an external condition). The priority network will select the highest priority processor-state (lowest in Hex number) which has the Service Request Line raised. The select processor-state is encoded into a four-bit number which can be loaded into P-register when a "CST" micro-instruction is issued. There is also a comparison network that compares the four-bit value with the current P-register and the output of the comparison can be tested by a conditional branch micro-instruction.

The micro-routine to test and execute a higher level processor state interrupt is:

BREAK: Branch to Next if no higher request; else Branch to Switch, save return in R(F).

NEXT:

SWITCH: Gate "CST" to P

Branch to location specified by R(F)

When a processor-state finishes its current processing, it will have its Service Request Line reset and executes the following micro-instruction:

FINI: Branch to Switch, Save Return in R(F).

Switching from one processor-state to another:

Switching Quantum = Maximum Time Span between two BREAKS

Switching (Transition) Time:

The BREAK (including SWITCH) = $5 \times 0.4 = 2 \mu\text{s}$ (taken)
= $0.4 \mu\text{s}$ (not taken)

The FINI (including SWITCH) = $4 \times 0.4 = 1.6 \mu\text{s}$

The maximum Time Span between two BREAKS is a critical parameter and is calculated to be $30 \mu\text{s}$

2.4 The Processes in State F

The lowest priority processor-state (F) has 16 registers and is used to process all non-time dependent jobs. Within the processor-state F, processes are organized to emulate the 8-processor-state of 7300 CPU. The scheduling of the 8 simulation processes is of major concern in the following section. Since all time-dependent jobs has been "farmed-out" to either DMA channels or high priority processor-states (0-7) there is no critical timing requirement on the scheduling of the processes. It has, however, to be logically correct in emulating 7300 processor-states, to avoid possible dead-lock situation and to improve overall system performance.

3.0

The Busy and Active Bits of 7300 CPU

Two flip-flops (the busy and active bits) are provided for each processor-state in 7300 CPU. These bits are used to indicate the state of the processor and the Busy bit is also used for requesting CPU cycle (time-slot). When a processor has its Busy bit reset, it is either not active or is waiting for an external signal to wake it up. To distinguish the "wait" state from the "in active" state, an Active bit is provided.

The Busy and Active bits are also used as attention signals for sending messages from one processor to another. When a processor sends a message to another, it attaches the message into a queue in the central memory and sends an attention signal to wake up the receiving processor-state. If the receiving processor-state is already running, the attention signal will be ignored; if it is waiting for some specific completion signal, it should not be disturbed (do not wake-up by the signal). The logic can be implemented in two ways: one is to have the attention signal masked by the receiving processor, the other is restraining the sending processor from issuing the attention signal. The MRX/40 and 50 systems adapt the latter discipline. For example, when the executive Processor (4) issues a job to an I/O Processor, it will put the job description into a work queue and set both Busy and Active bits on the I/O Processor if it is not already Active. If the receiving

processor is already Active, it is either running (Busy) or is waiting for an I/O completion signal. In either situation, the Executive Processor then will not disturb the Busy or Active bits of the receiving processor. The message attached to the work queue will only be processed after the I/O processor has serviced the I/O completion signal or the current running job.

When a processor-state is serving more than one device (e.g. the Communication Processor in the 7300 CPU controls up to 16 lines), the functions of the Busy/Action bits have to be somewhat modified. The Busy bit is still used to request CPU cycles. To display whether each device controller is waiting for a completion signal, an (Active) bit is needed for each device (line). Since the processor is now controlling more than one device, waiting completion signal from one device (having the particular Active bit set) should not block the processor from processing jobs issued for other devices. The "non-disturb" discipline of single device processor has to be modified into a multiplexing organization. In addition, the completion signals have to be held high indefinitely since they may not be recognized immediately.

4.0 Organization of the Emulation Package

Fig. 2 outlines the organization of the emulation package for execution OPSYS/I in 7100 CPU. Basic to the emulation package and not shown in Fig. 2 is a machine language emulator that emulates all non-privileged MRX/50 instructions.

In OPSYS/I, every processing job is assigned to one of the 8 processors. Programs executed by processors 4 to 7 are exclusively machine language subroutines. Programs executed by the four I/O processors (0-3) include both micro-command and machine language routines. The general approach of the emulator is to create an 8-processor organization within the 7100 CPU state F so that most of the machine language programs can be executed with minimum or no modification. The micro-command I/O routines that handle I/O command initializations and terminations as well as low speed data transfers are rewritten in the 7100 micro-commands and to be executed in dedicated 7100 CPU states (0-7). In terms of system/360, state 0 to 7 correspond to eight "I/O channels" and state F corresponds to "CPU".

The disc and selector channel data movements, processed by short micro-command lccps in 7300 CPU, are transferred to be handled by hardwired logics in DMA channel #1 and #2.

As we have discussed in the last section, the states of the 7300 processors are indicated by their Busy/Active bits and they are scheduled accordingly. The Busy bits are still used to indicate that a simulating process is in a ready state. The allocation of the CPU resources to the process, however, is quite different. Instead of time slicing on every major cycle, processes are usually allowed to execute (in state F) to completion. Changing from one process to another includes the lengthy operation of swapping (state F) CPU registers. The exact processes scheduling algorithm which involves the Tie-Breaker Register and Timer is detailed in the last section.

To activate an I/O state (0 to 7), the responsible CPU process loads its micro-command pointer (register F of the I/O State) and sets its Service Request Line. All states have their service request lines set are competing for execution at next program BREAK point. When a process-state finishes its current processing, it will have its Service Request Line reset and signals the responsible process (via its Busy/Active bits) before it issues a BREAK.

Integrated with each I/O state is an I/O adaptor for controlling a particular device. 8 of the 16 register addresses of the state are reserved for addressing adaptor registers. For disc and selector channel adaptors, they also include registers of the DMA's and thus provide communication and control of DMA operations.

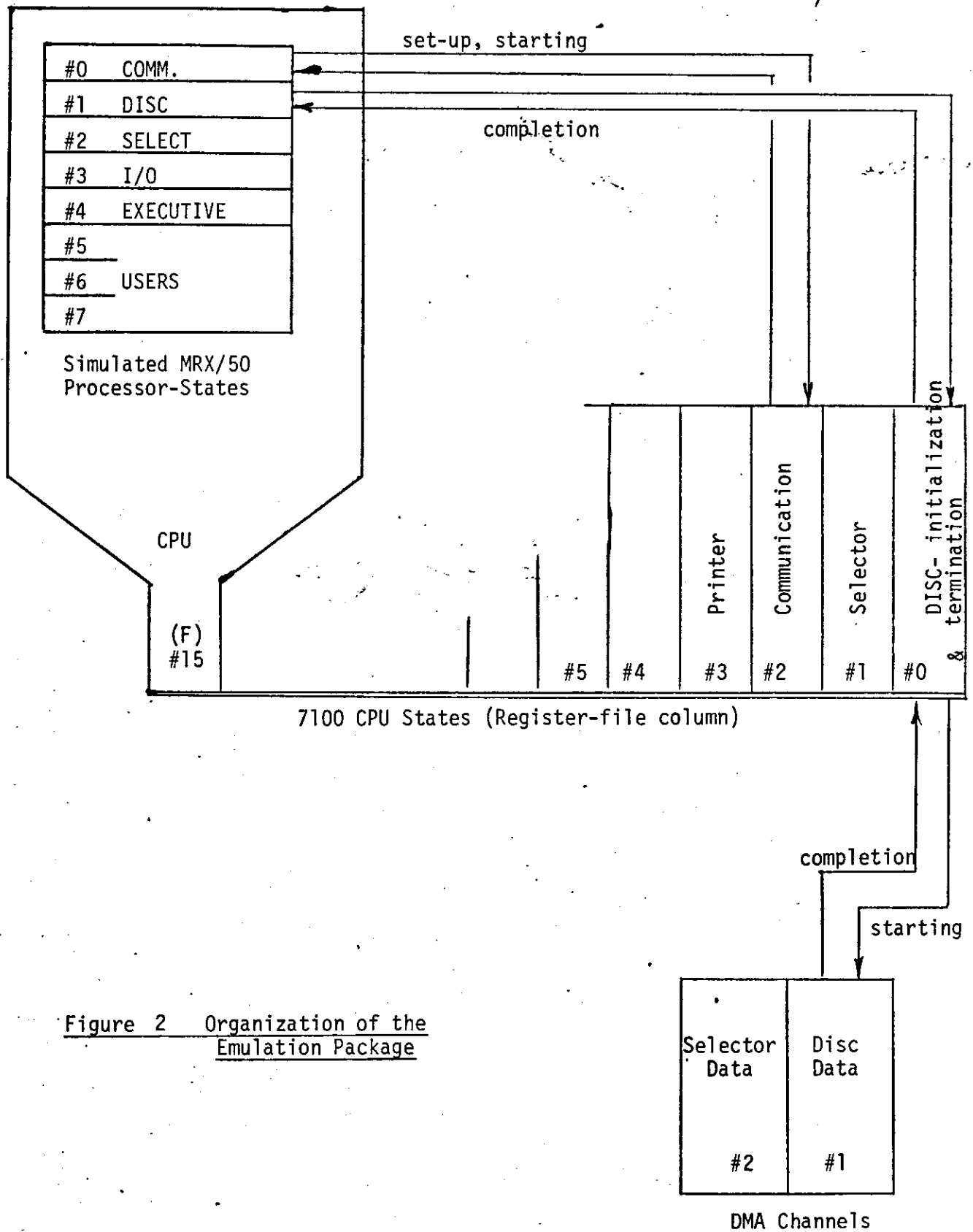


Figure 2 Organization of the Emulation Package

5.0 The Tie-Breaker Register and Process Scheduling

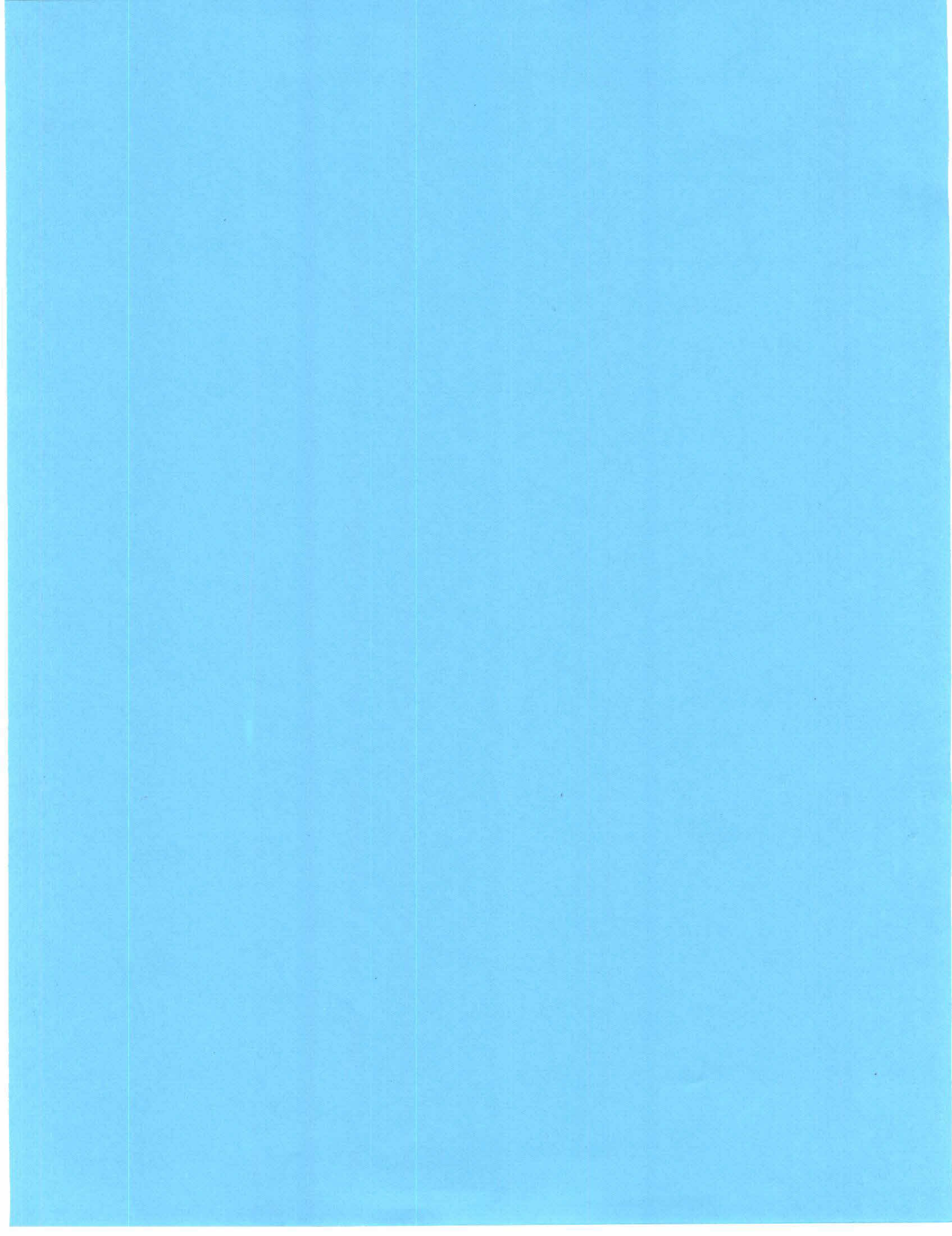
The Tie-Breaker Register is a hardware facility provided by the 7300 CPU to ease the implementation of synchronization of processors. In OPSYS/I, at the entry of a work queue a processor tests-and-sets a bit designated for the work queue in the Tie-Breaker Register. If the bit has already been set, the processor will loop and test again until the bit is reset.

The 7100 is a single processor system. Switching from one processor-state to another can only occur at program designated BREAK points. Therefore, the processors synchronization logic of test-and-set can be implemented with a simple fetch-branch-and-store macro provided no process switching is allowed inside the macro.

To emulate OPSYS/I in the 7100, the convention of simply looping if the Tie-Breaker Register is found set has to be somewhat modified. Because the processors in the 7300 CPU are scheduled cycle by cycle on "round-robin" basis, there is no danger of getting into "dead-lock" situation. The corresponding emulation processes are scheduled to run to completion in the 7100. Simple looping may create a "dead-lock" situation. The following conventions are adopted for the emulation package:

- (1) A Process becomes ready when its Busy bit is set. A process becomes not ready when its Busy bit is reset. A process becomes blocked when it tests the Tie-Breaker Register and found the bit set. A process becomes not blocked when the bit in the Tie-Breaker Register is reset.
- (2) Among the ready and not blocked processes, CPU is allocated to the highest priority process. The priority is fixed according to the process number (0 to 7).
- (3) An allocated process is normally scheduled to run to its completion (in Processor-State F)*except at the occurrence of the following conditions:
 - (a) Timer Expiration
 - (b) Manipulation of Busy bits
 - (c) Clear Tie-Breaker Register
 - (d) Test Tie-Breaker Register and found the bit set
- (4) The occurrence of condition (a), (b), or (c) will result to a process switching to a higher priority process which is ready and not blocked. If the current process is the highest one, no process switching will result.
- (5) The occurrence of condition (d) will always result to a process switching to a higher priority process which is ready and not blocked.

* (Remark) All processes (programs) executing in Processor-State F have to provide BREAKS for high-priority I/O Processor-State at least every 30 μ s.



MEMOREX 7100

MRX30 EMULATION AND PERFORMANCE

J.A. MILLER

7/19/72

MEMOREX CONFIDENTIAL

TABLE OF CONTENTS

1. Conclusion
2. Introduction
3. Evaluation of the MRX30
 - 3.1 Comparison of MRX30 and MRX50
 - 3.2 Comparison of MRX30 and IBM System 3
4. Emulation of the MRX30 Instruction Set
 - 4.1 The Basic Ops
 - 4.2 Branch Operations
 - 4.3 Variable Length Operations
 - 4.4 The Weighted Average Over Instruction Types

APPENDIX: Tables

1. Conclusion

The Memorex 30 has, on the average, an instruction execution speed 1.08 times faster than the IBM System 3.

2. Introduction

The purpose of this discussion is to evaluate the performance of the MRX30. In addition, the evaluation provides a basis to describe how the instruction set is emulated by the 7100 microinstructions. Throughout this discussion reference is made to the memo CPU Speed of MRX50 by G. H. Leichner (Santa Clara Systems Programming Technical Memo PER 002, January 26, 1972), in which similar evaluative techniques were used. It is particularly valuable as the source for instruction mix ratios. Instruction execution times for the MRX50 were obtained from the 7200/7300 Computers Product Description manual of March 1972.

3. Evaluation of the MRX30

Since the MRX30 user instruction set is compatible with the MRX50, a comparison of execution speed with the MRX50 is the natural way to evaluate the MRX30. Once this comparison has been made, the results may be translated into the results of comparisons with other computers, notably the IBM System 3.

In what follows we shall be dealing with ratios of execution times of various instructions. If we let $T(I)$ denote the execution time on machine I , the ratio we normally use is $R = \frac{T(MRX30)}{T(MRX50)}$. We also use

a ratio R' which means the same except that the MRX50 is assumed to be in single processor mode. Let us now let $S(I)$ be the speed of machine I . Since speed bears an inverse relation to time, we also have:

$$R = \frac{S(MRX50)}{S(MRX30)}$$

Where the R here is the same as above.

3.1 Comparison Between The MRX30 And MRX50

The MRX30 has a 400 nanosecond microinstruction cycle time and a 1.2 microsecond memory cycle time. In comparison, the MRX50 has a 100 nanosecond microinstruction cycle time and a .900 nanosecond memory cycle time. Thus, the MRX50 is 4 times the speed of the MRX30 in terms of microinstruction execution, but only 1.3 times the speed in memory operations. On this basis one would predict that the ratio R of MRX50 speed to MRX30 speed would fall between 4 and 1.3, with those operations requiring fewer memory cycles closer to 4, and those requiring more memory cycles closer to 1.3. This is true in almost all cases.

Another difference between the MRX30 and MRX50 is in the use of memory. In the MRX50 the microinstruction cycles are always locked to main memory cycles. In the MRX30, the only time the microinstruction execution becomes locked to memory timing is during a memory operation. This allows a timing on the MRX30 to be made in terms of microcycles rather than memory cycles. The result of this is that some operations, like indexing, take no time on the MRX50, but do take time on the MRX30.

The actual comparison between the MRX30 and MRX50 was done by microcoding several instructions on the MRX30 and comparing their timing to the MRX50. The instructions chosen were ADDR, ADD, MOVR, LOD, B (ranch), BCT and MOVX. It should be noted that this subset consists of Register-Register, Memory-register and variable length memory ops. ADDR and ADD were chosen as being representative of a large class of combinatorial ops. MOVR, LOD, B and BCT were chosen because they are frequently used ops. MOVX was chosen to represent the variable length ops. Section 4 describes this comparison in detail.

The comparison showed that, for all the ops but MOVX, the ratio R was very strongly dependent upon the indirect addressing and indexing options chosen. In particular, the more indirect addressing that was done, the smaller R became. As a result of this observation, it was decided to perform the analysis both on the simple (no indexing, no indirect addressing) versions of the ops, and also on an average op obtained by a weighted average of addressing types. The weighting factors were computed by assuming: 1) 50% of the ops would have at least 1 indirect address, and 2) 50% of the ops would use indexing.

It was also decided to perform the analysis using both the normal and "single processor mode" MRX50.

From these techniques an execution time for each op for each case was obtained. The times thus obtained were averaged using a set of weights derived from operation versus frequency of execution tables. The results of these weighted averages could be said to represent the contribution of the subset of instructions to the average instruction execution time. This gave a set of times both for the MRX50 and for the MRX30 which could be used to compute a ratio for each of 4 cases. These results were:

CASE	R
Simple Addressing - Normal MRX50	2.4
Addressing Average - Normal MRX50	2.3
Simple Addressing - Single Processor MRX50	1.9
Addressing Average - Single Processor MRX50	1.9

As expected, the worst case arises for simple addressing and a normal MRX50. It is on this case that the conclusions given in Section 1 above are based. It is surprising that the result is relatively insensitive to differences between simple and average addressing. This arises for two reasons. The first is that the addressing average is weighted towards the simple ops. The second is that over half the contribution to the average instruction execution time comes in all cases from MOVX, the op that is not address type sensitive. It is worth noting at this point that Leichners memo shows that, even with decimal multiply and divide excluded, the variable length operations contribute over half to the average execution time in the MRX50.

3.2 Comparison of MRX30 with IBM System 3

Having now arrived at a figure of 2.4 for the ratio for MRX50 speed to MRX30 speed, we need to relate this figure to the IBM System 3. If we use S(I) for the speed of machine I, then we have the following relations:

$$\frac{S (MRX50)}{S (MRX30)} = 2.4$$

The published figure for the MRX50 shows:

$$\frac{S (MRX50)}{S (IBM50)} = .7$$

The following two relations represent assumptions:

$$\frac{S (IBM30)}{S (IBM50)} = .3$$

$$\frac{S (IBM S3)}{S (IBM 30)} = .9$$

We have then:

$$\begin{aligned} \frac{S (MRX30)}{S (IBM S3)} &= \frac{S (MRX30)}{S (MRX50)} \cdot \frac{S (MRX50)}{S (IBM50)} \cdot \frac{S (IBM50)}{S (IBM30)} \cdot \frac{S (IBM30)}{S (IBM S3)} \\ &= \frac{1}{2.4} \times .7 \times \frac{1}{.3} \times \frac{1}{.9} = 1.08 \end{aligned}$$

Which is the figure given above.

4. Emulation of the MRX30 Instruction Set

The starting and ending point in the emulation of any instruction is the read next instruction sequence (RNIZ, Table 1). This sequence picks up the first word of the next instruction from memory and increments the program counter to the next word. In addition it checks for a pending interrupt condition. Once the instruction is available, the RNIZ exits to a location dependent upon the 8 bit op-code of the instruction.

In what follows we shall discuss in some detail how various instructions and instruction types are emulated. In addition we will give examples of the microprogramming for these emulations, and some timing analysis based upon this microcoding.

4.1 The Basic Ops

By the term basic ops here, we mean the functions of ADD, SUB, CMP, XOR, EOR, MOV, and MVI, in conjunction with the addressing types RR, MR, Direct, Immediate, and Memory-memory.

The PIRD decode branch for each of the basic ops branches to a sequence of two subroutine calls. The first of these subroutine calls is a branch to an addressing type routine. The second call is a branch to an "operator" function that actually performs the operation.

The first (addressing type) routine takes care of getting the operands. The second (operation type) routine performs the operation on the operands, and takes care of disposition of the results, including the condition code, if any.

Thus PIRD becomes one of a number of program segments like:

```
(PIRDX)  BSR = XXXX  
        BSR = XXXX
```

Where the individual routines are given in tables below:

	ADDRESSING TYPE	
	ROUTINE TABLE	
<u>XXXX</u>	<u>ADDRESSING TYPE</u>	<u>OPERATION TABLE</u>
ADRCT	DIRECT	1
AIMM	IMMEDIATE	1
AMR	MEMORY → REGISTER	1
ARR	REGISTER → REGISTER	1
AMM	MEMORY → MEMORY	2

OP TABLE 1

<u>YYYY</u>	<u>OPERATION</u>
OADD	ADD
OSUB	SUBTRACT
OAND	AND (Logical Product)
OIOR	EXCLUSIVE OR
OIOR	INCLUSIVE OR
OCMP	COMPARE
OMOV	MOVE
OMVI	MOVE INVERSE

OP TABLE 2 - Note these are in general alternate entries to routines listed in Op Table 1.

<u>YYY</u>	<u>OPERATION</u>
MADD	ADD
MSUB	SUBTRACT
MAND	AND (Logical Product)
MEOR	EXCLUSIVE OR
MIOR	INCLUSIVE OR
MCMP	COMPARE
MMOV	MOVE
MMVI	MOVE INVERSE

For the purposes of the timing analysis, the ARR and AMR addressing routines, and the OADD and OMOV operation routines were micro-coded. This coding is shown in Tables 2, 3, 4 and 5. Timings were obtained from this coding for all addressing variations of the instructions ADDR, MOVR, ADD and LOD. Summarys of these timings appear on Tables 6 and 7.

From these times an average time for each instruction was also computed. Each addressing variation in this average was weighed according to the following rules: 1) 50% of the instructions would use indirect address; and 2) 50% of the instructions would use indexing.

These calculations were made for the MRX50 in both the normal (MRX50 Short) and single processor mode (MRX50 Long). Tables 8 and 9 show these calculations.

4.2 Branch Operations

Emulation of the various branch operations generally consists of two program segments: 1) The execution of a branch condition test routine; 2) followed under certain conditions by the execution of a branch generation routine. It should be noted that the double subroutine call technique is not necessary since the branch generator used by a particular branch test routine is unique. The unconditional branches will branch directly to a branch generator, since no condition test is necessary.

BRANCH GENERATORS

- BGN1 - Generates a normal, post-indexed branch
- BGN2 - Generates a register branch
- BGN3 - Generates a pre-indexed branch
- BGN4 - (Alternate entry to BGN3) Generates a non-indexed branch

BRANCH CONDITION TEST ROUTINE

RTN	USED FOR	CORR. BRANCH GENERATOR
TBA1	Branch Add 1	BGN1
TBA2	Branch Add 2	BGN1
TBOF	Branch if Bit off	BGN4
TBON	Branch if Bit on	BGN4
TBRN	Branch if Reg not zero	BGN1

TBRZ	Branch if Reg Zero	BGN1
TBCF	Branch if Condition False	BGN1
TBCT	Branch if Condition True	BGN1
TBS1	Branch Subtract One	BGN1
TBS2	Branch Subtract Two	BGN1

For the purposes of the timing analysis one branch generator (BGN1) and one branch condition test routine (TBCF) were coded. These are shown in Tables 10 and 11. From this coding timings we obtained for all addressing types for the instructions B and BCF. For BCF timings were obtained for both the branch and no branch cases. These results are presented in Table 12. In order to arrive at a single value for the branch operation, the following assumptions were made: 1) 50% of branch operations were unconditional and 50% were conditional; 2) 50% conditional branches were taken, and 50% were not. Further, for address averaging, the assumptions made in Section 4.1 were also made. The results of these averages are shown in Table 13.

4.3 Variable Length Operations

The variable length operations first pass through a variable length set up section which computes the effective operand addresses and rectifies the count fields. This initialization section is instruction independent, except that SHFK and MOVL have their own. Subsequent to this set up section, a section dependent upon the actual op is reached. This is accomplished by the same sort of double branch technique employed for the basic ops.

For the purposes of the timing analysis, a set-up section and the instruction dependent section for MOVX were microprogrammed. This coding is shown in Tables 14 and 15. For the timing, it was assumed (following Leichner) that the two operand lengths were both equal at 14.8 bytes. A summary of the timing for all addressing options is shown in Table 16. Since the times are only slightly dependent on addressing option, it was decided to use, for the MRX30, a value of 65.8 for the no address average case, and 69.0 for the address average case.

4.4 The Weighted Average Over Instruction Type

From Leichners memo a set of weights, one for each instruction, were derived that are proportional to the frequency of execution of that instruction. By using these as the weights in a weighted average we arrive at an overall figure of merit. There are 6 instances of this figure of merit. They break down into 2 cases of 3 instances each. The two cases are with and without address averaging. The 3 instances are: 1) The MRX50 running in normal mode (MRX50 Short); 2) The MRX50 running in single processor mode (MRX50 Long); and 3) the MRX30. Once the figure of merits have been developed, the ratios that were summarized above are calculated. It should be noted that, although only ADD and ADDR were evaluated, it was assumed that SUB and SBR would be the same and the weights were adjusted to include them in the average.

MRX30 EMULATION AND PERFORMANCE

APPENDIX: TABLES

RNIZ AMARIS, FMK
B = I(x'a')
S = ADD, HDM
PIR = MDR
BRA = PIRD
BRA. INT = IRUPT

Timing = 6 cycles

42.381 50 SHEETS 5 SQUARE
42.382 100 SHEETS 5 SQUARE
42.389 200 SHEETS 5 SQUARE
NATIONAL

Table 1 - RNIZ Microcode

ARR BRA, N122 = ARRI
 BRA = SRET
 B = R2I

ARRI MAR = R2I, FMR
 BRA = SRET, HOM
 B = MDR

 3 opels for OS and 4
 5 opels for RD indirect

Table 2 - RR type address decode

```

AMR      AMAR=S, FMR
          S=ADD
          BRA. I2Z=AMRCP
          ,HDM
          MDR → MAR, FMR
AMRCP    BRA. R2Z=AMRCX
          ,HDM
          MAR=MDR, FMR
AMRCB    BRA=SRIT, HDM
          B=MDR

AMRCX    A=R2I, HDM
          B=MDR
          MAR=ADD, FMR
          BRA=AMRCB

```

timing 9 cycles for direct with no index
 10 cycles for direct with index
 11 cycles for indirect with no index
 12 cycles for indirect with index

Table 3 - MR type address decode

```

OADD      BRA: NIIZ = OADI
          A = R1I
          R1I = ADD
OADB      A = MCR
          B = COND
          COND = IZ, HDM
          BRA: RNIZ
OADI      MAR = R1I, FMR
          , HDM
          A = MDR
          MDR = ADD, FMW
          BRA = OADB

```

timing - 7 cycles when R1 is direct
 11 cycles when R1 is indirect

Table 4 - ADD Routine

OMU BRA = R1Z = OMU
 RAI = BDB
 BRA = R1Z

OMU MDR = BDB
 MAR = R1I, FMW
 , HDM
 BRA = R1Z

timing 3 cycles when R1 direct
 6 cycles when R1 indirect

Table 5 - MU Routine

ADD & ADDR Timing

	MRX50			MRX30				
	Cycles	Time	T'	u. cycles	time	R	R'	
RR	M ₁ +M ₂	1.7	2.1	18	7.2	4.2	3.4	
RI	2 M ₁	1.8	2.2	20	8	4.4	3.6	
II	3 M ₁ +M ₂	3.5	4.3	22	8.8	2.5	2.0	
II	4 M ₁	3.6	4.4	24	9.6	2.6	2.2	
MR	3 M ₁	2.7	3.3	24	7.6	3.6	2.9	
M(X)R	3 M ₁	2.7	3.3	25	10.0	3.7	3.0	
M(I)R	4 M ₁	3.6	4.4	26	10.4	2.9	2.4	
M(IX)R	4 M ₁	3.6	4.4	27	10.8	3.0	2.5	
MI	5 M ₁	4.5	5.5	28	11.2	2.5	2.0	
M(X)I	5 M ₁	4.5	5.5	29	11.6	2.6	2.1	
M(I)I	6 M ₁	5.4	6.6	30	12.0	2.2	1.8	
M(IX)I	6 M ₁	5.4	6.6	31	12.4	2.3	1.9	

Table 6 - ADD & ADDR Timing Summary

MΦV - LΦD Timing

	MRX 50			MRX 30		R	R'
	Cycles	Time	T'	Cycles	Time		
RR	M1+M2	1.7	2.1	14	5.6	3.3	2.7
2I	2M1	1.8	2.2	16	6.4	3.6	2.9
1I	2M1+M2	2.6	3.2	17	6.8	2.6	2.1
II	3M1	2.7	3.3	19	7.6	2.8	2.3
MR	3M1	2.7	3.3	20	8.0	3.0	2.4
M(X)R	3M1	2.7	3.3	21	8.4	3.1	2.5
M(I)R	4M1	3.6	4.4	22	8.8	2.4	2.0
M(IX)R	4M1	3.6	4.4	23	9.2	2.6	2.1
MI	4M1	3.6	4.4	23	9.2	2.6	2.1
M(X)I	4M1	3.6	4.4	24	9.6	2.7	2.2
M(I)I	5M1	4.5	5.5	25	10.0	2.2	1.8
M(IX)I	5M1	4.5	5.5	26	10.4	2.3	1.9

Table 7 - MΦVR & LΦD timing summary

42-381 50 SHEETS 5 SQUARE
42-382 100 SHEETS 5 SQUARE
42-383 100 SHEETS 5 SQUARE



NATIONAL

ADDR MRX 50 short

$$1.7 \times .5 + (1.8 + 3.5 + 3.6) / 6$$

$$= \left[1.7 + \frac{(1.8 + 3.5 + 3.6)}{3} \right] / 2 = 2.3 \quad (R = 3.5)$$

ADDR MRX 50 long

$$\left[2.2 + \frac{(2.2 + 4.3 + 4.4)}{3} \right] / 2 = 2.9 \quad (R' = 2.8)$$

ADDR MRX 30

$$\left[7.2 + \frac{(8 + 8.8 + 9.6)}{3} \right] / 2 = 8$$

ADD MRX 50 short

$$\left[2.7 + 2.7 + \frac{(3.6 + 3.6 + 4.5 + 4.5 + 5.4 + 5.4)}{3} \right] / 4 = 3.6 \quad (R = 2.9)$$

ADD MRX 50 long

$$\left[3.3 + 3.3 + \frac{(4.4 + 4.4 + 5.5 + 5.5 + 6.6 + 6.6)}{3} \right] / 4 = 4.4 \quad (R' = 2.4)$$

ADD MRX 30

$$\left[9.6 + 10.0 + \frac{(10.4 + 10.8 + 11.2 + 11.6 + 12.0 + 12.4)}{3} \right] / 4 = 10.6$$

Table 8 - Address Averaging for ADD & ADDR

MUR MRX50 short

$$= \left[1.7 + \frac{1.8 + 2.6 + 2.7}{3} \right] / 2 = 2.0 \quad (R=3.2)$$

MUR MRX50 Long

$$\left[2.1 + \frac{2.2 + 3.2 + 3.3}{3} \right] / 2 = 2.5 \quad (R=2.5)$$

MUR MRX30

$$\left[5.6 + \frac{6.4 + 6.8 + 7.6}{3} \right] / 2 = 6.3$$

L&D MRX50 short

$$\left[2.7 + 2.7 + \frac{(3.1 + 3.6 + 3.6 + 3.6 + 4.5 + 4.5)}{3} \right] / 4 = 3.3 \quad (R=2.7)$$

L&D MRX50 Long

$$\left[3.3 + 3.3 + \frac{(4.4 + 4.4 + 4.4 + 4.4 + 5.5 + 5.5)}{3} \right] / 4 = 4.0 \quad (R=2.2)$$

L&D MRX30

$$\left[8.0 + 8.4 + \frac{(8.8 + 9.2 + 9.2 + 9.6 + 10.0 + 10.4)}{3} \right] / 4 = 8.9$$

TABLE 4 Address Averaging for MUR & L&D

42,381 50 SHEETS 3 SQUARE
42,382 100 SHEETS 3 SQUARE
42,383 200 SHEETS 3 SQUARE
42,384 400 SHEETS 3 SQUARE
42,385 800 SHEETS 3 SQUARE
42,386 1600 SHEETS 3 SQUARE
42,387 3200 SHEETS 3 SQUARE
42,388 6400 SHEETS 3 SQUARE
42,389 12800 SHEETS 3 SQUARE
42,390 25600 SHEETS 3 SQUARE
42,391 51200 SHEETS 3 SQUARE
42,392 102400 SHEETS 3 SQUARE
42,393 204800 SHEETS 3 SQUARE
42,394 409600 SHEETS 3 SQUARE
42,395 819200 SHEETS 3 SQUARE
42,396 1638400 SHEETS 3 SQUARE
42,397 3276800 SHEETS 3 SQUARE
42,398 6553600 SHEETS 3 SQUARE
42,399 13107200 SHEETS 3 SQUARE
42,400 26214400 SHEETS 3 SQUARE



BGN1

MAR = S, FMR
BRA.NIIZ = BQNA
, HDM
MAR = MDR, FMR

BQNA

BRA.NIIZ = BGNX
, HDM
S = MDR
BRA = RNIIZ

BGNX

A = RLI, HDM
B = MDR
S = ADD
BRA = RNIIZ

- Timing:
- 6 cycles direct
 - 7 cycles indirect
 - 8 cycles indirect
 - 9 cycles indirect and indirect-

Table 10 - Branch Generator BGN1

TBCF

B = PIR

A = COND

BBIT

BRA. NUZZ = BGN1

A = S

B = X'2'

S = ADD

BRA = RNIZ

timing: 4 cycles if branch

8 cycles if no branch

42.381 50 SHEETS 5 SQUARE
42.382 100 SHEETS 5 SQUARE
42.383 200 SHEETS 5 SQUARE
42.384



Table 11 - Branch Condition Test TBCF

Branch Timing

Unconditional Branch

	MRX 50		T'	MRX 30		R	R'
	Cycle	Time		Cycle	Time		
N	2 M ₁	1.8	2.2	12	4.8	2.7	2.2
X	2 M ₁	1.8	2.2	13	5.2	2.9	2.4
I	3 M ₁	2.7	3.3	14	5.6	2.1	1.7
I X	3 M ₁	2.7	3.3	15	6.0	2.2	1.8
Branch on Condition (true or false)							
N - B̄	2 M ₁	1.8	2.2	14	5.6	3.1	2.5
N - B	2 M ₁	1.8	2.2	16	6.4	3.6	2.9
X - B̄	2 M ₁	1.8	2.2	14	5.6	3.1	2.5
X - B	2 M ₁	1.8	2.2	17	6.8	3.8	3.1
I - B̄	3 M ₁	2.7	3.3	14	5.6	2.1	1.7
I - B	3 M ₁	2.7	3.3	18	7.2	2.7	2.2
I X - B̄	3 M ₁	2.7	3.3	14	5.6	2.1	1.7
I X - B	3 M ₁	2.7	3.3	19	7.6	2.8	2.3

Table 12 - Branch Timing Summary

For No Address Average

MRX50 is independent of Branch type & whether or not branch was taken

$$\text{MRX50 Short } t = 1.8 \quad (R = 3)$$

$$\text{MRX50 Long } t = 2.2 \quad (R' = 2.5)$$

$$\text{MRX30 } t = \left[4.8 + \frac{(5.6 + 6.4)}{2} \right] / 2 = 5.4$$

For Address Average

$$\text{MRX50 Short } t = (1.8 + 2.7) = 2.25 \quad (R = 2.6)$$

$$\text{MRX50 Long } t = (2.2 + 3.3) = 2.75 \quad (R' = 2.1)$$

$$\text{MRX30 } t = \left[(4.8 + 5.2 + 5.6 + 6.0) / 4 + (4 \times 5.6 + 6.4 + 6.8 + 7.2 + 7.6) / 8 \right] / 2 \\ = 5.85$$

Note that the above average averages over:

1. Branch Type
2. Addressing type
3. Whether Branch taken or not.

Table 13 - Branch Type & Address Averaging

<p>AUL AMAR = S, FMR B = X'2' S = ADD, HDM BRA. NR1Z = AULX1 OAI = MDR</p> <p>AULB1 AMAR = S, FMR B = X'2' S = ADD, HDM BRA. NR1Z = AULX2 OAI = MDR</p> <p>AULB2 AMAR = S, FMR B = X'2' S = ADD, HDM L1 = SMDR B = X'1' MAR = ADD, L2 = SMDR A = OAI B = L1 A = ADD B = X'1' OAI = SUB B = OAI A = SUB B = L2 OAI = ADD A = L1 CMP BRA. NCM = AULG6</p>	<p>AULX1 A = MDR B = R1I OAI = ADD BRA = AULG1</p> <p>AULX2 A = MDR B = R2I OAI = ADD BRA = AULB2</p>
<p>AULG6 RES = SUB CNT = L1 BRA = SRET PIR = X'0'</p>	<p>AULG6 B = SUB A = ERB RES = SUB CNT = L2 BRA = SRET PIR = X'8'</p>

33 + 3 for each X + 2 for LAG

TABLE 14 - Variable Length Set-up section

OMXUX B = Y'1'
 CNT = CNT
 BRA. #4B = OMXOT
 OMXTP AMAR = OA1, PMK
 OA1 = ADD, HDM
 AMAR = OA2, HMW
 OA2 = ADD
 A = CNT
 CNT = SUB, HDM
 BRA. N#0B = OMXTP
 OMXBT RES = RES
 BRA. #0B = RUIZ

(this section is unimportant because
 the evaluation will assume that
 $L_1 = L_2$, so that $RES = 0$)

5 + 8 L

Table 15 - MOUX op-dependant code

MDOX Timing

Set up Only

	MRX 30		MRX 50		T'	R	R'
	Cycles	Time	Cycles	Time			
N	46	18.4	6M, + 3M2	7.8	9.6	2.4	1.9
X	49	19.6	6M, + 3M2	7.8	9.6	2.5	2.0
2X	52	20.8	6M, + 3M2	7.8	9.6	2.7	2.2
L2G - N	48	19.2	6M, + 3M2	7.8	9.6	2.5	2.0
L2G - X	51	20.4	6M, + 3M2	7.8	9.6	2.6	2.1
L2G - 2X	54	21.6	6M, + 3M2	7.8	9.6	2.8	2.3

Loop Only
L₁ = L₂ = 14.8

	8L	47.4	2LM ₁	26.6	32.6	1.8	1.5
--	----	------	------------------	------	------	-----	-----

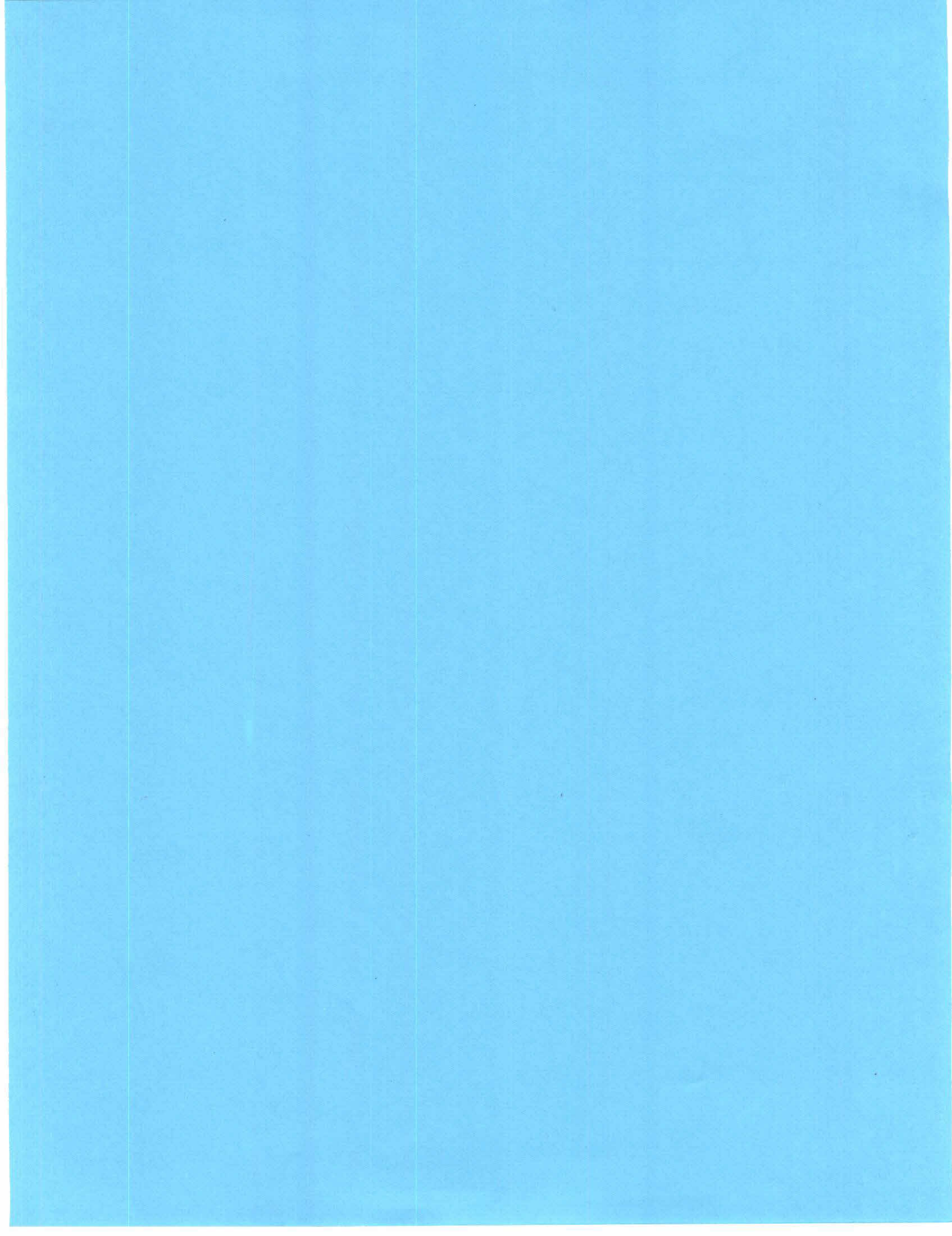
Total

N	46+8L	65.8	6M, + 3M ₂ + 2LM ₁	34.4	42.2	1.9	1.6
X	49+8L	67.0	"	34.4	42.2	1.9	1.6
2X	52+8L	68.2	"	34.4	42.2	2.0	1.6
L2G - N	48+8L	66.6	"	34.4	42.2	1.9	1.6
L2G - X	51+8L	67.8	"	34.4	42.2	2.0	1.6
L2G - 2X	54+8L	69.0	"	34.4	42.2	2.0	1.6

Table 16 - MDOX Timing Summary

Inst	Weight	N. Address Average				Address Average							
		MRX50 short	MRX50 long	MRX30	MRX50 short	MRX50 long	MRX30	MRX50					
ADDR	.0352	1.7	.05484	2.1	.07392	7.2	.25344	2.3	.06096	2.9	.10208	8.0	.2816
ADD	.0152	2.7	.04104	3.3	.05016	9.6	.14592	3.6	.05472	4.4	.06688	10.6	.16112
MOV	.0176	1.7	.02992	2.1	.03696	5.6	.04856	2.0	.0352	2.5	.044	6.3	.11088
LDD	.1626	2.7	.43902	3.3	.53658	8.0	1.3008	3.3	.53658	4.0	.6504	8.9	1.44714
JMP	.2324	1.8	.41832	2.2	.51128	5.4	1.25496	2.25	.5229	2.75	.6391	5.85	1.35954
MOVX	.0480	34.4	1.6512	42.2	2.0256	65.8	3.1584	34.4	1.6512	42.2	2.0256	69.0	3.312
TOTAL			2.64		3.23		6.21		2.88		3.53		6.67
R			2.4		1.9				2.3		1.9		

Table 17 - Parameters of the weighted Average over Instruction Type



MEMOREX 7100

MRX30 PHASE "0" COST ESTIMATE

M. GREGORY

7/19/72

MEMOREX CONFIDENTIAL

The following section presents the product cost estimates for the 7100 processor and peripherals. The assumptions used to generate these estimates are also presented. Field Engineering costs will be included at a later date.

COST ASSUMPTIONS

I. 7100 PROCESSOR

A. DIRECT LABOR (ULTIMATE)

1. Manufacturing

The following direct manufacturing hours are estimated for the three processor configurations:

Small Configuration	150 hours
Median Configuration	175 hours
Large Configuration	200 hours

These hours include contingency, processor assembly and unit test as well as system assembly and test. The direct labor rate was assumed to be \$4 per hour.

2. Quality Control and Receiving Inspection

Q.C. and R.I. is taken as 10% of the direct manufacturing hours at a rate of \$4.30 per hour.

B. MATERIAL (ULTIMATE)

Ultimate material costs for the processor and I/O Adapters are included on the attached matrix. All of these costs include a 30% contingency. Floor loss is estimated as 3% of ultimate PCB component material costs plus 0.5% of all other ultimate material costs.

C. BURDEN

1. Direct labor burden on manufacturing Q.C. and R.I. is 125%.

2. Material burden on direct material plus floor is 12.9%.

D. MANUFACTURING SUPPORT

Manufacturing support includes a 94% burden. These costs are distributed on the attached matrix.

E. MANUFACTURING PROGRESS

Total manufacturing progress is estimated as 11% of base ultimate manufacturing cost, and is distributed on the attached matrix.

F. DEVELOPMENT ENGINEERING EFFORT

The cost of Development Engineering is based on the following breakdown of the cost of an engineering manmonth.

Controllable Department Cost	\$2470/mo
Non-Controllable Dept. Costs	410/mo
Controllable Product Support	970/mo
Non-Controllable Product Support	160/mo
TOTAL	\$4010/mo

The development engineering effort does not include Programming or Publications.

G. E.C. SCRAP AND REWORK

E.C. scrap and rework is estimated as 2.5% of base ultimate manufacturing costs.

H. EXPENSE

Manufacturing expense includes a 94% burden, and is distributed on the attached matrix.

I. OTHER COSTS

Other costs includes the shipping group cost.

CPU AND ADAPTERS

BASE ULTIMATE MATERIAL COST - 7100 PROCESSOR

<u>ITEM</u>	<u>COST W CONTINGENCY</u>
CPU 10K (28K)	\$4138
CPU 14K (32K)	4416
CPU 18K (36K)	4809
CPU 26K (44K)	5365
CPU 34K (52K)	6036
IFA	520
SEL. CHANNEL	260
ASYNCR LA	130
BI-SYNCR LA (225%)	195
WIDEBAND LA (225%)	195
ICA	325
IPA 150 LPM (320%)	195
300 LPM (638%)	390
8610-11 ADAPTER (250%)	195
8633 ADAPTER (250%)	195
8643 ADAPTER (250%)	195
8653 ADAPTER (250%)	195
8655 ADAPTER (250%)	195
8660 ADAPTER (250%)	195
80 COL. READER (340%)	130
80 COL. R/P (185%)	195
DATA ENTRY ADAPTER (200%)	195
STORAGE PROT & TIMER	260
CONSOLE ADAPTER	130

MRX30 PERIPHERAL PRODUCT COST*

CONSOLE	\$ 1560
PRINTER 150 LPM	4225
300 LPM	6404
600 LPM	10358
1200 LPM	10514
TRIDENT	1288
8610	3313
8611	4166
8630	2798
8633	1508
8643	3283
8653	3907
8655	9088
8660	9093
80 COL. CR 300	2322
600	2533
80 COL R/P 500/100	14010
TAP CT + 30KB	6182
CT + 60KB	7061
30KB	4180
60KB	4690
DATA ENTRY DISPLAY	1560

*Includes material, labor and burden.

	1972	1973	1974	1975	1976	1977	TOTAL
MFG SUPPORT	—	\$328K	\$197K	\$66K	\$33K	\$31K	\$655K
MFG PROGRESS	—	10%	80%	10%	—	—	100%
DEV ENG	\$435K	\$1545K	\$960K	\$363K	\$210K	\$112K	\$3625K
E.C. SCRAP	—	10%	70%	15%	5%	—	100%
MFG EXPENSE	—	\$110K	\$270K	280	200	50K	\$910K
MFG CAPITAL	—	\$312	\$156K	\$26K	\$26K	—	\$520K

