Programming Manual

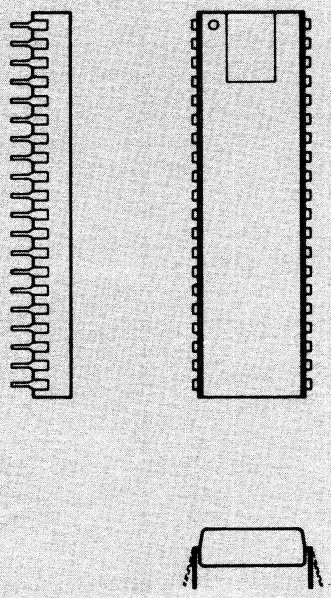
(SOFTWARE REFERENCE MANUAL)

AMI® S2000

REVISION 2
JUNE, 1978

A COMPLETE SINGLE-CHIP MICROCOMPUTER

ACTUAL SIZE



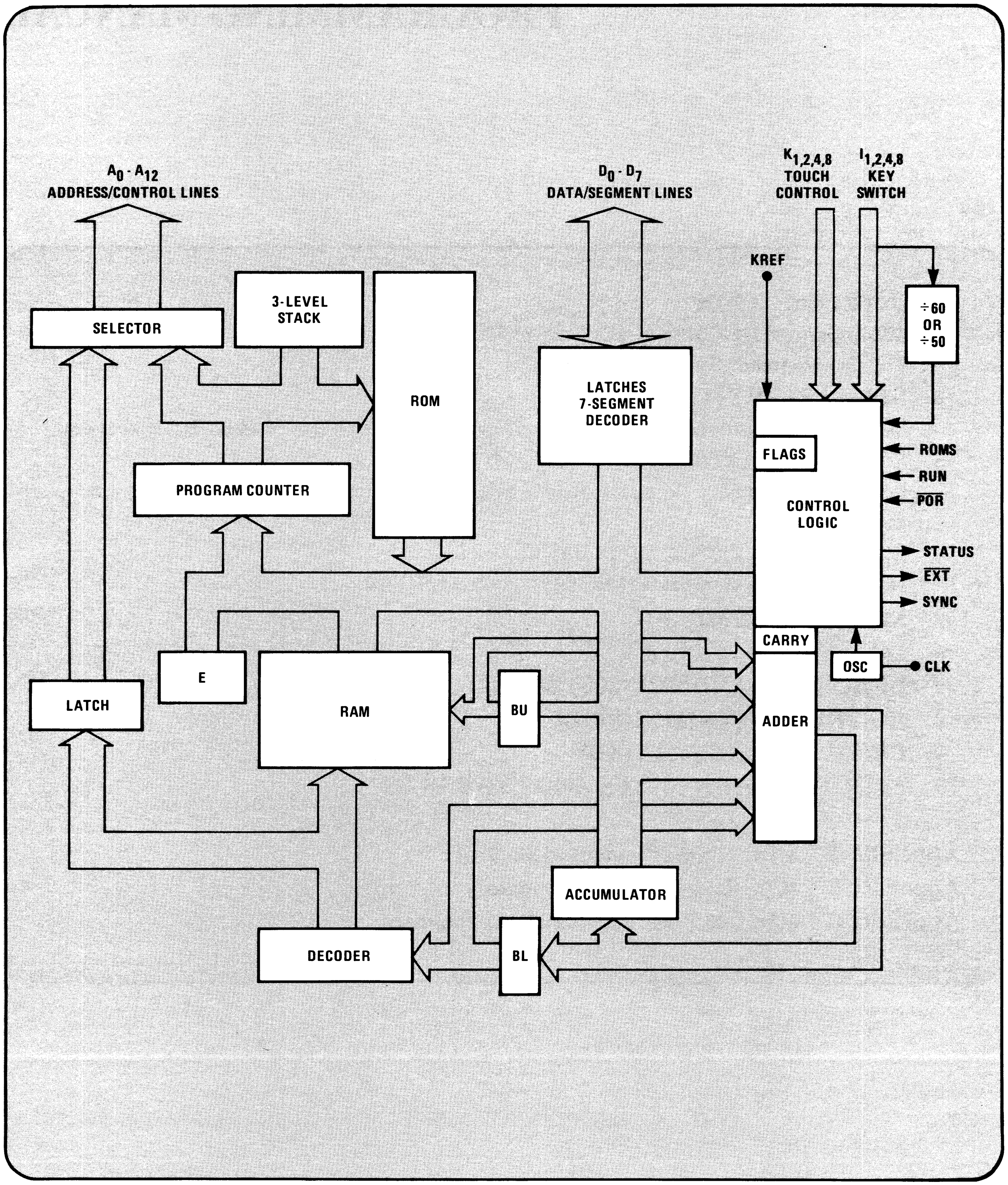
FEATURING

- 1024x8 Program ROM On-Chip and Expandable to 8192x8
- 64x4 Scratchpad RAM On-Chip
- 13 Outputs, 8 Inputs, Plus 8 Bi-Directional Three-State Lines
- TouchControl™ Capacitive Switch Interface
- Seconds Timer for Both 60Hz and 50Hz Lines
- 7-Segment Display Decoders and LED Drivers
- TTL-Compatible Outputs
- Single +9V Power Supply
- 4.5μs Execution Cycle
- 51 Instructions – All Single-Byte; 49 Are Single-Cycle
- 3-Level Subroutine Stack
- Built-in Production Test Mode
- Single-Step Capablity

IN A SINGLE STANDARD 40-PIN PACKAGE

AMI S2000 PROGRAMMING MANUAL

AMI S2000 SIMPLIFIED BLOCK DIAGRAM



1 INTRODUCTION

S2000 GENERAL DESCRIPTION

AMI's S2000 is a complete single-chip microcomputer which brings all the advantages of computer control to low-cost keyboard/display systems. The S2000 is ideal for a wide range of appliance and process monitoring applications. Versatile input/output and an instruction set optimized for its intended applications make the S2000 preferable to more expensive multiple-chip solutions with dramatic cost reductions during product design, manufacture, testing, and maintenance.

The S2000 has an on-chip 1024-instruction **ROM.** If necessary, additional program memory can be added externally up to a maximum of 8192 instructions. The **Program Counter** is a pointer to the next instruction to be executed. The **Subroutine Stack** holds return addresses during execution of subroutines.

The scratchpad **RAM** is used for temporary data storage of up to 64 4-bit data words, typically numeric quantities. The **BU and BL Registers** are pointers used to access RAM words. The **E Register** can be used as a general purpose register or as an index limit register for controlling RAM accesses.

Arithmetic and logical operations are performed by the **Adder** and 1-bit **Carry Register** with results stored in the **Accumulator** and the **Carry Register**. Two **Flag** bits are available which can be set, reset and tested as temporary indicators by software.

The **Control Logic** provides centralized control for all S2000 operation and includes three inputs and three outputs for interfacing external devices. The **Oscillator** generates all clocking signals and needs only an external RC circuit to set its rate. the **KREF Input** is the analog reference for TouchControl^{TM} and similar interfaces. Software decision-making instructions sample the four **K Inputs** and the four **I Inputs**, one of which can be used in conjuction with the **50 or 60 cycle frequency** counter to provide a one-second pulse for real-time applications.

The eight bidirectional three-state **D Lines** are general-purpose data signals. The thirteen **A Lines** are outputs for displays, keyboard strobes, and other applications.

APPLICATIONS

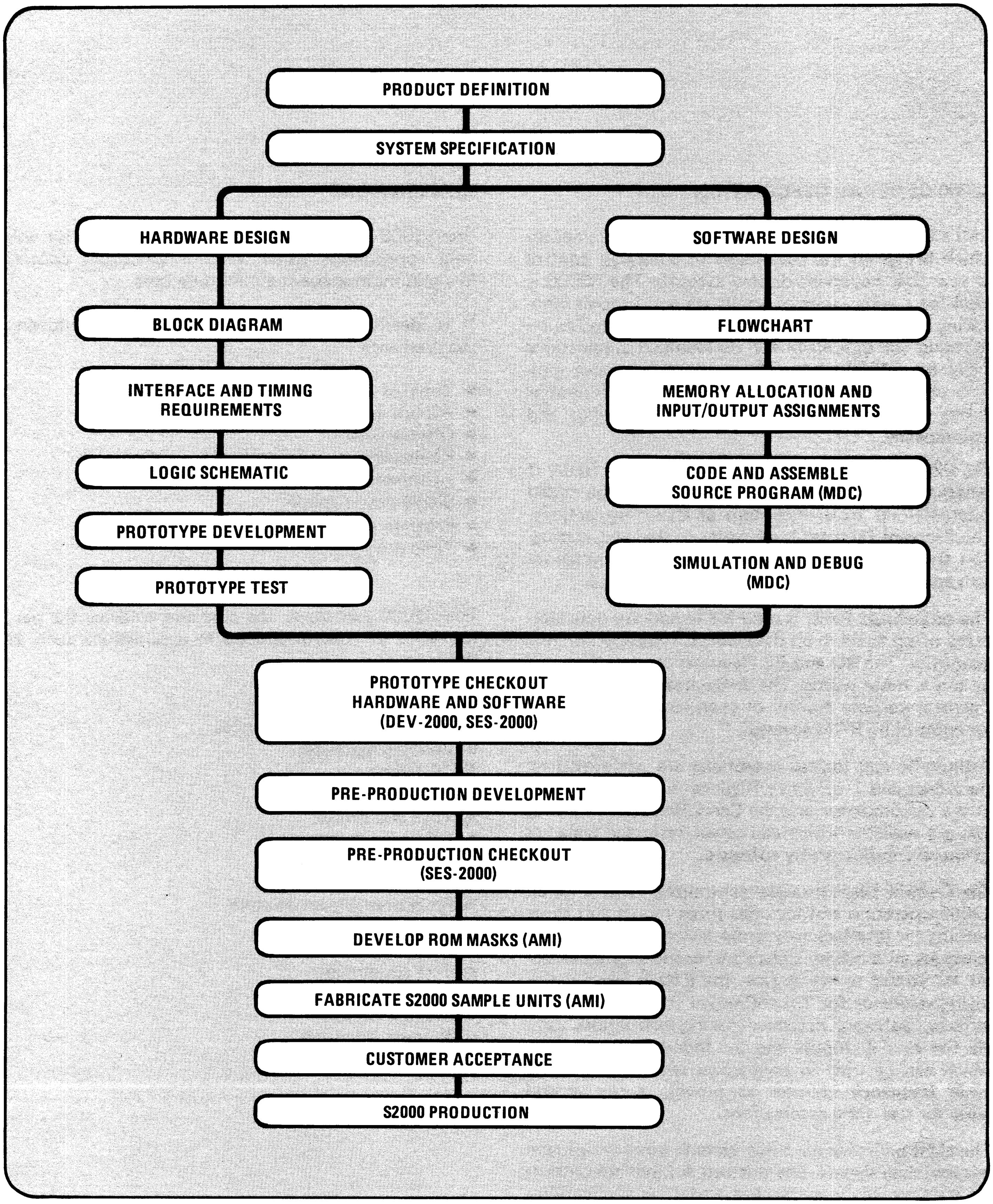
The S2000 is a computer on a chip, suitable for volume applications which require intelligent control in a minimum space at a minimum cost.

It is ideally suited for systems with the following requirements.

- Time-of-day and interval timer control
- AC line synchronization
- Display drive
- Keyboard inputs (ohmic or TouchControl)
- Arithmetic operations
- Single power supply
- Program expandability and testability
- Triac drive

The S2000 can lower the cost and enhance the performance of control circuits in applications such as the following:

- Major household appliances
- Vending machines
- CB radios
- Electronic scales
- Toys and games
- Lab instruments
- Point-of-sale devices
- Vehicle instruments
- Programmable calculators
- Data sampling devices
- Data logging devices
- Test equipment
- Keyboard devices
- Display devices
- Remote monitors


PRODUCT DEFINITION
SYSTEM SPECIFICATION
HARDWARE DESIGN
BLOCK DIAGRAM
INTERFACE AND TIMING REQUIREMENTS
LOGIC SCHEMATIC
PROTOTYPE DEVELOPMENT
PROTOTYPE TEST
SOFTWARE DESIGN
FLOWCHART
MEMORY ALLOCATION AND INPUT/OUTPUT ASSIGNMENTS
CODE AND ASSEMBLE SOURCE PROGRAM (MDC)
SIMULATION AND DEBUG (MDC)
PROTOTYPE CHECKOUT HARDWARE AND SOFTWARE (DEV-2000, SES-2000)
PRE-PRODUCTION DEVELOPMENT
PRE-PRODUCTION CHECKOUT (SES-2000)
DEVELOP ROM MASKS (AMI)
FABRICATE S2000 SAMPLE UNITS (AMI)
CUSTOMER ACCEPTANCE
S2000 PRODUCTION

THE DEVELOPMENT CYCLE

Product development using AMI's S2000 single-chip microcomputer proceeds in generally the following sequence. Following **Product Definition** and **System Specification**, the customer's development effort splits into the **Hardware Design** and **Software Design** phases.

The Hardware Design team produces a **Block Diagram** identifying the major functional blocks as well as **Interface and Timing Requirements.** Detailed design activity results in a **Logic Schematic** which is the "blueprint" from which actual hardware prototypes can be built. A "breadboard" or **Prototype** unit is built for the purpose of checking out both the hardware and software designs. **Prototype Test**, independent of the software effort, assures a level of confidence in the prototype unit prior to combining it with the actual software in the **Prototype Checkout** phase.

Working in parallel, but interacting with the hardware effort when necessary, the Software Design team produces a **"Flowchart"** which identifies the major functional and operational blocks of the software design. Detailed design activity may result in a second or more detailed flowchart depending on the complexity of the system. **Memory Allocation, Input/Output Assignments** and any special software requirements are also factored into the design. **Coding and Assembling** of the product software is followed by **Simulation and Debug** in a non-hardware environment prior to the Prototype Checkout phase. AMI'S MDC plays a major supporting role during these Coding, Assembling, Simulation and Debug stages of the Software Design effort.

Hardware and Software efforts join together for the **Prototype Checkout** phase which verifies that the actual hardware and software designs perform as required. AMI's MDC, DEV-2000 and SES-2000 development tools provide the required effective test environment during this checkout phase. A **Pre-Production Development and Checkout** phase may follow completion of the prototyping effort if the customer feels that the prototype units are not sufficiently representative of the final product.

When the customer is satisfied that the software is ready for production, AMI takes on the task of developing the **ROM Masks** and fabricating **Sample Units** of the product. Following **Customer Acceptance** of the sample units, AMI proceeds into high volume **S2000 Production.**

2 S2000 ARCHITECTURE

ROM

The **ROM** addressing range is divided into eight "banks" of 1024 locations. Bank 0 is on the S2000 itself and the others can be provided externally (Figure 2.1). Each bank is further divided into sixteen "pages" of 64 locations and each location holds one S2000 instruction. The **Program Counter** is thus thirteen bits wide — three bits for the bank, four bits for the page within a bank, and six bits for an instruction within a page.

When a power-on reset occurs, the Program Counter is zeroed, so execution starts at Bank 0, Page 0, Location 0. Normal execution proceeds sequentially until a Jump or Return forces a new Program Counter value.

To save ROM bits in Jump-type instructions, only the Location is indicated. A special instruction — **Prepare Page** — can set a new Page (and if required, a new Bank) just before a long (i.e., off page) Jump.

EXAMPLE: TO TRANSFER TO A LABEL LL IN THE SAME PAGE, USE

```
     JMP      LL
```

TO TRANSFER TO A LABEL LP IN A DIFFERENT PAGE, USE

```
     PP       LP/64          ;Set Page Address
     JMP      LP
```

TO TRANSFER TO A LABEL LB IN A DIFFERENT BANK, USE

```
     PP       LB/64          ;Set Page Address
     PP       LB/1024        ;Set Bank Address
     JMP      LB
```

As an additional ROM-saving feature, the JMS (Jump-to-Subroutine) instruction automatically performs its own Prepare Page to Page F (decimal 15). Hence, by convention, most subroutine entries reside on Page F. However, subroutines may reside on any page, and a Prepare Page instruction preceding a JMS will override the automatic Prepare Page to Page F.

The **Subroutine Stack** remembers the Page and the Location for three successive return addresses, which allows for complex "nested" logic within each Bank.

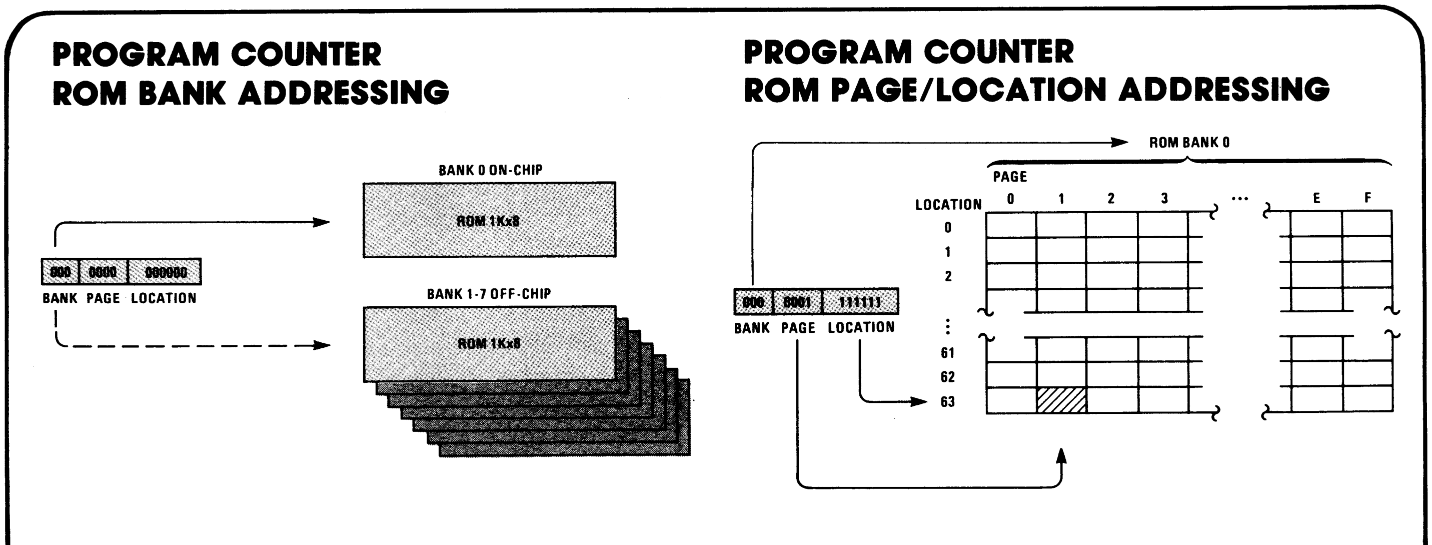


Figure 2.1

RAM

The S2000's **RAM** is divided into Pages and Locations – four Pages, each containing sixteen Locations. In each Location is a "Word" of four data bits. RAM is addressed by the BU (2-bit) and BL (4-bit) Registers. The BU Register selects one of four RAM pages, and the BL selects one of sixteen words in a page. Data handling is made simple by the fact that the BL Register is also a general-purpose register that can be operated upon arithmetically – there are instructions to load it, increment it, decrement it, and exchange it with the Accumulator.

Instructions that access RAM permit loading the addressed RAM word into the Accumulator and automatically incrementing or decrementing the BL Register. This permits processing RAM words top-down or bottom-up. In addition, within a selected RAM word, any bit can be set, cleared or tested.

Like the BL Register, the E Register can also be used for general storage, but it has the additional capability of serving as a RAM index pointer since it can be compared to BL and thus control RAM indexing limits.

To aid in program control, the S2000 includes two single-bit **Flags** which can be set, reset, or tested as (for example) subroutine parameters or result indicators.

RAM ADDRESSING

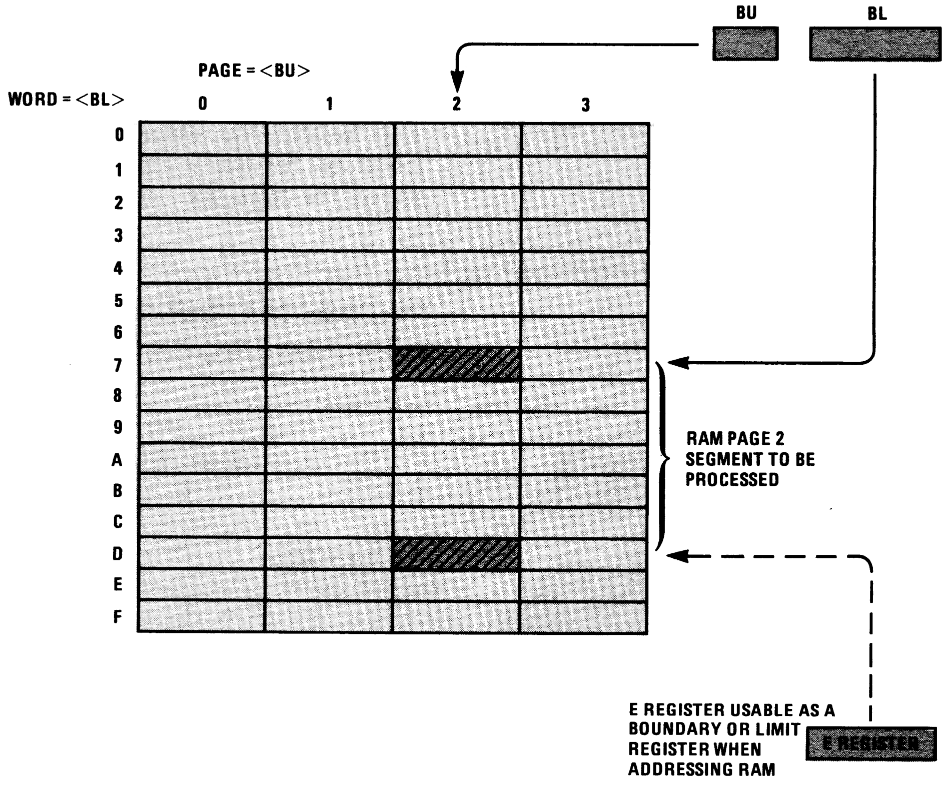
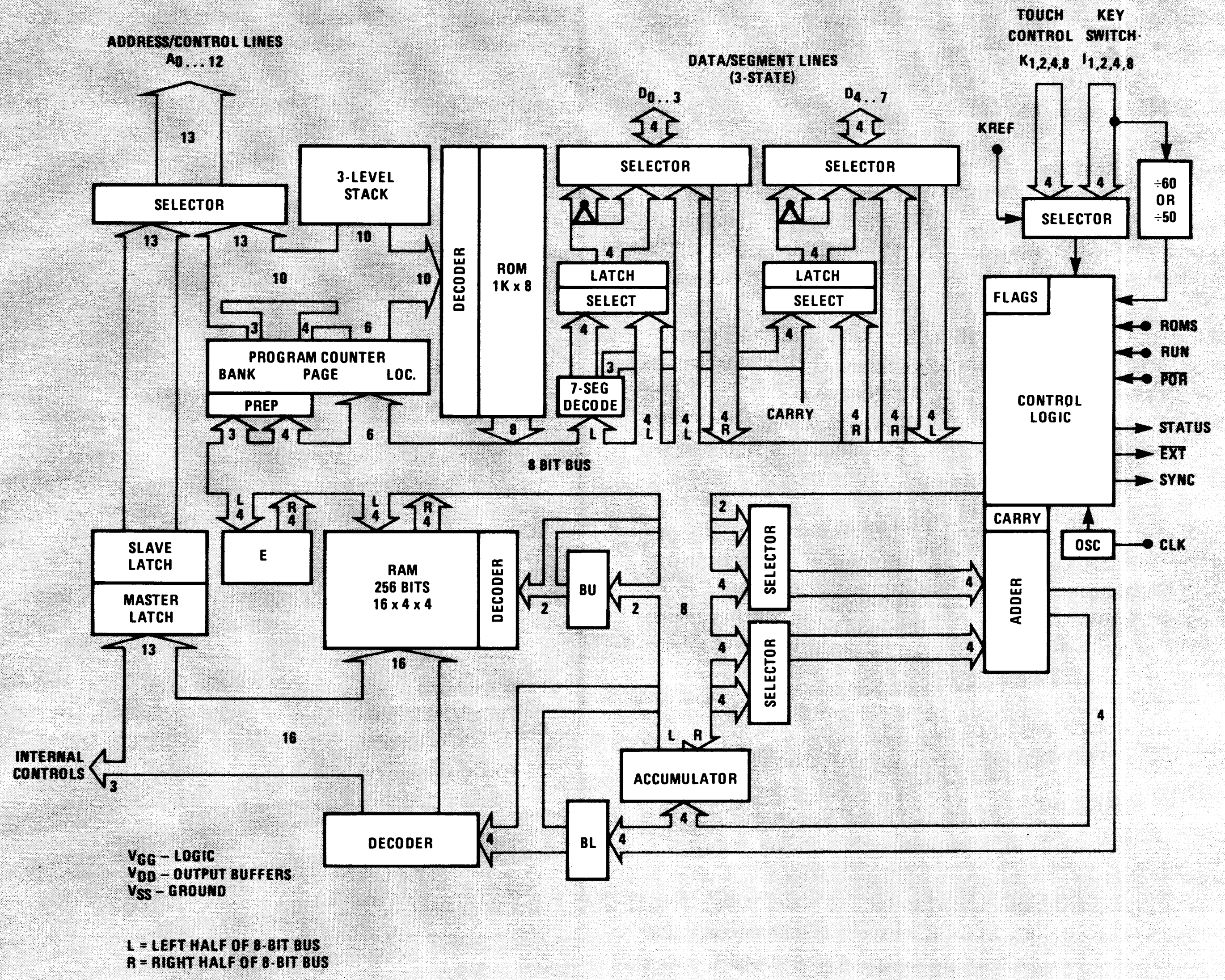


Figure 2.2

AMI S2000 DETAILED BLOCK DIAGRAM



VGG – LOGIC
VDD – OUTPUT BUFFERS
VSS – GROUND

L = LEFT HALF OF 8-BIT BUS
R = RIGHT HALF OF 8-BIT BUS

Figure 2.3

ARITHMETIC UNIT

The 4-bit parallel Adder and 1-bit Carry Register are used for adding, complementing, incrementing, decrementing, comparing and performing Boolean operations. Inputs to the ALU can be from many different sources depending upon the instruction. The ALU deposits its output into the Accumulator and Carry Register. The Carry bit can also be independently set, reset, or tested.

The Accumulator is the main working register and is the principle source and destination for data being operated upon, both internally and for I/O.

I INPUTS AND K INPUTS

The I and K Inputs transfer no data to internal registers. Both of these 4-bit groups can be sensed directly by certain Skip instructions, so they are useful in controlling program logic. Any combination of I inputs together or K inputs together can be sensed simultaneously.

The **I inputs** have internal pullup resistors, and can directly sense keyboards and switches. A timer is driven from a tap off the I8 input, so that a 60Hz or 50Hz input at I8 will cause the timer to set a flag (the "seconds" flag) once per second. The seconds flag can be tested and cleared under program control.

The **K inputs** can be used to implement TouchControl™ capacitive switching or similar analog interfaces, because of the voltage comparator and line-discharge transistors on the chip. K inputs may also be used as conventional keyswitch inputs using external pullup resistors.

SECONDS TIMER AND EUR INSTRUCTION

For time-of-day or interval timer applications, the product's power line frequency is fed to Input I8, where it passes through a Schmitt-trigger, a digital filter, and a ÷60/÷50 counter to the "seconds" flag. An instruction called SOS (Skip On Second) can test and reset the seconds flag to control program logic flow.

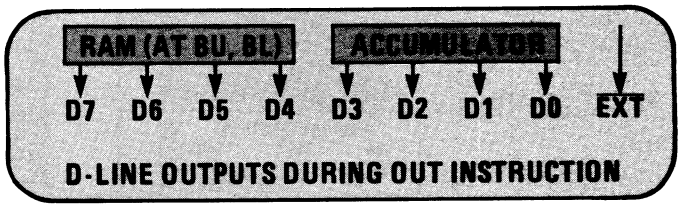
During power-on-reset, the S2000 assumes it is running in a 60Hz environment for the purposes of the SOS instruction. However, a single instruction (EUR) can initiate a switch to 50Hz (or back) as needed.

D-LINES AS INPUTS

The INP instruction inputs 8 bits of data from the bidirectional D Lines. Four bits of the data go to the Accumulator and four bits go to the RAM word addressed by BU and BL. These lines are three-state. An instruction, MVS, is provided for strobing a peripheral device and making the lines available to the peripheral.

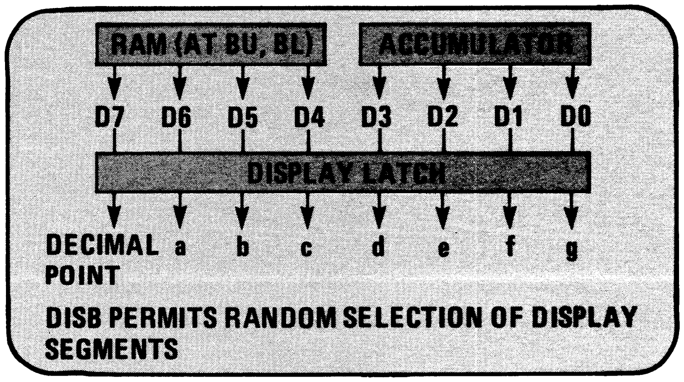
D-LINES AS OUTPUTS

The eight bidirectional three-state D Lines are typically used for display segment drive and 8-bit I/O data transfer. Data outputs are all 8-bit parallel. During the execution of the OUT instruction, contents of the RAM and ACC are directly transferred to the D Lines. Simultaneously the EXT signal is being generated for use by external circuitry as a "data strobe" or the D Lines can all be latched.


RAM (AT BU, BL)
ACCUMULATOR
D7 D6 D5 D4 D3 D2 D1 D0 EXT
D-LINE OUTPUTS DURING OUT INSTRUCTION

For 7-segment display applications, output-latching of the segment data is performed during either a DISN (Display Number) or a DISB (Display Binary) instruction. During a DISN the ACC contents (Carry = Decimal Point) are encoded automatically into the 7-segment hexadecimal codes shown on the next page. All the D Lines are then latched.

During a DISB the contents of the RAM and the ACC are directly loaded into the Display Latch, bypassing the display encoder. This allows arbitrary binary patterns to be displayed.


RAM (AT BU, BL)
ACCUMULATOR
D7 D6 D5 D4 D3 D2 D1 D0
DISPLAY LATCH
DECIMAL POINT a b c d e f g
DISB PERMITS RANDOM SELECTION OF DISPLAY SEGMENTS

DISN INSTRUCTION CODING

SEGMENT-LATCH OUTPUTS, NONINVERTED

ACCUMULATOR VALUE	a	b	c	d	e	f	g	DISPLAY
	D6	D5	D4	D3	D2	D1	D0	
0	1	1	1	1	1	1	0	0
1	0	1	1	0	0	0	0	1
2	1	1	0	1	1	0	1	2
3	1	1	1	1	0	0	1	3
4	0	1	1	0	0	1	1	4
5	1	0	1	1	0	1	1	5
6	1	0	1	1	1	1	1	6
7	1	1	1	0	0	0	0	7
8	1	1	1	1	1	1	1	8
9	1	1	1	1	0	1	1	9
10	1	1	1	0	1	1	1	A
11	0	0	1	1	1	1	1	b
12	1	0	0	1	1	1	0	C
13	0	1	1	1	1	0	1	d
14	1	0	0	1	1	1	1	E
15	1	0	0	0	1	1	1	F

The Carry Bit is output at D7 for decimal point display.
D7 = Carry if normal output polarity is selected using the EUR instruction.

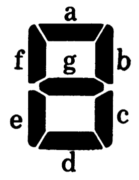


Table 2.1

A LINE OUTPUTS

The A Lines — the 13 Address/Control Outputs — are used both for general-purpose control outputs, for strobing displays and keyboards, and for outputing the contents of the Program Counter.

A set of Master Latches can be set by a sequence of PSH (Preset High) and PSL (Preset Low) instructions, during which the contents of the BL Register indicates which latch is to be affected as shown in Figure 2.4.

Once the strobe pattern is set up, an MVS (Move Strobe) instruction will copy the bits into some Slave Latches and then onto the A Lines. The MVS will also float the D Lines, making them available to the peripheral devices being strobed.

LIGHT-EMITTING DIODE DISPLAYS

The S2000 can directly drive a common-cathode LED display. The D Lines are the source for the LEDs, and the A Lines are the sink. Furthermore, the polarity of both the A Lines and the latches on the D Lines are software-controlled. Thus a wide variety of display types can be accommodated.

I/O STROBE GENERATION

- Use the Preset High and Preset Low instructions to set the Master Latch bits as indicated by BL
- Use the Move Strobe instruction to copy the pattern to the Slave Latch bits and the A Lines
- Repeat the process to terminate the strobes

For example, to strobe Output A8 —

LABEL	OPERATION	OPERAND	COMMENTS
	LAI	8	; value of 8 to Accumulator
	XAB		; exchange with contents of BL
	PSH		; preset Master Latch 8 high
	MVS		; move strobes thru to A Lines
	JMS	DELAY	; call real-time delay routine
	PSL		; preset Master Latch 8 low
	MVS		; terminate strobes on A Lines

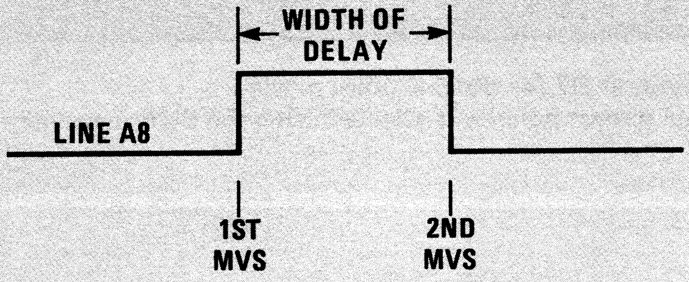


Figure 2.4

3 S2000 INSTRUCTION SUMMARY

REGISTER INSTRUCTIONS

MNEMONIC	OPERAND	DESCRIPTION AND DATA FLOW	
LAB		Load ACC from BL Register	BL → ACC
LAE		Load ACC from E Register	E → ACC

LAI	X	*Load accumulator immediate, select K and I lines. Discharge to V_{SS} the K lines corresponding to the bits of X containing zeroes. For TouchControl applications, see page 56.	X → ACC
			0 ⩽ X ⩽ 15
			SELECT K INPUTS
			SELECT I INPUTS

LBE	Y	*Load BL with E and BU with Y	E → BL Y → BU 0 ⩽ Y ⩽ 3
LBEP	Y	*Load BL with E plus 1 and BU with Y	E + 1 → BL Y → BU 0 ⩽ Y ⩽ 3
LBF	Y	*Load BL with fifteen and BU with Y	15 → BL Y → BU 0 ⩽ Y ⩽ 3
LBZ	Y	*Load BL with zero and BU with Y	0 → BL Y → BU 0 ⩽ Y ⩽ 3

XAB		Exchange ACC and BL Registers	BL ↔ ACC
XABU		Exchange ACC and BU Registers. (The most significant ACC bits are unchanged.)	BU ↔ ACC (0,1)
XAE		Exchange ACC and E Registers	E ↔ ACC

*The first instruction to follow POR may not be an LB-type or an LAI.
Only the first LAI of an LAI sequence is executed, and only the first LB of an LB sequence is executed.

RAM INSTRUCTIONS

In all cases below, MEM refers to the word in RAM addressed by BU, BL.

MNEMONIC	OPERAND	DESCRIPTION AND DATA FLOW	
LAM	Y	Load ACC from memory, then modify BU	MEM → ACC BU ⊕ Y → BU
XC	Y	Exchange ACC with memory, then modify BU	MEM ↔ ACC BU ⊕ Y → BU
XCI	Y	Exchange ACC with memory, then increment BL and modify BU. Skip* if BL = 0 after incrementing.	MEM ↔ ACC BL + 1 → BL BU⊕Y → BU
XCD	Y	Exchange ACC with memory, then decrement BL and modify BU. Skip* if BL = 0 before decrementing (15 after decrementing)	MEM ↔ ACC BL – 1 → BL BU ⊕ Y → BU

The exclusive-OR of BU with Y allows bouncing back and forth between 2 "pages" of RAM without concern for which was the starting page. The same subroutine can manipulate two different pairs of pages.

		Old BU			
		0	1	2	3
	0	0	1	2	3
New BU	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0
		Y			

STM	B	Set bit B in memory to 1	1 → MEM BIT B $0 \leqslant B \leqslant 3$
RSM	B	Reset bit B in memory to 0	0 → MEM BIT B $0 \leqslant B \leqslant 3$

*Skips, when invoked, skip the very next instruction; whenever that skipped instruction is a PP (Prepare Page) however, the next instruction is skipped as well.

ARITHMETIC AND LOGICAL INSTRUCTIONS

In all cases below, MEM refers to the word to RAM addressed by BU, BL.

MNEMONIC	OPERAND	DESCRIPTION AND DATA FLOW	
ADD		Add memory to ACC. Carry unaltered	MEM + ACC → ACC

ADCS		Add memory to ACC with carry and skip* if sum ≤ 15.	MEM + ACC + CY → ACC, CY
			IF SUM ≤ 15, SKIP
			IF SUM > 15, NO SKIP

ADIS	X	Add X to ACC immediate and skip* if sum ≤ 15. Carry unaltered.	X + ACC → ACC
			CARRY UNALTERED
			IF SUM ≤ 15, SKIP
			IF SUM > 15, NO SKIP

AND		Logical AND, memory with ACC.	MEM & ACC → ACC
XOR		Logical Exclusive-OR, memory with ACC.	MEM ⊕ ACC → ACC
CMA		Logical 1's complement accumulator	15 − ACC → ACC
STC		Set carry to 1	1 → CY
RSC		Reset carry to 0	0 → CY

SF1		Set flag 1	1 → F1
RF1		Reset flag 1	0 → F1
SF2		Set flag 2	1 → F2
RF2		Reset flag 2	0 → F2

*Skips, when invoked, skip the very next instruction; whenever that skipped instruction is a PP (Prepare Page) however, the next instruction is skipped as well.

SKIP* INSTRUCTIONS

MNEMONIC	OPERAND	DESCRIPTION AND DATA FLOW	
SAM		Skip if ACC = memory at BU, BL.	If ACC = MEM, SKIP
SZM	B	Skips if the RAM word addressed by BU, BL contains a 0 in bit B.	IF $MEM_B = 0$, SKIP $0 \leqslant B \leqslant 3$
SBE		Skip if BL = E.	IF BL = E, SKIP
SZC		Skip if carry = 0.	IF CY = 0, SKIP
SOS		Skip on 'seconds' Flag (i.e., timer output). If 'seconds' Flag = 1, skip and reset 'seconds' Flag.	IF SF = 1, $0 \rightarrow$ SF and SKIP
SZK		Skip if zero in K input. The bits in the argument of the last executed LAI instruction select the corresponding K inputs and discharge the others. SZK skips if selected K inputs are 0 (i.e., < KREF). For TouchControl applications, see page 56.	$K_{8,4,2,1} = 0$, SKIP
SZI		Skip if zero in I input. The bits in the last executed LAI instruction select the corresponding I inputs. The instruction skips if selected I input is 0.	IF $I_{8,4,2,1} = 0$, SKIP
TF1		Test flag 1, skip if set.	IF F1 = 1, SKIP
TF2		Test flag 2, skip if set.	IF F2 = 1, SKIP

*Skips, when invoked, skip the vcry next instruction; whenever that skipped instruction is a PP (Prepare Page) however, the next instruction is skipped as well.

PROGRAM CONTROL INSTRUCTIONS

MNEMONIC	OPERAND	DESCRIPTION AND DATA FLOW	

PP	Y	Prepare Page or Bank Register. If previous instruction was not a PP, Y → Prepare *Page* Register. (PPR)
		If previous instruction was a PP, Y → prepare *Bank* Register. If a PP is skipped the following instruction will also be skipped. Any number of PP's plus the following JMP or JMS may be skipped in this fashion.
			IF PREV INSTR ≠ PP, Y → PPR
			$0 \leqslant Y \leqslant 15$
			IF PREV INSTR = PP, Y → PBR
			$0 \leqslant Y \leqslant 7$

JMP	X	Jump to location X within present Page and Bank.	X → LR $0 \leqslant X \leqslant 63$
		If the previous instruction was a PP then load the Page and Bank Register from their respective prepare registers and jump to location X in the new Page and Bank.	X → LR PPR → PR PBR → BR
		Example: To transfer to a label LL in the same page, use JMP LL To transfer to a label LP in a different page, use PP LP/64 JMP LP To transfer to a label LB in a different bank, use PP LB/64 PP LB/1024 JMP LB	
JMS	X	Jump to location X on page 15. Save PR and LR + 1 in program stack.	LR + 1 → L STACK PR → P STACK X → LR 15 → PR
		Exception: if previous instruction was a PP, jump to PBR, PPR, X.	LR + 1 → L STACK PR → P STACK X → LR PPR → PR PBR → BR

PROGRAM CONTROL INSTRUCTIONS (Cont.)

MNEMONIC	OPERAND	DESCRIPTION AND DATA FLOW	
RT		Return from a subroutine.	L STACK → LR P STACK → PR
RTS		Return from a subroutine, and skip* the next instruction.	L STACK → LR P STACK → PR SKIP
NOP		No operation.	

*All skips (including XCI, XCD, ADCS, and ADIS) automatically ignore PP instructions and skip the next *non-PP* instruction.

Note: When executing an INP instruction, the D-Bus must be in the floating state, or else the Display Latch contents will be input to the RAM and Accumulator.

Note: The Bank Register is not saved on the stack.

Note: It is common practice to put most subroutines in page 15. If there are too many to fit, a subroutine may be placed in some other page, and a jump to it is placed in page 15; alternately, a PP-JMS sequence may be used.

INPUT/OUTPUT INSTRUCTIONS

In all cases below, MEM refers to the word in RAM addressed by BU, BL.

In all cases involving 8-bit data transfers, the low-ordered bits are in the accumulator and the high-ordered bits are in RAM.

MNEMONIC	DESCRIPTION AND DATA FLOW
INP	Input data from D Lines to ACC and memory, if D-bus is floating. Otherwise, transfer Display Latch to ACC and memory. $D_{3-0} \rightarrow$ ACC, $D_{7-4} \rightarrow$ MEM
OUT	Output data to D Lines from ACC and memory. An EXT Pulse is generated at T7 time. The display latch remains unchanged. ACC $\rightarrow D_{3-0}$, MEM $\rightarrow D_{7-4}$
DISB	Display binary data from ACC and memory. Exit from floating mode on D Lines. The display latch outputs may invert, depending upon the last executed EUR instruction. ACC $\rightarrow$ DISPLAY LATCH (3-0) $\rightarrow D_{3-0}$ MEM $\rightarrow$ DISPLAY LATCH (7-4) $\rightarrow D_{7-4}$
DISN	Display number in ACC, decimal point from Carry. Exit from floating mode on D Lines. The display latch outputs may invert, depending upon the last executed EUR instruction. ACC $\rightarrow$ SEGMENT DECODE $\rightarrow$ DISPLAY LATCH (6-0) $\rightarrow D_{6-0}$ CARRY $\rightarrow$ DISPLAY LATCH $(_7) \rightarrow D_7$
MVS	Move A-Line Master-Strobe-Latch to A Lines. Enter floating mode on D Lines. MASTER STROBE $\text{LATCH}_{12-0} \rightarrow A_{12-0}$

PSH	Preset high the Master-Strobe-Latch addressed by BL
	if $0 \leqslant BL \leqslant 12$: SET LATCH BIT (BL) HIGH.
	if BL = 13 : SET MULTIPLEX OPERATION.
	if BL = 14 : EXIT D LINES FROM FLOATING MODE
	if BL = 15 : SET ALL MASTER STROBE LATCH BITS HIGH.

PSL	Preset low the Master-Strobe-Latch addressed by BL
	if $0 \leqslant BL \leqslant 12$: SET LATCH BIT (BL) LOW.
	if BL = 13 : SET STATIC OPERATION.
	if BL = 14 : FLOAT D LINES.
	if BL = 15 : SET ALL MASTER STROBE LATCH BITS LOW

EUR	(European) — Set 50/60 HZ and display latch polarity. ACC BIT 0 = 1 : NORMAL POLARITY IN D LINES (POWER-UP SETTING) = 0 : INVERTED POLARITY IN D LINES ACC BIT 3 = 1 : 50 HZ OPERATION = 0 : 60 HZ OPERATION (POWER-UP SETTING)

INSTRUCTION CODES AND TIMING

MNEMONIC	CYCLES	HEX	BINARY
NOP	1	00	00000000
RT	1	02	00000010
RTS	1 + n ≥ 2	03	00000011
PSH	1	04	00000100
PSL	1	05	00000101
AND	1	06	00000110
SOS	1 + n	07	00000111
SBE	1 + n	08	00001000
SZC	1 + n	09	00001001
STC	1	0A	00001010
RSC	1	0B	00001011
LAE	1	0C	00001100
XAE	1	0D	00001101
INP	1	0E	00001110
EUR	1	0F	00001111
CMA	1	10	00010000
XABU	1	11	00010001
LAB	1	12	00010010
XAB	1	13	00010011
ADCS	1 + n	14	00010100
XOR	1	15	00010101
ADD	1	16	00010110
SAM	1 + n	17	00010111
DISB	1	18	00011000
MVS	1	19	00011001
OUT	1	1A	00011010
DISN	1	1B	00011011
SZM B	1 + n	1C to 1F	000111XX
STM B	1	20 to 23	001000XX
RSM B	1	24 to 27	001001XX
SZK	1 + n	28	00101000
SZI	1 + n	29	00101001
RF1	1	2A	00101010
SF1	1	2B	00101011
RF2	1	2C	00101100
SF2	1	2D	00101101
TF1	1 + n	2E	00101110
TF2	1 + n	2F	00101111
XCI Y*	1 + n	30 to 33	001100XX
XCD Y*	1 + n	34 to 37	001101XX
XC Y*	1	38 to 3B	001110XX
LAM Y*	1	3C to 3F	001111XX
LBZ Y	1	40 to 43	010000XX
LBF Y	1	44 to 47	010001XX
LBE Y	1	48 to 4B	010010XX
LBEP Y	1	4C to 4F	010011XX
ADIS X	1 + n	50 to 5F	0101XXXX
PP X*	1	60 to 6F	0110XXXX
LAI X	1	70 to 7F	0111XXXX
JMS X	2	80 to 8F	10XXXXXX
JMP X	1	C0 to FF	11XXXXXX

*Assembled code should contain the complements of these arguments: the AP Assembler complements them automatically

n = number of instructions skipped

Note: OP code 01 is reserved for use with the development tools as a breakpoint.

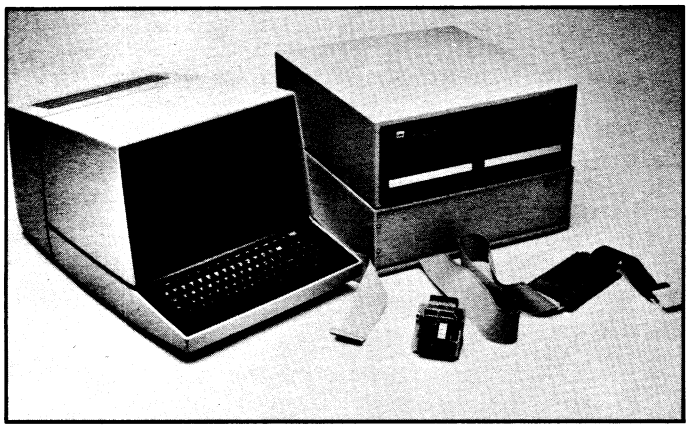
S2000 INSTRUCTIONS

0000	0000	NOP		NO OPERATION
11XX	XXXX	JMP	X	JUMP TO LOCATION X, X → LR (IF PREVIOUS INSTRUCTION A PP PPR → PR, PBR → BR)
10XX	XXXX	JMS	X	JUMP TO SUBRTN AT X, LR + 1 → L STACK, PR → P STACK, X → LR
0000	0010	RT		L STACK → LR, P STACK → PR
0000	0011	RTS		L STACK → LR, P STACK → PR, THEN SKIP INSTRUCTION
0110	XXXX	PP	X*	X → PP (IF ONE PP), X → PR (IF MORE THAN 1 PP)
0000	1001	SZC		SKIP IF CARRY = 0
0001	11XX	SZM	Z	SKIP IF RAM BIT Z = 0
0000	0111	SOS		SKIP IF 'SEC FF' = 1, 0 → 'SEC FF' (IF SKIP TAKEN)
0010	1000	SZK		SKIP IF K BIT(S) = 0, (BITS IN LAST LAI)
0010	1001	SZI		SKIP IF I BIT(S) = 0, (BITS IN LAST LAI)
0000	1000	SBE		SKIP IF BL = E
0001	0111	SAM		SKIP IF ACC = RAM
0010	1110	TF1		SKIP IF = 1
0010	1111	TF2		SKIP IF = 1
0001	0100	ADCS		RAM + ACC + CARRY, SKIP IF SUM ≤ 15
0101	XXXX	ADIS	X	X + ACC, SKIP IF SUM ≤ 15
0001	0110	ADD		ACC + RAM W/O CARRY
0000	0110	AND		ACC & RAM
0001	0101	XOR		ACC ⊕ RAM
0000	1010	STC		1 → CARRY
0000	1011	RSC		0 → CARRY
0001	0000	CMA		$\overline{\text{ACC}}$
0010	1011	SF1		1 → FLAG1
0010	1010	RF1		0 → FLAG1
0010	1101	SF2		1 → FLAG2
0010	1100	RF2		0 → FLAG2
0111	XXXX	LAI	X	X → ACC
0001	0010	LAB		BL → ACC
0000	1100	LAE		E → ACC
0001	0011	XAB		BL ↔ ACC
0001	0001	XABU		BU ↔ ACC
0000	1101	XAE		ACC ↔ E
0100	10XX	LBE	Y	Y → BU, E → BL
0100	00XX	LBZ	Y	Y → BU, 0 → BL
0100	01XX	LBF	Y	Y → BU, 15 → BL
0100	11XX	LBEP	Y	Y → BU, E + 1 → BL
0011	11XX	LAM	Y*	RAM → ACC, BU⊕Y → BU
0011	10XX	XC	Y*	RAM ↔ ACC, BU⊕Y → BU
0011	00XX	XCI	Y*	ACC ↔ RAM, BL + 1 → BL, BU⊕Y → BU SKIP IF BL = 0 (AFTER INC.)
0011	01XX	XCD	Y*	ACC ↔ RAM, BL - 1 → BL, BU⊕Y → BU SKIP IF BL = 0 (BEFORE DEC.)
0010	00XX	STM	Z	1 → RAM BIT Z
0010	01XX	RSM	Z	0 → RAM BIT Z
0000	1110	INP		D0-D3 → ACC, D4-D7 → RAM
0001	1010	OUT		ACC → D0-D3, RAM → D4-D7
0001	1011	DISN		ACC → SEGMENT DECODER → DISPLAY LATCH, CARRY → D7
0001	1000	DISB		ACC → D0-D3 → DISPLAY LATCH, RAM → D4-D7 → DISPLAY LATCH
0001	1001	MVS		A-LINE MASTER LATCH → SLAVE LATCH
0000	0100	PSH		PRESET HIGH [BL] → MASTER STROBE LATCH
0000	0101	PSL		PRESET LOW [BL] → MASTER STROBE LATCH
0000	1111	EUR		(EUROPEAN) SET 50/60HZ AND DISPLAY LATCH OUTPUT POLARITY

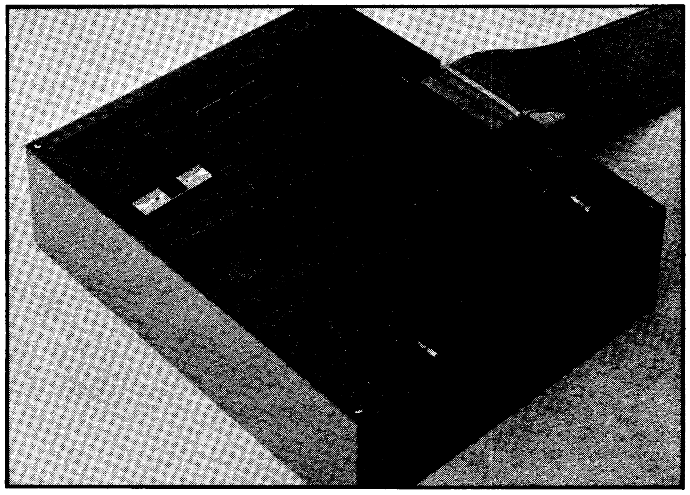
* ASSEMBLED CODE CONTAINS COMPLEMENT OF THOSE ARGUMENTS (AP DOES IT FOR YOU)

S2000 DEVELOPMENT TOOLS

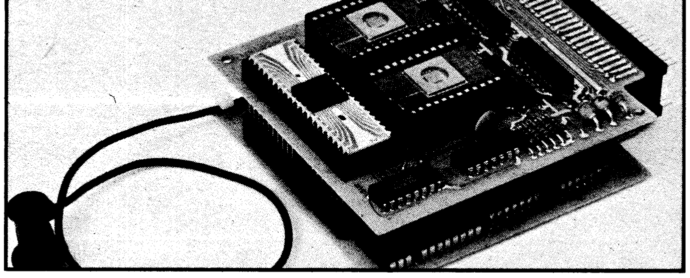
AMI Microcomputer Development Center



AMI's MDC with specialized S2000 components provides a complete software development center and hardware prototyping facility.



The DEV-2000 Development & Debug Module links the user's prototype to the MDC. It gives quick-turnaround, on-line debug of hardware and software, including trace, step and macro capabilities.



The SES-2000 emulator board, a pin-for-pin substitute for an S-2000 chip, provides program storage on UV erasable PROMs.

4 S2000 DEVELOPMENT TOOLS

DEVELOPMENT SYSTEM OVERVIEW

The S2000 single-chip microcomputer is fully supported by a proven array of sophisticated development aids:

	HARDWARE	SOFTWARE Written in 6800 assembly language for operation on an MDC
GENERAL-PURPOSE can also be used for development of 6800 microprocessor systems	Microcomputer Development Center (MDC) MDC-140 Logic Analyzer Line Printer	FDOS-II — disk operating system ED — text editor LA — logic analyzer P6834 — EPROM programmer
SPECIALIZED for S2000 developments	DEV-2000 Development Board SES-2000 Emulator Module TES-2000 Functional Tester	AP — assembler LD — loader DB — debugger SM — software simulator 68.F — LA display formats ALL — macros for DB and SM 2KF — skeleton program Cross-assemblers, editor and simulator/debug software are also available on a major timesharing service.

MICROCOMPUTER DEVELOPMENT CENTER (MDC)

AMI's MDC is a fully equipped, disk-based microcomputer development facility, complete with FDOS-II Floppy-Disk-Operating and File Management System. Controlled from its CRT terminal, MDC provides instant access to program and data files resident on removable diskettes. It comes complete with RS-232 interface, current-loop interface, EPROM burner, and self-diagnostics.

DEV-2000

The DEV-2000 Development Module provides quick-turnaround on-line debug of your prototype S2000 system hardware and software, by linking your prototype system with the MDC. It comes fully assembled including an S2000 chip and a TTL-compatible 40-pin DIP connector that plugs right into your S2000 socket. When running DB (the S2000 Debugger Program) on the MDC, you can fully and easily control and interrogate the S2000 chip on the DEV-2000. This sophisticated Debugger permits full execution control — including single-step, N-step, and breakpoint modes — over your S2000 prototype hardware. You can jam-load all the S2000's registers, trace their behavior, and get an output listing of everything that appears on the CRT. And you can work in a higher level of language by using the Debugger's Macro capability and its accessibility to special-purpose 6800 subroutines.

SES-2000

The SES-2000 Emulator Module acts like an S2000 microcomputer with erasable program memory. This compact unit comes fully assembled with an S2000 chip and two S6834 EPROMs, which can be erased by ultraviolet light and electrically reprogrammed. SES-2000 offers real-time execution at a low cost.

TES-2000

The TES-2000 is a dedicated S2000 tester which allows functional go/no-go comparison of CPU and ROM against those of a model S2000 chip.

MDC-140 LOGIC ANALYZER

The MDC-140 Logic Analyzer is an advanced debug tool connected as a peripheral device of the AMI Microcomputer Development Center (MDC). Features include:

- Captures 1024 Events of 40 Parallel Inputs
- Captures Data Under Control of Programmable Start on Data Content
- Delay of -1024 to +64K Clock Periods
- Setup and Display of Captured Data Under Control of MDC Software
- Display Format is User-Definable; Captured Data Can Be Displayed in a Mix of Hex, Octal, Binary, ASCII and Special Formats for Support of S6800, S6820, S2000, 8080, etc.
- Four Clock Sources
- Input Voltage Range = -15 to +15 volts
- Adjustable Input Thresholds
- Data-Dependent Output for Triggering an Oscilloscope

CUSTOMER ASSISTANCE

AMI's S2000 Applications Engineering staff are available for consultation regarding all aspects of S2000 usage. AMI's staff is also available to discuss any special modifications to the S2000 for high volume applications.

5 MACRO ASSEMBLER

INTRODUCTION

This section defines the functional characteristics of the AMI S2000 Assembler Program (AP). AP is an MDC disk resident assembler which processes S2000 Assembly Language programs, translating mnemonic operation codes to the binary codes needed in machine instructions for the S2000 microcomputer. Its inputs are a Source Program File and optionally a Macro Library File. Its outputs are (1) a formatted assembly listing of the program which includes notification of errors detected during the assembly process, and (2) a binary file of the machine instructions and data assembled. The significant features of the AP are:

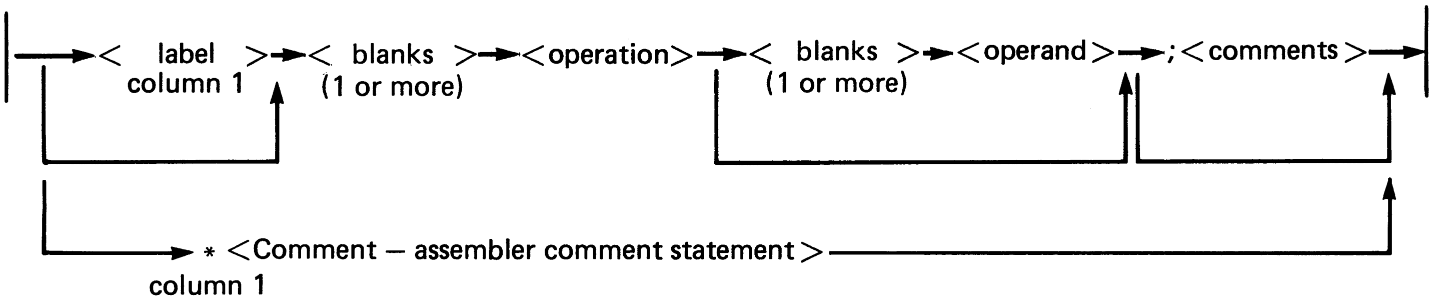
- Symbolic representation of instruction operation codes and memory addresses
- Predefined symbols for program constants and data
- Operand expressions
- Macros
- Conditional assembly options

FUNCTION

AP processes an S2000 source program as a sequence of statements defining machine instructions, program data or various assembler directives. Depending on the operating system environment, sections of the source program may reside in other files and may be included in the assembly, and Macros may be locally defined in a program or may reside in a Macro Library and be included in the assembly. Typically a unique source program will be created with the MDC Text Editor (ED) and assembled with predefined Macros. FDOS-II only supports a single input file; MERGE can be used to incorporate a Macro Library.

The binary words into which statements are translated are normally placed in successively increasing memory locations when loaded by the S2000 Loader (LD). The assembler keeps an internal location counter to point to where each newly translated code is to be placed. This location counter is accessible as an operand and modifiable. The location counter is defined as a binary sequence for the generation of program data.

ASSEMBLER STATEMENT



SOURCE PROGRAMS

A source program consists of two or more statements of the form:

LABEL	OPERATION	OPERAND	COMMENT	end-of-statement

A statement must contain at least one of these fields, otherwise it is a blank line generating no code.

(Note that throughout this chapter example listings generated by the AP assembler program may include features not described until later sections.)

```
0015
0016
0017         * THIS SAMPLF COMMENT (ASTERISK) BEGINS LISTING IN THE LABEL FIELD.
0018
0019                                     ;THESE SAMPLE COMMENTS, BEGINNING
0020                                     ;WITH A SEMICOLON, BEGIN LISTING
0021                                     ;IN THE COMMENT FIELD.
0022
0023
0024 0000 00           NOP               ;NO LABEL OR OPERAND
0025
0026 0001 00 ONE       NOP               ;GLOBAL LABEL 'ONE'
0027
0028 0002 00 :ONE      NOP               ;LOCAL LABEL ':ONE'
0029
0030                                     ;LABELS HAVE THE VALUE
0031                                     ;OF THE CURRENT LOCATION
0032                                     ;COUNTER.
0033
0034 0003 73           LAI    3          ;NO LABEL, OPERAND=CONSTANT
0035
```

NOTE: No embedded blank characters within label or operation fields; blanks (one or more) separate the operation field. Semicolon begins comment field.

Each statement may (1) define one or more machine language instructions, or (2) invoke one of the many assembler directives. Which function is performed is defined by the OPERATION field thus defining the Statement Type. The first statement of the program should be a TITLE statement (optional) and the last statement must be an END statement.

```
A.M.I. S2000 ASSEMBLER        USER'S GUIDE SAMPLES             PAGE 0001
VERSION 1.3

STMT LOC OBJ SOURCE

0001
0002
0003                   TITLE  USER'S GUIDE SAMPLES
0004
```

FIELDS

Statements consist of four fields, namely: the Label field, the Operation field, the Operand field and the Comment field. The Comment field begins with the character ';' and terminates at the end-of-statement character (eos), carriage return. A full comment record may begin with an asterisk (*) in position one.

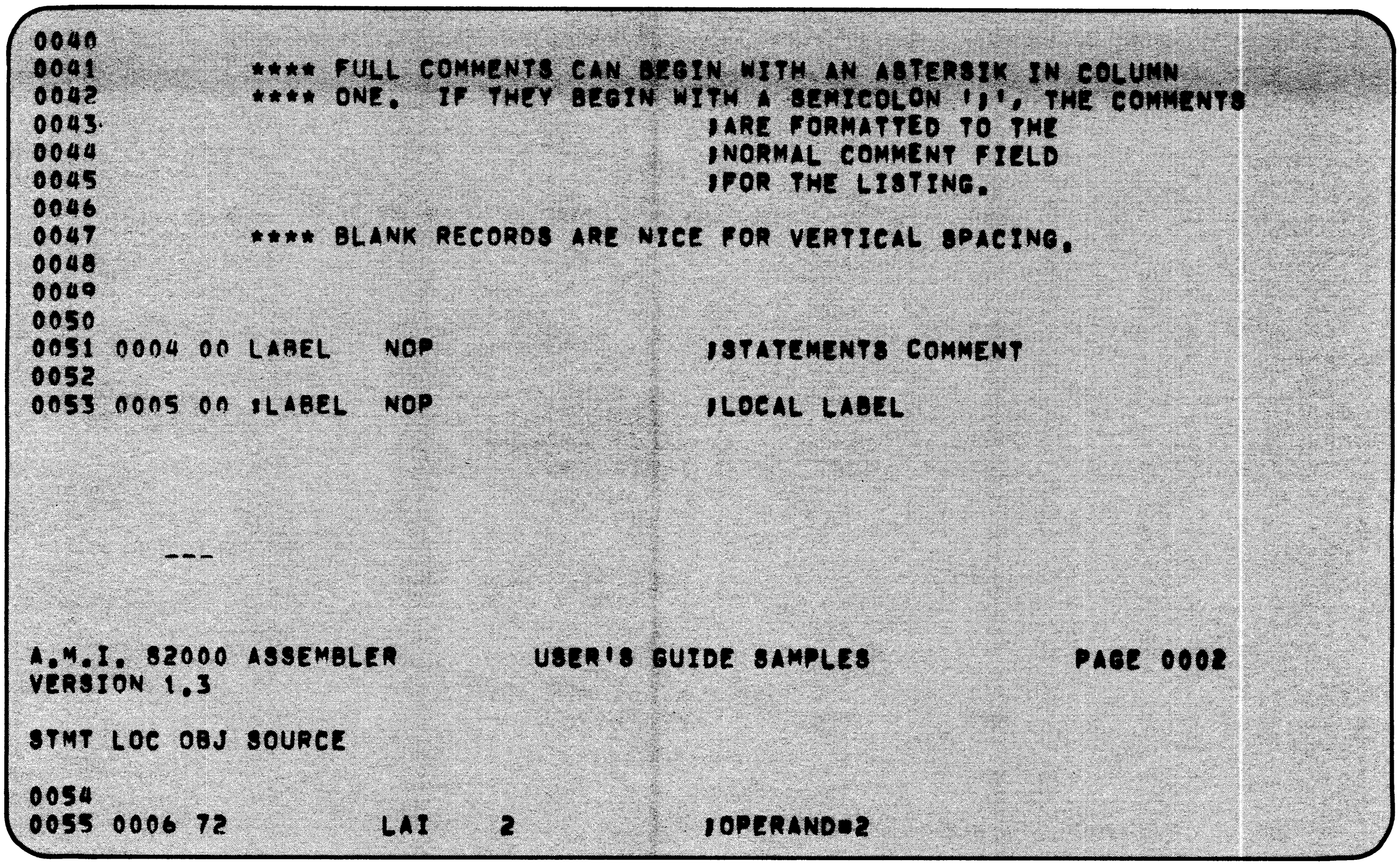
The Label field is present when the **first** character of the statement is non-blank and not (*) or (;).

The Label field terminates at the occurrence of the first blank.

The Operation field begins at the first non-blank character following the Label field and terminates at the occurrence of the first blank, the Comment field or the end-of-statement.

The Operands field begins at the first non-blank character following the Operation field and terminates at the Comment field or end-of-statement.

The following examples detail the statement structure:



```
0040
0041         **** FULL COMMENTS CAN BEGIN WITH AN ASTERSIK IN COLUMN
0042         **** ONE.  IF THEY BEGIN WITH A SEMICOLON ';', THE COMMENTS
0043                                    ;ARE FORMATTED TO THE
0044                                    ;NORMAL COMMENT FIELD
0045                                    ;FOR THE LISTING.
0046
0047         **** BLANK RECORDS ARE NICE FOR VERTICAL SPACING.
0048
0049
0050
0051 0004 00 LABEL   NOP                ;STATEMENTS COMMENT
0052
0053 0005 00 :LABEL  NOP                ;LOCAL LABEL




      ---


A.M.I. S2000 ASSEMBLER       USER'S GUIDE SAMPLES          PAGE 0002
VERSION 1.3

STMT LOC OBJ SOURCE

0054
0055 0006 72         LAI    2           ;OPERAND=2
```

Label Field

The Label field (when present) defines a symbol known as a label which "labels" either

1) a unique location in the S2000 memory space, or
2) a macro in a MACRO definition statement, or
3) a variable value as the result of an assignment statement (SET), or
4) an absolute data value (EQU).

The form of a label is a single letter [A . . Z], or a letter followed by a sequence of letters or digits [0 . . 9], six characters maximum.

If the first character of a label is a colon (:), the symbol is called a local symbol and is defined only within a local region. This allows the same symbol (e.g. :LOOP) to be defined more than once; each definition must be in a separate local region and each definition is treated by the assembler as unique.

The directive LOCAL is used to define the beginning of a new local region and the end of an old one. A local region may be nested within another, giving up to 8 levels of nesting, using the directives LBEG and LEND. LBEG begins, and LEND ends, a local region nested within a larger region.

```
               * <<<<<<< LOCAL LABELS >>>>>>>>


                           LOCAL
     0007 00               NOP                       ;ANY CODE REPRESENTED BY * * *

               *     * * *

     0008 00   $MORE       NOP

               *     * * *

     0009 29               SZI                       ;TEST AND JUMP
     000A CB               JMP     $MORE             ;BACKWARD

               *     * * *

                           LOCAL                     ;NEW REGION

               *     * * *

     000B 29               SZI                       ;TEST AND JUMP
     000C CD               JMP     $MORE             ;FORWARD

               *     * * *

     000D 00   $MORE       NOP

               *     * * *

     000E 00   MORE        NOP                       ;GLOBAL LABEL (NO $)
                                                     ;CAN ONLY BE DEFINED ONCE
                                                     ;AND IS AVAILABLE TO ALL
                                                     ;REGIONS.
```

Operation Field

The Operation field defines the statement type and may be optionally omitted in which case the statement is a full comment statement. The operation 'mnemonic' is a symbol which defines a machine instruction or assembler directive. Table 3-2 is a list of the predefined instruction mnemonics and Table 5-1 summarizes the assembler directives. The different instruction mnemonics may define variations of the operand field, since some instructions permit or require different formats of operands. Assembler directives are used to assign a value to a label, to define a macro, to generate object code data or to control the assembly process.

Operand Field

The Operand field is the third field of the source statement. It is used to define the instruction operands, parameters for data in macro calls and assembler directives. This field is required by some statement types, and not by others. The operand field may contain any number of sub-fields dependent upon the statement type. The range of an operand is specifically defined by its operation mnemonic. WHERE REQUIRED, OPERAND COMPLEMENTATION IS DONE AUTOMATICALLY, (e.g. PP instruction).

Comment Field

The Comment field is the last field and is signalled by the occurance of a ';' which is not embedded in any text string of an operand. The comment is optional but highly recommended for good program documentation. It may contain any characters and is terminated by the end-of-statement character.

SELF-DEFINING TERMS – OPERAND FIELD

Numeric and character data is required frequently in the operand field of the source statement. The character representations of these data are predefined within the assembler domain and are known as self-defining terms. All self-defining terms are, by definition, non-negative. They may be formed into negative values by prefixing with the unary operator '–'. Three types of self-defining terms are available; decimal, hexadecimal, and current location counter (*).

Decimal

Decimal radix is assumed; therefore, to specify a decimal number, one need only write the number. The range of decimal number is zero through 65535.

Examples: 32671   10   5   600

Hexadecimal

A hexadecimal self-defining term is preceded by a $ (dollar sign). The range of a hexadecimal number is zero to $FFFF_{16}$.

Examples: $A2 $80 $F

Current Location Counter

An asterisk (*) in an expression (where an operand is expected) is replaced by the current value of the location counter.

OPERAND EXPRESSIONS (See table, page 33)

The simple operand terms previously described (labels, numbers, location counter) may be combined with arithmetic and logical operators to produce complex expressions.

```
             *  <<<<<<< OPERAND EXPRESSIONS >>>>>>>>



     0107              ORG     $107           ;HEXADECIMAL CONSTANT
     0107      LIS     EQU     *              ;SYMBOL * HAS VALUE
                                              ;OF CURRENT LOCATION COUNTER.

     0107 C4           JMP     *-3            ;JUMP BACK

     0207              ORG     LIS+$100

     0207 68           PP      ENTRY1/64      ;CALCULATE PAGE
     0208 CC           JMP     ENTRY1

     01C0      PGFA    EQU     ENTRY1 & $3C0 ;PAGE ADDR, OTHER
                                              ;BITS MASKED.

     0007      PGEB    EQU     (ENTRY1 &$3C0)/64 ;PAGE ADDR SHIFTED.

     0027      MSK1    EQU     1 ! 2 ! 4 ! 32 ;LOGICAL OR

     0209 73           LAI     3 & MSK1

```

Expressions are evaluated according to these rules:

1. Portions within parenthesis are evaluated first.
2. Unless overridden by parenthesis, operations with higher precedence are done first.
3. If neither parenthesis or precedence supply an order, operations are performed from left to right.
4. Numbers may range from –32768 to 32767 (16 Bits), integer values only. Non-integer division results are truncated. Negative numbers are represented in two's complement form.

Sample uses of expressions:

1. Loop until Key Input NKEY is zero.

```

0001      NKEY1    EQU     1
0002      NKEY2    EQU     2

          * TEST FOR MULTIPLE KEYS

020A 73  $LOOP    LAI     NKEY1+NKEY2
020B 2A           SZK
020C CA           JMP     $LOOP

```

2. Prepare Page to the current memory page.

```

020D 6A           PP      ENTRY1/64
020E CC           JMP     ENTRY1
```

3. Determine whether or not label L is in the current page, and produce a prepare page to L's page if it is not.

```
          * TEST IF PREDEFINED LABEL IS IN CURRENT PAGE
FFFF               IF      (* & $3C0) <>  (LABEL & $3C0) ;TRUE=NON ZERO
020F 6F            PP      LABEL/64
                   IEND

          * NOTE ALL SYMBOLS IN EXPRESSION OF 'IF'
          * MUST BE DEFINED BEFORE 'IF', THEREFORE
          * ABOVE CONSTRUCTION IS NOT VERY USEFUL.
```

4. Set location to beginning of next page.

```
          * SET LOCATION TO DESIRED PAGE BOUNDARY

00C0               ORG     3*64          ;PAGE 3

0100               ORG     (*/64 + 1)*64 ;NEXT PAGE, '*'
                                         ;CAN BE CURRENT LOCATION
                                         ;OR MULTIPLICATION OPERATOR.

0140               ORG     (*/64 + 1)*64 ;NEXT
0180               ORG     (*/64 + 1)*64 ;NEXT
```

Note that the assembled operand values are truncated depending on the length required for the particular instruction (and complemented if required).

Note: For explanation of Assembler Directives, e.g., EQU, ORG, see index on page 44.

TABLE OF OPERATORS

Symbol	Precedence	Description
–	6	Unary Minus.
∧	6	Logical Not. Complements all 16 bits of the number.
*	5	Multiplication.
/	5	Division.
+	4	Addition.
–	4	Subtraction.
<	3	Less Than. All relational operators produce $FFFF if the relation is true, and 0 if the relation is false.
>	3	Greater Than.
< =	3	Less Than or Equal.
> =	3	Greater Than or Equal.
=	3	Equal.
< >	3	Not Equal.
&	2	Logical And. Each bit of the result is the And of the corresponding bits from the initial values.
!	1	Logical Or.
#	1	Exclusive Or.

Note: An asterisk ("*") has three interpretations: (1) the beginning of a comment; (2) the value of the present location counter contents; or (3) the multiplication operator. Its context determines its meaning.

ASSEMBLER DIRECTIVES

The assembler directives enable the programmer to control the assembly process. The directives appear in the operation field. They are used to allocate working memory space, assign values to data, control the sequencing of source programs through the assembler and control the format of the assembly listable output. Assembler directives used as pairs (MACRO/MEND, IF/IEND) are described in the following sections.

Heading Statements

A TITLE statement is optional. The label is ignored if present. The Operand field will be placed at the top of each page of listable output. A PAGE statement can be used to force a new listing page and add a secondary title from its operand field.

Source Program Terminations

END is the last statement of the program and terminates the assembly process. The assembler is a two pass assembler. At the end of the first pass, the input file pointer is reset to point to the first record of the file and the second pass is begun. At the end of the second pass, the assembly is complete.

Vertical Spacing

Blank statements (one or more spaces followed by carriage return) are used for vertical spacing.

Variable Value Definition

The SET statement defines the symbol in the Label field, giving it the value of the expression in the Operand field. Symbols defined with a SET statement may be redefined. Symbols previously defined in the Label field of **any** statement other than a SET statement may not be redefined with a SET statement. All terms in the expression must eventually be defined. The range of the expression is –32768 to 32767.

Among the more useful functions of the SET statement is the saving of the current value of the location counter for later usage.

```
          * <<<<<<<<< VARIABLES >>>>>>>


          * SET SYMBOLS CAN BE REDEFINED WITHOUT ERROR

     0180 CLOC     SET     *

     0007 CLOC     SET     7

     0009 SWITCH   SET     CLOC+2

     000B SWITCH   SET     11
```

Location Counter Control

The ORG statement permits the programmer to direct the binary machine code of the translated statements to specific sequential locations within the S2000 8K memory space. The location counter is set to the value of the expression. All terms used in the expression must be defined. The value of the expression must be in the range of 0 to $1FFF_{16}$.

```
A.M.I. S2000 ASSEMBLER          USER'S GUIDE SAMPLES              PAGE 0005
VERSION 1.3                     SECONDARY HEADING

STMT LOC OBJ SOURCE

0175
0176
0177           *
0178           * PROGRAM RESET ADDRESS IS ZERO
0179           *
0180 0000              ORG     0             ;DECIMAL
0181 0000              ORG     $0            ;HEXADECIMAL
0182 0000              ORG     *-*           ;EXPRESSION
0183
0184           *
0185           * SUBROUTINES USUALLY GO IN PAGE 15
0186           *
0187 03C0              ORG     960           ;DECIMAL
0188 03C0              ORG     $3C0          ;HEX
0189 03C0              ORG     15*64         ;CONVENIENT EXPRESSION
0190
0191 01CC              ORG     12 + 7*64     ;MIDDLE OF PAGE 7
```

Symbol Value Definition

The EQU statement defines the symbol in the Label field, giving it the value of the expression in the Operand field. A symbol once defined in the Label field of any statement other than a SET statement may not be reused in the Label field of an EQU statement.

All terms in the expression must be defined. The range of the expression is –32768 to 32767.

```
0194
0195 01CC      ENTRY1  EQU    *
0196
0197 0008      LENGTH  EQU    LAB2 - LAB1
0198
0199 001F      CONS1   EQU    31
0200
0201 0093      MSK3    EQU    ( (MSK + 7 ) ! CONS1 ) & $F3
0202
0203 01CC      LAB1    EQU    *
0204
0205 01D4      LAB2    EQU    * + 8
0206
0207 0080      MSK     EQU    $80
```

Data Generation/Reservation

The assembler recognizes four directives for data generation and memory reservation, namely: FCB, FDB, FCC, and RMB. These directives are included for future extensions to the processor. They are of limited utility for the S2000. The Operand field is required for each of these directives. The Label field is optional, and if present, the symbol is assigned the value and mode of the location of the first byte of the data or reserved space.

FCB

The Form-Constant-Byte directive generates one byte of data for each expression (subfield) in the Operand. Subfields are separated by commas.

Each expression is evaluated and the resultant byte is formed from the modulo 256 remainder of the expression.

```
0212 01CC 01 TBL      FCB    1,2,$03,*/256
          02 03 01
0213
0214 01D0 41 STNG     FCC    /ABCD/        ;DELIMETER '/' REQUIRED
          42 43 44
0215
0216 01D4 41 STNG2    FCC    /AB//CD/      ;TWO DELIMETERS IN SUCCESSION
          42 2F 43 44
0217                                       ;CAUSE CHAR / TO BE INCLUDED
0218                                       ;ONCE.
0219                                       ; (FOUR FOR TWICE...)
0220
0221 01D9             RMB    9             ;SAVE SPACE
0222
0223 01E2 01 AENTR1   FDB    ENTRY1        ;DOUBLE WORDS
          CC
0224 01E4 00          FDB    1,2,22987,*+3 ;
          01 00 02 59 CB 01 E7
```

FCC

The Form-Constant-Character directive generates n bytes of 7-bit ASCII code corresponding to the character string in the Operand field. Any of the ASCII character set may be used.

FDB

The Form-Double-Byte directive generates two bytes of code for each expression in the Operand field. The format of the FDB statement is identical to that of the FCB. Each expression is evaluated as a signed 16-bit value in the range –32768 to 65535, with the most significant eight bits placed in the first byte.

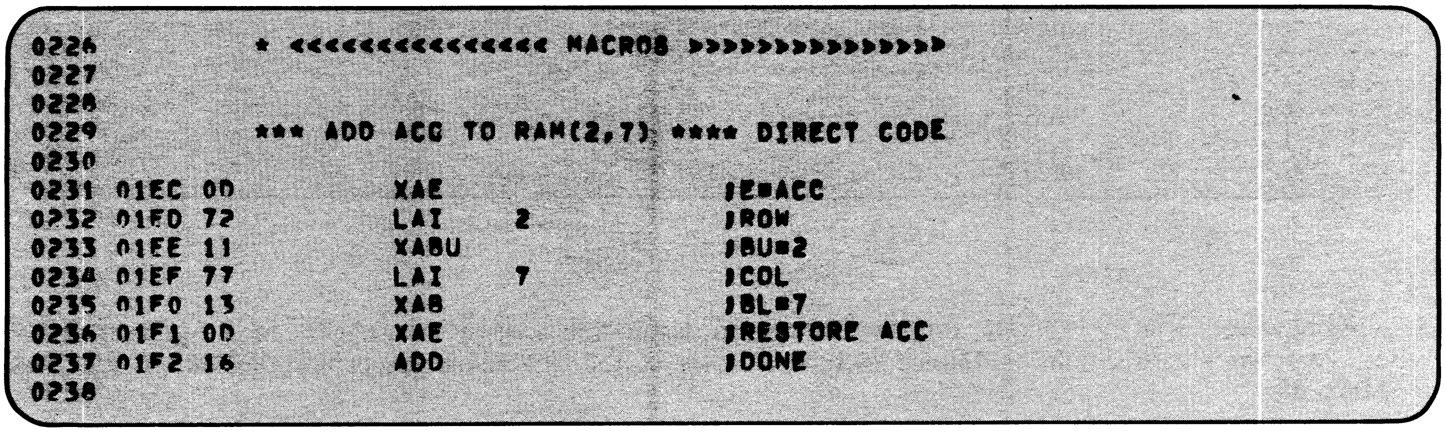
RMB

The Reserve-Memory-Block directive is used to reserve one or more bytes of memory without filling them with data. The expression in the Operand field must evaluate to a positive integer and defines the number of bytes to reserve.

MACROS

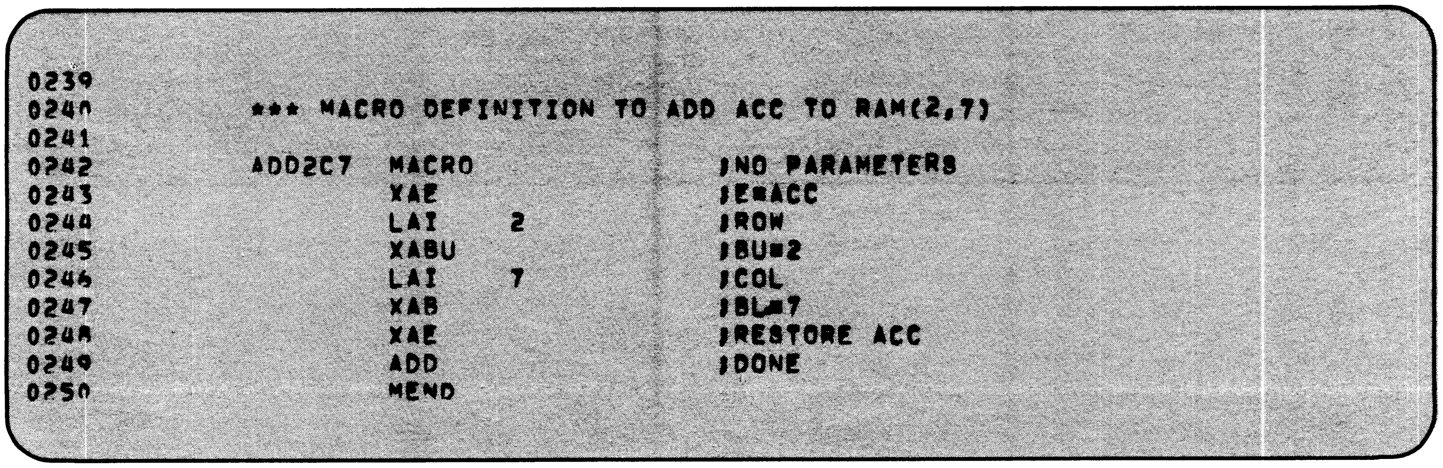
Often the programmer will find the same or similar sequence of source language instructions occurring in the program in several different places. With an ordinary assembler, the duplicate code must be rewritten for each occurrence. A Macro is a device which permits the programmer to give a symbolic name to a sequence of source code. Then in each place where that sequence would occur, its name is placed as if it were an instruction mnemonic, and the assembler substitutes instead the body of the macro, which is the sequence of instructions with that name. Parameters may be defined for the macro, which further specialize the sequence for each use of it.

For example, suppose the programmer has several occasions to perform the same operation. This may be coded in the source language as the following:



```
                * <<<<<<<<<<<<<<<< MACROS >>>>>>>>>>>>>>>>


                *** ADD ACC TO RAM(2,7) **** DIRECT CODE

      01EC 0D           XAE                     ;E=ACC
      01ED 72           LAI     2               ;ROW
      01EE 11           XABU                    ;BU=2
      01EF 77           LAI     7               ;COL
      01F0 13           XAB                     ;BL=7
      01F1 0D           XAE                     ;RESTORE ACC
      01F2 16           ADD                     ;DONE

```

If every occurrence of these instructions together involved the same two memory locations, a macro might be defined with the name ADD2C7:



```

                *** MACRO DEFINITION TO ADD ACC TO RAM(2,7)

                ADD2C7  MACRO                   ;NO PARAMETERS
                        XAE                     ;E=ACC
                        LAI     2               ;ROW
                        XABU                    ;BU=2
                        LAI     7               ;COL
                        XAB                     ;BL=7
                        XAE                     ;RESTORE ACC
                        ADD                     ;DONE
                        MEND
```

Then where the sequence is actually to occur, the single mnemonic ADD2C7 may be used.

```

                *** TWO SUCCESSIVE MACRO CALLS (OR INVOCATIONS) ***

     01F3                 ADD2C7
++01 01F3 0D              XAE                   ;E=ACC
++01 01F4 72              LAI     2             ;ROW
++01 01F5 11              XABU                  ;BU=2
++01 01F6 77              LAI     7             ;COL
++01 01F7 13              XAB                   ;BL=7
++01 01F8 0D              XAE                   ;RESTORE ACC
++01 01F9 16              ADD                   ;DONE

     01FA                 ADD2C7
++01 01FA 0D              XAE                   ;E=ACC
++01 01FB 72              LAI     2             ;ROW
++01 01FC 11              XABU                  ;BU=2
++01 01FD 77              LAI     7             ;COL
++01 01FE 13              XAB                   ;BL=7
++01 01FF 0D              XAE                   ;RESTORE ACC
++01 0200 16              ADD                   ;DONE
```

Suppose however, as is more likely, that the source and destination addresses are not the same for every occurrence. Then the macro should be defined with parameters, so that the actual source and destination addresses may be used:

```

                *** MORE GENERAL MACRO DEFINITION TO
                *** ADD ACC TO RANDOM RAM(ROW,COL).

                ADDRM     MACRO   @ROW,@COL
                          XAE                   ;E=ACC
                          LAI     @ROW          ;DESIRED ROW
                          XABU                  ;BU=ROW @ROW
                          LAI     @COL          ;DESIRED COLUMN
                          XAB                   ;BL=COL @COL
                          XAE                   ;RESTORE
                          ADD                   ;DONE
                          MEND
```

The mnemonic must be accompanied by the actual addresses (or their symbolic names):

```

               *** ADD RAM(2,7) WITH ADDRM MACRO
               *** MUST SPECIFY ROW,COL

     0201               ADDRM  2,7
++01 0201 0D            XAE                  ;E=ACC
++01 0202 72            LAI    2             ;DESIRED ROW
++01 0203 11            XABU                 ;BU=ROW 2
++01 0204 77            LAI    7             ;DESIRED COLUMN
++01 0205 13            XAB                  ;BL=COL 7
++01 0206 0D            XAE                  ;RESTORE
++01 0207 16            ADD                  ;DONE

               *** HOWEVER, NOW CAN USE SAME MACRO
               *** FOR OTHER RAM LOCATIONS.

     0208               ADDRM  1,9
++01 0208 0D            XAE                  ;E=ACC
++01 0209 71            LAI    1             ;DESIRED ROW
++01 020A 11            XABU                 ;BU=ROW 1
++01 020B 79            LAI    9             ;DESIRED COLUMN
++01 020C 13            XAB                  ;BL=COL 9
++01 020D 0D            XAE                  ;RESTORE
++01 020E 16            ADD                  ;DONE

     020F               ADDRM  3,1
++01 020F 0D            XAE                  ;E=ACC
++01 0210 73            LAI    3             ;DESIRED ROW
++01 0211 11            XABU                 ;BU=ROW 3
++01 0212 71            LAI    1             ;DESIRED COLUMN
++01 0213 13            XAB                  ;BL=COL 1
++01 0214 0D            XAE                  ;RESTORE
++01 0215 16            ADD                  ;DONE

```

Note that in each reference, the first parameter is substituted for the dummy parameter "@ROW" and the second for the dummy parameter "@COL"; the three macro expansions follow each call or invocation.

MACRO Directive

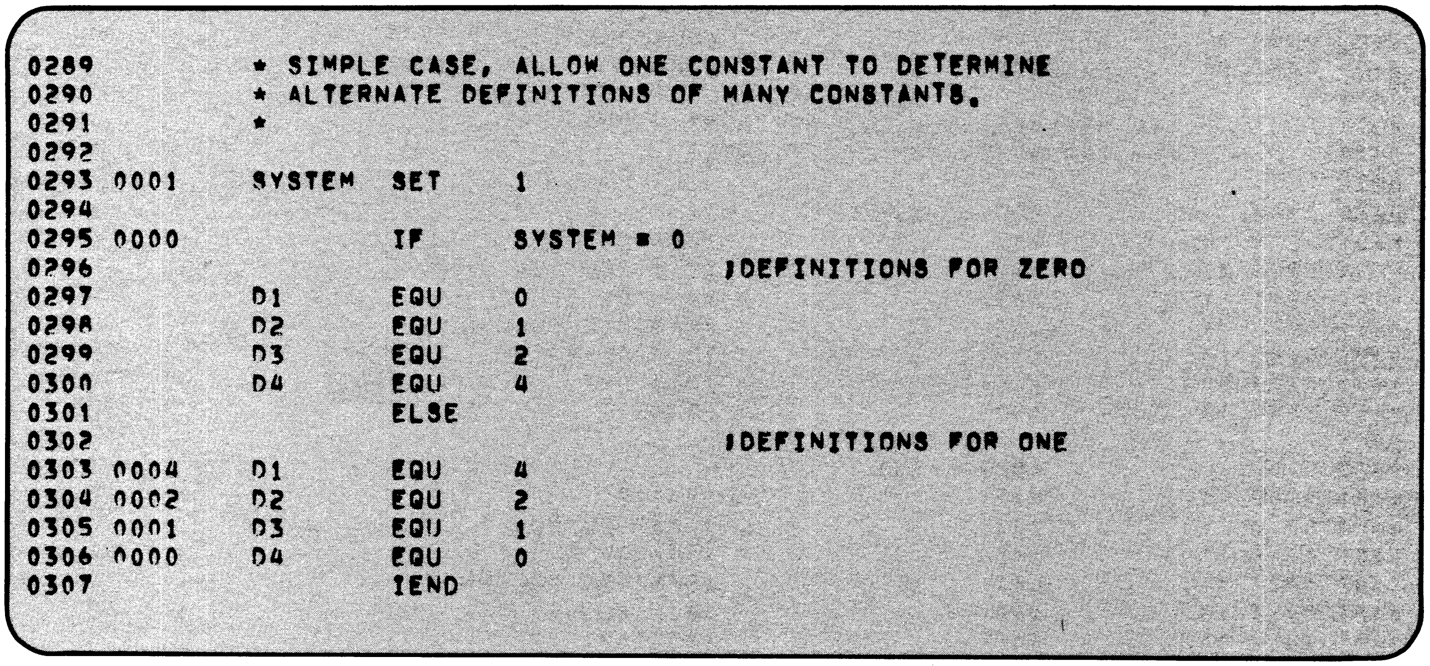
The MACRO directive signals the beginning of a macro definition. The symbol in the Label field of the MACRO statement is required, and becomes the name of the macro. The Operand field is optional. The Operand field may contain any number of symbols, which are defined within the macro as parameters; subsequent use within the macro definition of the parameter symbols become references to the macro parameters. The symbols in the Operand field of the MACRO statement are formal or "dummy" parameters, and will not conflict with any symbols defined elsewhere in the program; they are thus local to the macro definition. Each formal parameter must begin with the character '@.' When the macro is called or invoked, the actual arguments (which form sub-fields of the calling statement's Operand field) replace every occurrence of the corresponding formal parameters. The parameters are passed as strings, so there are no restrictions in the manner of their use in the macro definition. The macro call must specify the same number of parameters as the definition. Null parameters can be specified in the macro call by leading or successive commas, in which case the corresponding actual parameters are null, and their references in the macro expansion are deleted.

MEND Directive

The MEND directive designates the end of a macro definition. Every occurrence of a MACRO directive must be followed eventually by exactly one matching MEND directive. The MEND directive permits neither Label nor Operand. Review the examples in the previous section for MEND usage.

CONDITIONAL ASSEMBLY

One of the advantages the use of a microprocessor brings to a system design is the ease of restructuring it for different configurations by simply changing the program. When the larger part of a program is the same for the different versions of the system, and only sections of it need to be added or deleted to configure the various versions, it is convenient to write one program incorporating all of the potential sections, then tell the assembler to omit those parts which are not applicable to the particular configuration. This is done by the feature know as "conditional assembly" since parts of the program are assembled or not, based on some defined conditions. The same method may be used in writing a program which includes debugging and test statements during checkout, but excludes them in the final product.



```
         * SIMPLE CASE, ALLOW ONE CONSTANT TO DETERMINE
         * ALTERNATE DEFINITIONS OF MANY CONSTANTS.
         *

0001     SYSTEM  SET    1

0000             IF     SYSTEM = 0
                                       ;DEFINITIONS FOR ZERO
         D1      EQU    0
         D2      EQU    1
         D3      EQU    2
         D4      EQU    4
                 ELSE
                                       ;DEFINITIONS FOR ONE
0004     D1      EQU    4
0002     D2      EQU    2
0001     D3      EQU    1
0000     D4      EQU    0
                 IEND
```

Conditional assembly is controlled by the IF, ELSE, and IEND assembler directives. There are two basic forms:

```
        IF        condition
        •
        •
        •
        IEND
```

and

```
        IF        condition
        •
        •
        •
        ELSE
        •
        •
        •
        IEND
```

In the first case, the statements between the IF and the IEND will be assembled only if the condition is true (least significant bit = 1). When an ELSE is included, the statements between the IF and the ELSE will be assembled when the condition is true, and the statements between the ELSE and the IEND will be processed when the condition is false. IFs may be nested up to eight deep; that is to say, a complete IF – ELSE – IEND block may appear within the range of another IF.

CONDITIONAL ASSEMBLY (Continued)

```
0309            * MOST USEFUL WITHIN MACRO TO GENERATE
0310            * EFFICIENT CODE.
0311
0312            * MACRO ADDR IS USED TO ADDRESS A DESIRED RAM LOCATION.
0313            * RAM ADDRESS ARE CONSIDERED AS INTEGERS BU*16 + BL.
0314
0315 0010       BUK      EQU     16
0316
0317            * SET BU, BL TO POINT TO ADDRESS IN RAM
0318            *     ARGUMENT IS BU*16 + BL
0319            *     ACCUMULATOR MAY BE CLOBBERED; MAY BE SAVED USING XAE
0320            *
0321            ADDR     MACRO   #A
0322                     IF      ((#A)&15)=15
0323                     LBF     (#A)/BUK
0324                     ELSE
0325                     LBZ     (#A)/BUK
0326                     IF      ((#A)&15) <>0
0327                     LAI     #A            ;;A CLOBBERED
0328                     XAB
0329                     IEND
0330                     IEND
0331                     MEND
```

```
0332
0333 0013     FLAG1    EQU     19            ;MEMORY LOC
STMT LOC OBJ SOURCE
0337 0216              ADDR    FLAG1         ;REQUIRES THREE INSTRUCTIONS
++01 0000              IF      ((FLAG1 )&15)=15
++01                   LBF     (FLAG1 )/BUK
++01                   ELSE
++01 0216 41           LBZ     (FLAG1 )/BUK
++01 FFFF              IF      ((FLAG1 )&15) <>0
++01 0217 73           LAI     FLAG1         ;;A CLOBBERED
++01 0218 13           XAB
++01                   IEND
```

ERROR

The Error directive will force an assembly error message. The errror text is taken from the Operand field of the directive.

Example of Use: Assume your program may not exceed 1K (400 hex) in size. The sequence below will produce an error message if the limit is exceeded.

```
IF *>= $400
ERROR PROGRAM TOO LARGE
IEND
END
```

Error messages are listed under the offending statement. The flag

> > > > ERROR < < < < [message]

appears immediately below the statement; the unique self-explanatory message that caused the error follows the flag.

LISTABLE OUTPUT

The assembly listing is formatted in pages of 52 lines per page plus headings. The first line contains the Operand field appearing on the TITLE statement and a page number. The second line contains the Operand field of the most recent PAGE statement. The third line is blank and the fourth line contains the heading categories for the listing. The category names are: statement number (STMT), location counter (LOC), object code (OBJ), and source statement (SOURCE).

The set of statements to be included in the assembled output listing can be controlled by the following directives:

LIST	– the primary switch for generation of the listing information; its initial value is set by the FDOS option selected when calling the assembler:

```
!AP, inputfile, outputfile, option
```

The following table shows the destination of both the output listing and the object code, in response to the chosen "option."

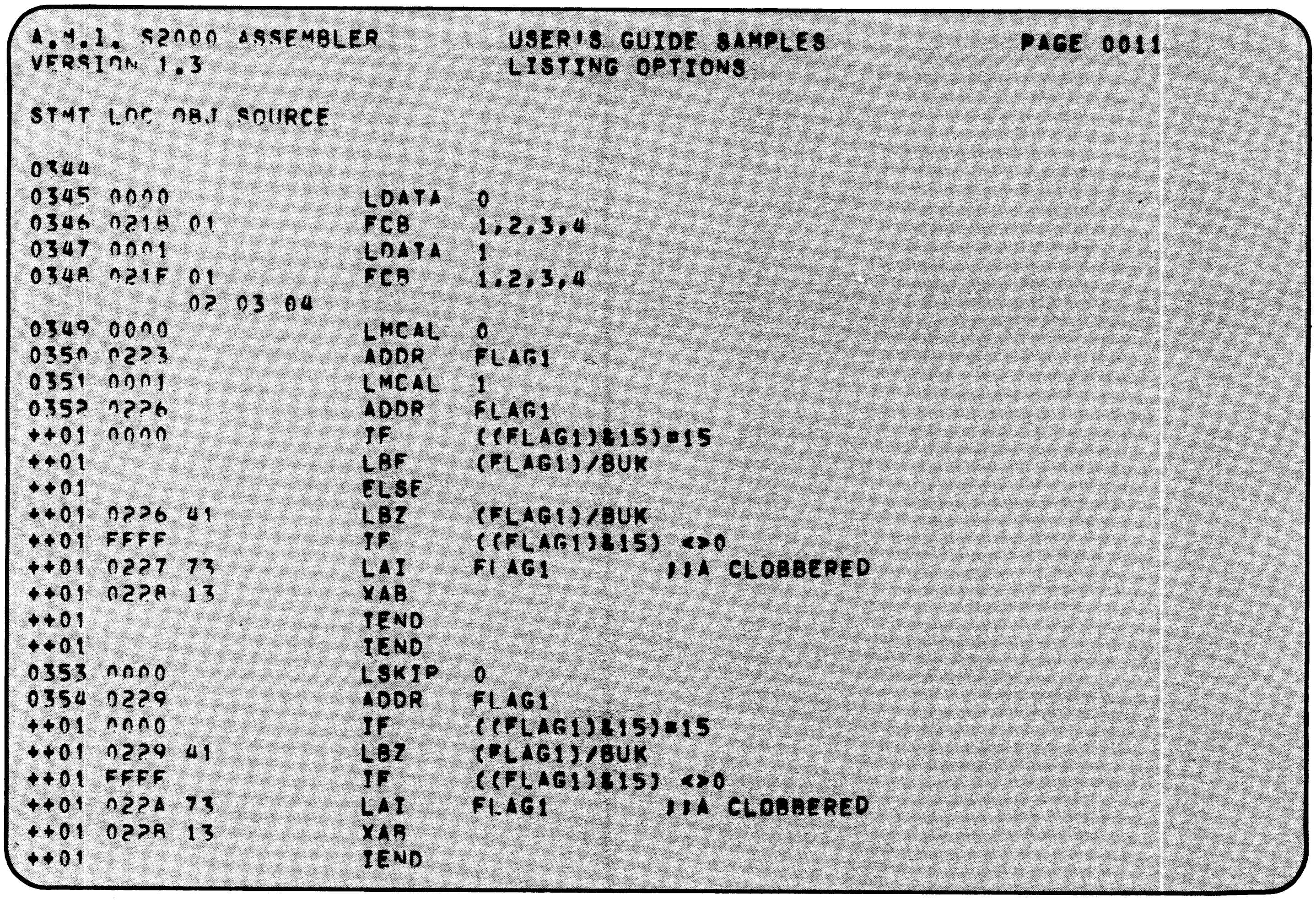
If "Option" Is:	Destination of Listable Output	Destination of Object Code	Remarks
0	disk outputfile	–	0 is the default value of "option."
1	disk outputfile	disk outputfile	List and object interleaved.
2	MDC screen	disk outputfile	
3	MDC screen	–	
4	–	disk outputfile	

LDATA	– list object for statements generating more than one byte of data; initial value "off."
LMCAL	– list statements generated from macro calls (invocation of previously defined macros); initial value "on."
LMDEF	– list macro definitions; initial value "on."
LSKIP	– list statements skipped by conditional assembly directives; initial value "on."
LSYMB	– list symbol table after assembly is complete; (value only checked once); initial value "off."

The value of a directive's operand expression determines whether the option is on or off, i.e., whether the statements controlled by the directive are included in the listing. A "true" value (least significant bit = 1) requests "on" or "inclusion"; a "false" value (0) requests exclusion.

The previous values are maintained on an eight level stack. If no operand is present for the directive, then the previous value is pulled from the stack and this value is used to control the listing. This feature is very useful when using the options in macros.

Note that when LIST is off, the other directives have no effect.



```
A.M.I. S2000 ASSEMBLER          USER'S GUIDE SAMPLES          PAGE 0011
VERSION 1.3                     LISTING OPTIONS

STMT LOC OBJ SOURCE

0344
0345 0000            LDATA  0
0346 021B 01         FCB    1,2,3,4
0347 0001            LDATA  1
0348 021F 01         FCB    1,2,3,4
          02 03 04
0349 0000            LMCAL  0
0350 0223            ADDR   FLAG1
0351 0001            LMCAL  1
0352 0226            ADDR   FLAG1
++01 0000            IF     ((FLAG1)&15)=15
++01                 LBF    (FLAG1)/BUK
++01                 ELSE
++01 0226 41         LBZ    (FLAG1)/BUK
++01 FFFF            IF     ((FLAG1)&15) <>0
++01 0227 73         LAI    FLAG1          ;;A CLOBBERED
++01 0228 13         XAB
++01                 IEND
++01                 IEND
0353 0000            LSKIP  0
0354 0229            ADDR   FLAG1
++01 0000            IF     ((FLAG1)&15)=15
++01 0229 41         LBZ    (FLAG1)/BUK
++01 FFFF            IF     ((FLAG1)&15) <>0
++01 022A 73         LAI    FLAG1          ;;A CLOBBERED
++01 022B 13         XAB
++01                 IEND
```

34	TITLE
34	PAGE
34	END
35	ORG
35	EQU
34	SET
29	LOCAL
29	LBEG
29	LEND
37	MACRO
37	MEND
42	LIST
42	LDATA
42	LMCAL
42	LMDEF
42	LSKIP
42	LSYMB
40	IF
40	ELSE
40	IEND
36	FCB
36	FDB
36	FCC
36	RMB
41	ERROR

TABLE 5-1 DIRECTIVE SUMMARY

6 LOADER

LOADER

The AMI S2000 Loader (LD) is a MDC 6800 program that is used to load object files created by the AMI S2000 Assembler into MDC memory. This allows operation of the DEV-2000 Board with the MDC S2000 Debug (DB) program for verifying the correct functioning of the S2000 with a user's program; it is also necessary to use LD prior to using the software simulator (SM) and the EPROM programmer (P6834).

FUNCTION

LD offsets the user's defined memory allocation by $(2000)_{16}$ before loading. In addition, it optionally places a WAI instruction in all undefined memory locations. The DEV-2000 Board, when issuing S2000 memory requests, also offsets the request by $(2000)_{16}$. This allows the MDC programs (e.g., LD, DB) to execute in the lower 8K of MDC memory. It also facilitates loading separate S2000 object programs before beginning the debug process. The upper 8K (2000-3FFF) is shared memory; while the debug program is running, it is available to the MDC; while the S2000 program is executing, it is available to the DEV-2000 Board.

EXAMPLE:

To load an S2000 program assembled by AP (for example, DEMO), use

```
!LD, DEMO
```

7 DEBUGGER AND DEV-2000 BOARD

INTRODUCTION

The AMI S2000 Debug Program (DB) is a powerful tool designed to allow the users to interact with the combined MDC/DEV-2000 Board for controlled testing of their S2000-based hardware. The commands available to the user provide far more extensive capabilities than are normally found on a computer control panel.

To call DB from FDOS, type

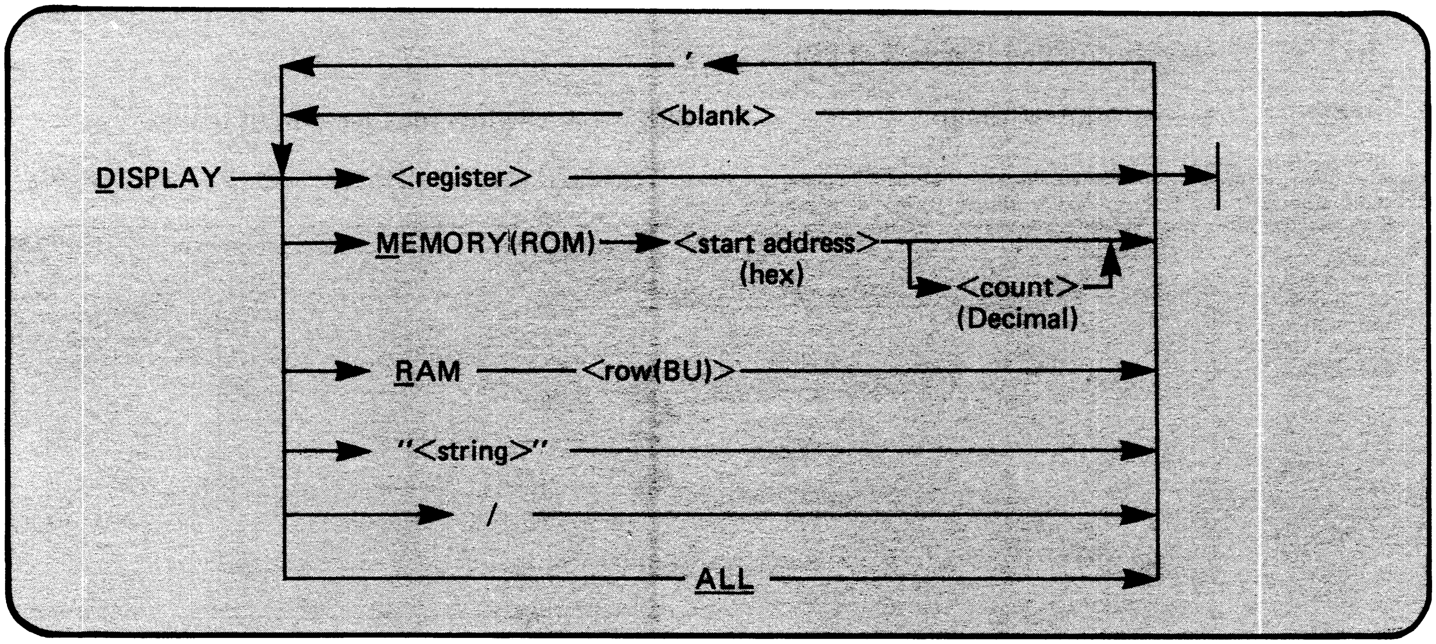
```
!DB,ALL,<outputfile>
```

where ALL is a file containing Debugger commands in the form of macros. While a value of 1 is in the pseudo-register DSK, anything that appears on the CRT will be output to the disk file named. To exit the Debugger, close the output file, and return to FDOS, use the shifted DPLX key.

BASIC COMMANDS

DISPLAY
SET
REMOVE
EXECUTE

The minimum abbreviation is underlined on each syntax diagram.



Displays the contents of S2000 registers or memory.

"ALL" displays all S2000 registers. Note: ALL is an S2000 library macro which must be loaded explicitly, i.e., DB, ALL.

Quoted strings are displayed verbatim. They may be used to format and label the display output.

Slashes cause a carriage return — line feed to be output.

EXAMPLES:

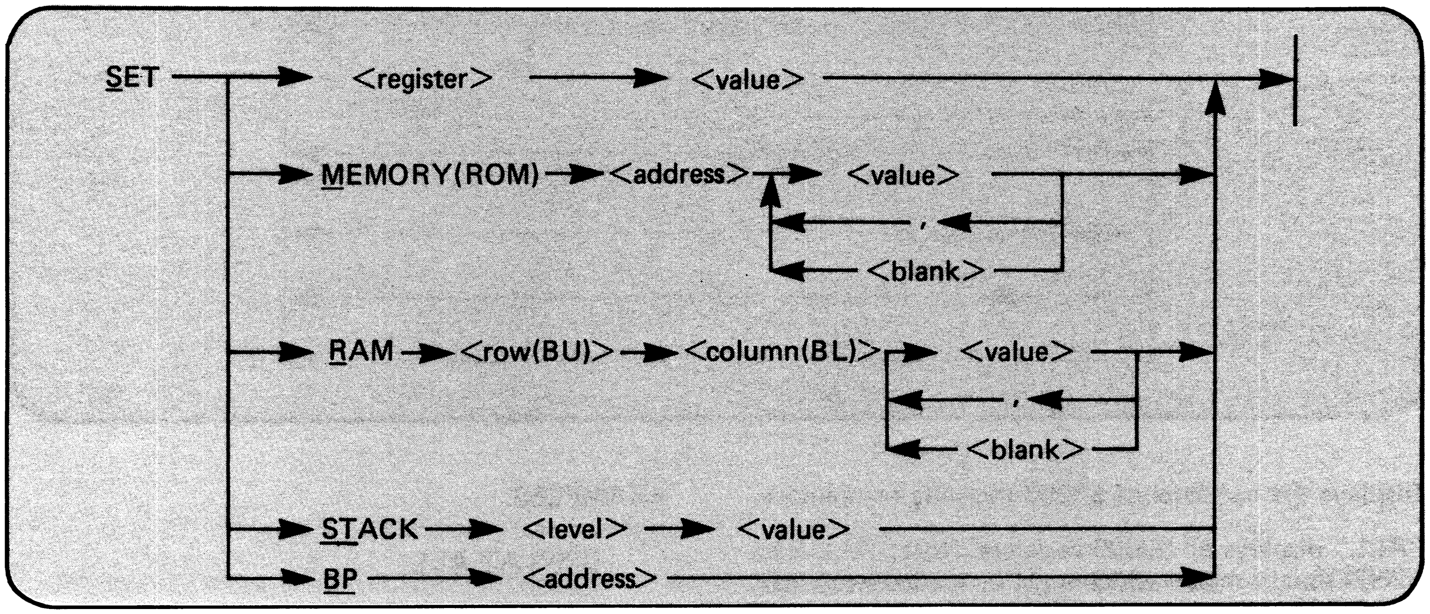
```
DISPLAY ALL

DISPLAY A, E, BL, BU

DISPLAY "BL =" BL

D M 400 50
```

REGISTERS:

A	
BL	
BU	
CARRY	
DISPLAY	= DIN if in float; = SL if not floating
E	
I	
K	
KIS	Input Source for K & I lines (1 = external, 0 = MDC control)
LI	Last Executed Instruction
NI	Next Instruction (Memory at PC)
MAL	Master Address Latch

SAL	Slave Address Latch
PC	Program Counter = LR + PR + BR
STACK	(level Ø is closest to the PC)
BP	Break Points
MEMORY	ROM
RAM	RAM
DF	Float on D bus*
DP	Polarity of display outputs (1=normal)
DINPUT	D external input source
SL	Segment latches
F1	
F2	
DSK	Switch for CRT output to disk

*When an "INP" instruction is executed, the source of the data being put into the Accumulator and RAM is a function of DF; if the D Bus is floating, data comes from DIN (i.e., from an outside source); if DF = Ø, data comes from SL.



The SET statement assigns new values to S2000 registers or memory.

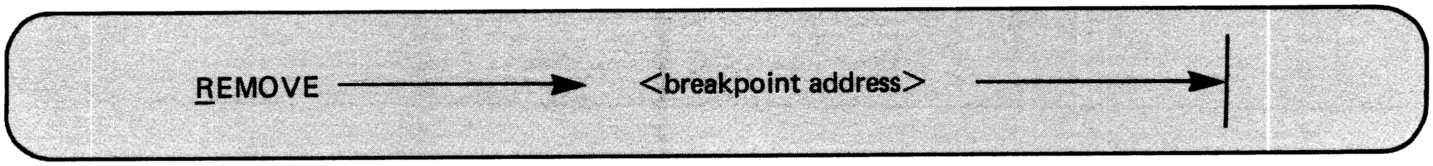
MEMORY refers to program storage, which is actually RAM located in the MDC system.

RAM refers to S2000 internal RAM, which is organized into 4 rows of 16 digits each. <row> and <column> provide the starting digit location. Row ranges from

0 to 3, column from 0 to F (hex). Providing more than one value after row/column will set multiple digits along a row.

EXAMPLES:

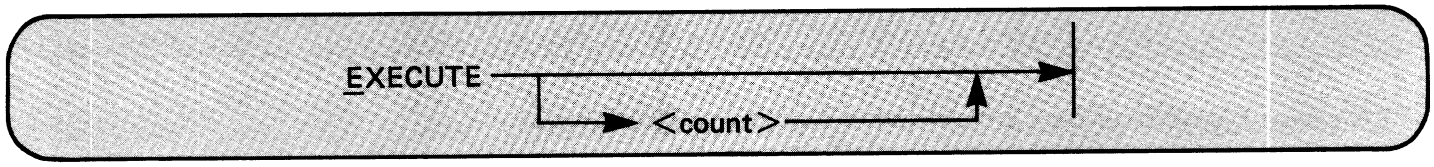
```
SET     BL 6

SET     RAM 3 0 6, 6, 7, 9
```



Breakpoints are set with the SET command, displayed by the DISPLAY command, and removed by the REMOVE command. A maximum of 16 breakpoints may be set concurrently.

EXAMPLES:

```
REMOVE     13F0

R DF1
```

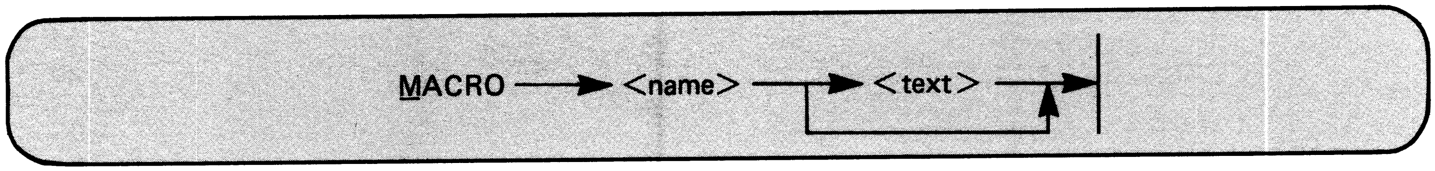


EXECUTE causes the S2000 to run. If no count, or a count of 0 is provided, it will run until either a breakpoint is encountered or a key is depressed.

A count greater than zero will cause that many S2000 instructions to be executed.

EXAMPLES:

```
EXECUTE

E  3
```



The MACRO command is used to attach a name to a frequently used debugger command sequence. Once a macro has been defined, the name may be used any place the original text is desired. This can save lots of typing.

A MACRO definition with no text will cause any previous definition to be deleted.

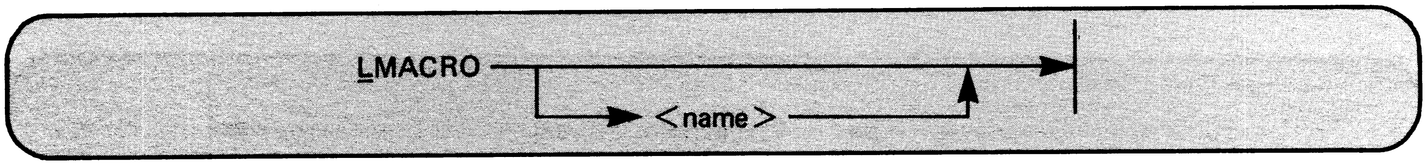
The text includes everything between the name and the end of the line.

The name must begin with a letter and consist entirely of letters and numbers. There is no restriction on length.

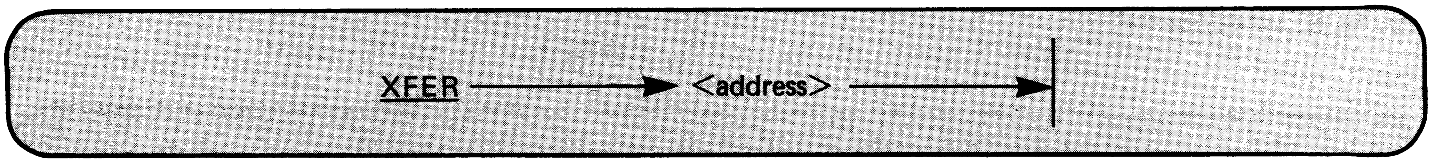
EXAMPLES:

```
MACRO  DABE   DISP   "A="A"B="B"E="E

MACRO  I01  E;D  SL;S  K  O;E;  D  SL

M KILL
```

A special macro named PROFILE is called automatically after each execution of the EXECUTE command. Thus a special display can be implemented each time a breakpoint is encountered, or each time the EXECUTE counter is decremented. To disable the PROFILE macro, simply redefine it with no argument.



The LMACRO command lists all macro names and definitions if no name is specified, or the definition of the named macro if a name is supplied.



XFER causes the 6800 to do a JSR to the indicated address. This allows convenient access to special purpose user written debug functions.

EXAMPLES:

```
XFER    EC00
```

NOTE: This is a good one to try.
$EC00 is the MDC boot address.

8 INSTRUCTION SIMULATOR

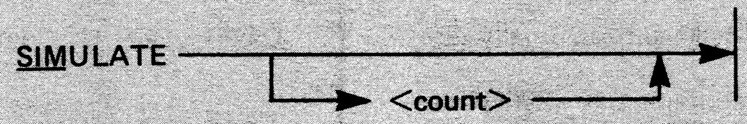
SIMULATION

The S2000 simulator, SM, is identical in operation with the DB program except the S2000 instructions are simulated entirely by software without any external attachments. Thus, the I/O ports are not available to connect to real devices. The simulator is useful for initial debug of algorithms or for educational purposes in learning how each instruction operates on the S2000 machine state. FINAL PROTOTYPE DEBUG SHOULD BE DONE WITH THE SES-2000.

The user program is limited to 4K. Any PC reference ⩾$1,000 is a simulated "HALT" and immediate return to the debugger (which is included as part of the SM program).

All registers described for DB are available to the simulator for setting and display. The Execute command is disabled. The SIMulate command is enabled and the following additional simulation only-registers are available:

CYC	Cycle counts
SFF	Seconds FF, set to 1 every 250,000 cycles, reset by SOS
SEC	Running seconds counter, increments every 250,000 cycles
MPX	State of S2000 simulated mode, 0 = static



No count or count = 0 causes simulation to begin at the current PC value and continue until:

a) HALT ($01) or DB breakpoint occurs

b) Keyboard input (e.g., DISPLAY)

c) (PC) ⩾$1,000

A positive count causes simulation of "count" instructions unless a condition (a-b-c above) occurs before the count is finished.

A negative count causes the display of the previous "count" PC values (⩽50) LIFO.

Examples:

```
SIMULATE
SIM 7
SIM -7
SIM 250
```

Programs loaded for simulation by LD should have memory initialized (by LD) to 01 (HALT).

9 PROM/ROM GENERATION

6834 PROMS

AMI 6834 PROMs may be programmed on the MDC from RAM memory. Note that the LD program loads S2000 programs offset by 8K (2000 HEX) which is out of the way of the PROM programmer software, P6834.

Issue the FDOS-II command

```
LD,filename
```

to load the S2000 object program into memory.

Issue the FDOS-II command

```
P6834
```

to run the 6834 PROM programmer. Commands to P6834 are entered via the keyboard after the prompt character "%" is displayed. The programming command is

Pxxxx,yyyy,zzzz where

xxxx	is the hexadecimal low memory address
yyyy	is the hexadecimal high memory address
zzzz	is the starting PROM address, usually zero

Because the 6834 PROM contains 512 (200 HEX) bytes and LD offsets the S2000 program by 8K (2000 HEX) the command to program the lower 512 bytes of a S2000 program would be

```
P2000,21FF,0
```

and the next locations would require

```
P2200,23FF,0
```

After typing the P command, the prompt

```
TURN PWR ON, STRIKE ANY KEY
```

is displayed. This is a request to the user to place his blank PROM in the programming socket and turn the power switch on (and then strike any key to signal the power is on).

Following programming, a comparison is made between memory and the PROM to ensure correct programming. If a discrepancy exists, the following error message is displayed

xxxx yy zz where

xxxx	is the memory address
yy	is the data in the PROM
zz	is the data in memory

After the comparison is complete, the message

```
TURN PWR OFF, STRIKE ANY KEY
```

is displayed. This requires the user to immediately place the power switch in the "off" position and signal the program through the keyboard. Additional PROMs may then be programmed.

Other P6834 commands are

Cxxxx,yyyy	Compare yyyy (hex) bytes in the PROM with memory at xxxx (hex)
Txxxx,yyyy	Transfer yyyy (hex) bytes in the PROM to memory beginning at xxxx (hex)

ROM GENERATION

The S2000 includes masked read only memory. Mask generation is performed on the B6700 computer system. The binary bit pattern from a S2000 object program can be transferred from a diskette file on the MDC to a B6700 CANDE file.

10 PROGRAMMING NOTE AND EXAMPLES

INVERTED INSTRUCTION ARGUMENTS:

Several instructions in the S2000 instruction set require that the argument of the instruction be inverted. Specifically, these are the XCI, XCD, XC, LAM, and PP instructions. The assembler automatically performs the required inversion. The instruction descriptions are written from the point of view of an assembly-language programmer; therefore the arguments are presented in their uncomplemented form. Thus for example, PP Y in the descriptions assembles as HEX 60 + ↑Y.

Program Start

When power is turned on, an automatic power-on reset ($\overline{\text{POR}}$) occurs. $\overline{\text{POR}}$ sets the Program Counter to Bank 0, Page 0, and Location 0 (NOTE: the Prepare Registers must be cleared by the programmer); it sets the 50/60Hz Counter to 60Hz, the Display Latch outputs to "non-inverted," and the mode to STATIC with a FLOATING D-bus. The first instruction to follow $\overline{\text{POR}}$ may not be an LAI, LBE, LBEP, LBF, or LBZ. A dual PP should be used to initiate the Prepare Registers.

Program Control

Unless a JMP or JMS instruction is **immediately** preceded by a PP, the Page Register will retain its contents and not be loaded from the Prepare Page Register.

NOTE: The last location on a page cannot be a JMP or JMS, or else the jump destination will be in the very next page, rather than the intended page, and the AP assembler generates an error message.

BU Modification Instruction

Several instructions modify the BU register according to the equation: new BU = old BU$\oplus$Y where Y is the instruction's argument.

old BU

new BU	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

Y

NOTE: If Y = 0
BU does not
change

The generation of a new BU value by exclusive OR-ing of the old value and an instruction variable allows bouncing back and forth between 2 RAM pages without concern for what the RAM page was at the beginning.

LAI Instruction

LAI instruction is used to load a number (0-15) into the Accumulator. If there is a group of LAI instructions immediately following one another, only the first one is executed. This is useful for coding subroutines with many entry points using multiple accumulator preloading. For example,

```
K0      LAI     0
K3      LAI     3
K8      LAI     8
        RT
```

is a subroutine which will load the Accumulator with the value 0, 3, or 8 depending upon whether it is entered at address K0, K3, or K8. The same is true when using the LAI instruction to select one of the K or I inputs. Note that the variable could select more than one input.

Subroutines and Page 15

If possible, arrange for page 15 to contain jumps to each of the major subroutines. Arrange for all subroutines and other program modules to start on page boundaries if there is enough room to do this – this will both decrease the likelihood that PP instructions will be needed, and decrease the likelihood that inserting or deleting an instruction (typically near the beginning of the program) will cause instructions further down to move across page boundaries, causing PP instructions to appear or disappear (causing further changes).

A convenient macro to assist this process is the following:

```
NEWPAG MACRO                    ;START A NEW ROM PAGE
       ORG (*/64 + 1) *64
       MEND
```

Skip Instructions and Program Size

Since skip instructions are assymetric (you can skip on only one of the two possible conditions), a little thing like the sense of a flag bit (i.e., 1 or 0 for condition true) can make a large difference in the size and efficiency of the resulting program. Internal flags can of course be redefined; this is more difficult with external hardware. It is essential that the programmer be aware of any hardware restrictions, and establish such things as the sense of the various switches and indicator lights as early as possible. If the program is begun soon enough into the hardware development, the programmer can make suggestions as to the most convenient hardware configuration from his/her point of view. It is most important to avoid unpleasant surprises late in the project ("Didn't you know you get a 0 input if the touch pad is touched?").

Skip Instructions and Documentation

For documentation, it is useful to define mnemonic macros for such things as I-Input tests, and symbolic names for A lines and the like. Thus, if A6 is the output which turns on an alarm, write

```
        ALARM EQU 6
```

And if I4 is 0 if the door is open, write

```
        SKDOOR MACRO
               LAI 4
               SZI      ;;SKIP IF DOOR OPEN
               MEND
```

It is important to note that the most important part of the above macro is the comment, which appears automatically every time the macro is invoked!

Control Transfer Macros:

The following macros are useful for calling subroutines not in page 15, and transferring control to locations out of page. Note that with the automatic local label convention in macros, local labels cannot be passed to these macros (i.e., they only work with global labels).

```
    CALL   MACRO     @ADDR      ;CALL SUBROUTINE AT ADDR
           PP        @ADDR/64
           JMS       @ ADDR
           MEND

    GOTO   MACRO     @ADDR      ;TRANSFER CONTROL TO ADDR
           PP        @ADDR/64
           JMP       @ADDR
           MEND

  EXAMPLE:
           CALL      SUB1
           GOTO      LOOP
```

TouchControl Capacitive Touch Keyboard Sensing:

The touch keyboard must be sensed in the following peculiar manner:

```
                ;SET BL TO ADDRESS APPROPRIATE A LINE
   PSL          ;NOT NECESSARY IF A LINE ALREADY LOW
   MVS          ;NOT NECESSARY IF A LINE ALREADY LOW
   PSH          ;NOT NECESSARY IF A LINE PRESET HIGH (BUT OUTPUT LOW)

   LAI     X    ;SELECT EXACTLY ONE K INPUT, DISCHARGE ALL OTHER K INPUTS TO VSS
   MVS          ;STROBE A LINE HIGH – ONE POSITIVE-GOING STROBE FOR EACH KEY PAD
 * (SOME NUMBER OF NOPs MAY BE REQUIRED IF THERE ARE PROPAGATION DELAYS IN THE
 * EXTERNAL CIRCUIT.)
   SKZ          ;SKIP IF SELECTED K INPUT IS 0 (KEY TOUCHED!)
```

A convenient trick is to preceed a sequence of tests on the same A line by setting a byte in memory to 15 (all 1's), and make the skipped instruction an RSM. This results in a byte in memory with a "1" bit for each key touched. Thus, with 4 keys on A0,

```
   SF1                          ;FLAG 1 WILL REMAIN SET IF THE KEYBOARD IS
                                ; THE SAME AS IT WAS LAST TIME IT WAS SCANNED,
                                ; THUS ALLOWING FOR NOISE AND "BOUNCE".
   LBZ           0              ;SELECT MEMORY BYTE AT BU = BL = 0
   LAI           15             ;ALL ONES
   XC            0              ;ACC ↔ MEM
   XAE                          ;SAVE OLD MEM VALUE IN E FOR DEBOUNCE TEST

TTOUCH MACRO     @NUM, @BIT     ;MACRO TO TEST BIT IN TOUCH KEYBOARD
   PSL                          ;STROBE LOW
   MVS
   PSH                          ;READY TO STROBE HIGH
   LAI           @NUM           ;SELECT K1
   MVS                          ;STROBE HIGH
   SZK
   RSM           @BIT
   MEND

TTOUCH 1,0                      ;TEST TOUCH PAD 1
TTOUCH 2,1                      ;TEST TOUCH PAD 2
TTOUCH 4,2                      ;TEST TOUCH PAD 4
TTOUCH 8,3                      ;TEST TOUCH PAD 8

   XAE                          ;GET OLD KEYS BACK FOR DEBOUNCE TEST
   SAM                          ;SKIP IF SAME AS LAST TIME
   RF1                          ;    ELSE RESET FLAG 1
```

RAM Addressing:

If a program has any number of flags and variables scattered throughout memory, it is almost essential to have a set of macros designed to simplify the process of addressing them. In the following macros, addresses are represented as BU * 16 + BL. A constant BUK has been defined for clarity: The BU part of an address ADR can then be expressed as ADR/BUK.

```
BU      EQU          16

*
*    SET BU, BL TO POINT TO ADDRESS IN RAM
*       ARGUMENT IS BU*16 + BL
*       ACCUMULATOR MAY BE CLOBBERED; MAY BE SAVED USING XAE
*
ADDR    MACRO    @A
        IF       ((@A)&15) = 15
        LBF      (@A)/BUK
        ELSE
        LBZ      (@A)/BUK
        IF       ((@A)&15) <> 0
        LAI      @A              ;;A CLOBBERED!
        XAB
        IEND
        IEND
        MEND
```

Note that the most commonly used variables should have BU equal to either 0 or 15, so that they can be addressed in a single byte (using LBZ or LBF) without changing the accumulator.

```
*
* LOAD A FROM MEM AT ADR
*
LOADA   MACRO    @ADR
        ADDR     @ADR
        LAM      0
        MEND

*
* STORE A TO MEM AT ADR
*     OLD VALUE OF MEM TO A
*
STORA   MACRO    @ADR
        IF       ((@ADR)&15) = 15
        LBF      (@ADR)/BUK
        ELSE
        IF       ((@ADR)&15) = 0
        LBZ      (@ADR)/BUK
        ELSE
        XAE
        ADDR     @ADR
        XAE
        IEND
        IEND
        XC       0
        MEND
```

```
*
* SET MEM AT ADR TO VAL
*     OLD VALUE OF MEM TO A
*
SETMEM MACRO    @ADR, @VAL
       ADDR     @ADR
       LAI      @VAL
       XC       0
       MEND

*
* SET BL REGISTER
*     A CLOBBERED
*
SETBL  MACRO    @BL
       LAI      @BL
       XAB
       MEND

*
* SET BU REGISTER
*     A CLOBBERED
*
SETBU  MACRO    @BU
       LAI      @BU
       XABU
       MEND

*
* INCREMENT BL
*    LEAVES ACC AND MEM INTACT
*
INCBL  MACRO
       XC       0
       XCI      0
       MEND

*
* DECREMENT BL
*    LEAVES ACC AND MEM INTACT
*
DECBL  MACRO
       XC       0
       XCD      0
       MEND
```

2KF

2KF is a text file containing a suggested standardized format for S2000 source programs. It is a basic skeleton to which you can add your own code to create a complete S2000 source program. For a copy, contact AMI.

Appendix A MDC FDOS II Command Summary

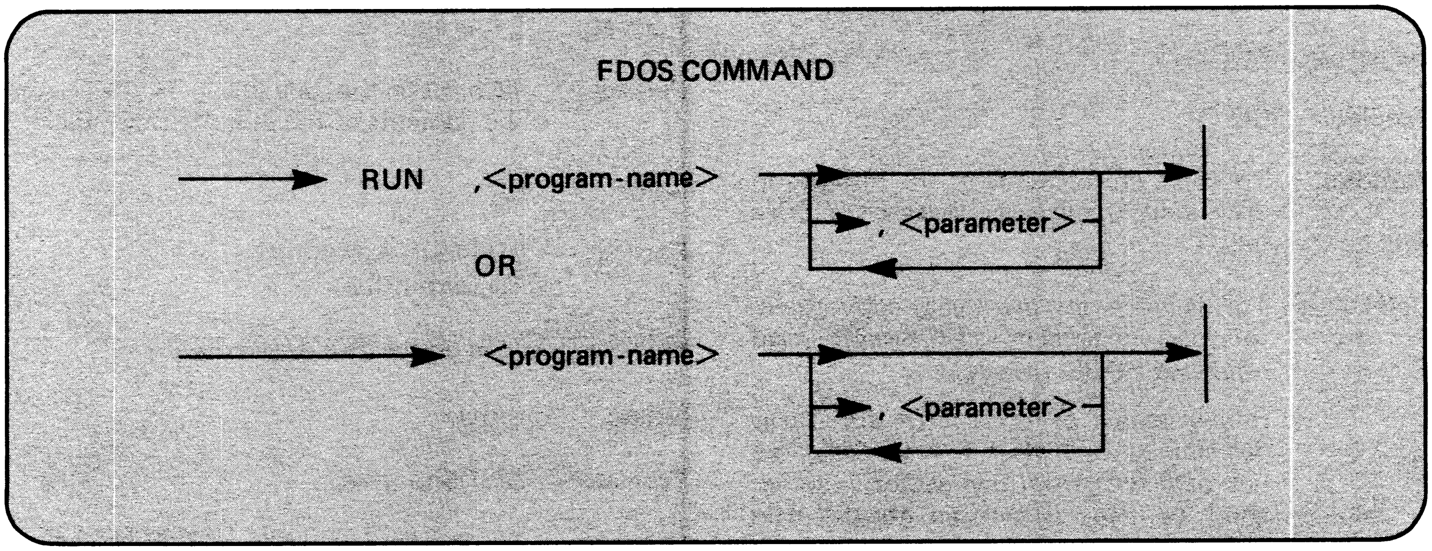
MDC Switches:

power on	underneath keyboard on left; toggle on front of diskette drive
<RESET>	top row, right most key, total system reset
<BOOT>	top row, left most key, bootstrap FDOS-II from diskette in drive 0
<COM>	top row, middle, MDC CRT/ keyboard simulated teletype

general: 20-key pad on right is for edit (program ED) operations; main keyboard is standard seven-bit ASCII.

Diskettes: Floppy but fragile; handle with care. Power should be on to the drive when inserting or removing a diskette. The only time the diskettes should be removed from their protective paper jackets is when they are in the disk drive.

Standard Operating Procedure: Power on system; insert diskette with S2000 software; wait approximately 20 seconds for diskette to become ready; <BOOT>; communicate with FDOS-II via keyboard.



NOTE: No blank characters anywhere in the command line. Parameters are order dependent for all programs.

EXAMPLES:

AP,<inputfilename>,<outputfilename>,
<option>
ED,<oldfile>,<newfile>

FDOS prompts user by displaying an exclamation mark (!). User commands are followed by carriage return <RETURN>. <BACKSPACE> deletes the most recently typed character.

Name: AP

Format: AP,sourcefilename,outputfilename,option

Purpose: To assemble the contents of the source file and generate an object or listing file.

Comments: option = 0 is default. Both filenames must be specified.

option = 0 listing to output file, no object
= 1 listing and object merged to output file
= 2 listing to CRT, object to output file
= 3 listing to CRT, no object
= 4 no listing, object to output file

EXAMPLE: to produce listing file

AP,TECSI,TECSL

to produce object file

AP,TECSI,TECSX,4

Name: COPY

Format: COPY

Purpose: To copy the contents of the diskette in drive unit 0 onto the diskette in drive unit 1.

Comments: This is a one-for-one image copy; therefore, the contents of either diskette need not be of FDOS-II format.

If any sector of the source diskette is determined bad after 5 read tries, the last data read from that sector, whether good or bad, is written to the new diskette.

Name: DELET

Format: DELET:unitnumber,filename1,filename2, ...filenamen

Purpose: To delete the designated, non-permanent, files from the diskette, in the specified drive unit, and to repack the contents of that diskette's user file area and file directory area, thus making the disk space available for additional files.

Comments: The file names need not be in any specific order.

The unit number refers to the drive unit in which the diskette, with the specified files to be deleted, is loaded. The unit number be 0, 1, 2, or 3. If the unit number is omitted, 0 is assumed.

Examples: DELET:2,JOE1,JOE7,AL,SAM,JACK

DELET,JOE1,JOE7,AL,SAM,JACK

Deletes the specified files from the diskette loaded into drive unit 0.

Name: DB

Format: DB,inputfilename

Purpose: To debug, using the MDC and S2000 emulation board, an S2000 object program previously loaded into memory.

Name: DPA

Format: DPA,inputfile

Purpose: To print on the Data Products line printer the contents of the diskette input file.

Name: ED

Format: ED,inputfile,outputfile
ED,,outputfile

Purpose: Text editor. See Appendix B.

Name: GPIO

Format: GPIO,inputfile

Purpose: To transfer input file (S2000 object) to B6700 for masked ROM generation. See Section 9.

Comments: See MDC software reference for other uses.

Name: LD

Format: LD,inputfile

Purpose: To load S2000 object program into MDC memory.

Name: LDIR

Format: LDIR:unitnumber,listdevice

Purpose: To print out the contents of the file directory on the diskette in the specified drive unit. Lists the file names, attributes, file's starting track and sector, and the file's size in sectors. Lists to the specified list device.

Comments: The unit number may be 0, 1, 2, or 3. If the unit number is omitted, 0 is assumed.

The listdevice operand may be C or L. If the listdevice operand is omitted, C is assumed. C specifies the console and L specifies the line printer.

Example:
```
LDIR:1,L
```

Lists the file directory of the diskette in drive unit 1 onto the line printer.

```
LDIR
LDIR:
LDIR:0
LDIR:0,C
LDIR:C
```

Lists the file directory of the diskette in drive unit 0 onto the console.

Name: MERGE

Format: MERGE,newfilename,filename1, filename2, ...filenamen

Purpose: To create a new file whose contents are the concatenation of the contents of the specified files, in the order in which they appear in the command.

Comments: The existing files are unaffected.

Examples:
```
MERGE,MAIN,SUB1,SUB2,SUB3
```

Creates the new file MAIN with the contents of files SUB1, SUB2, and SUB3, in that order.

```
MERGE,MAINC,MAIN
```

Copies the contents of file MAIN into a new file MAINC.

Name: PRINT

Format: PRINT,filename,listdevice

Purpose: To print the contents of the specified file to the designated list device.

Comments: The list device operand may be C or L. If the list device operand is omitted, C is assumed. C specifies the console and L specifies the line printer.

Examples:
```
PRINT,JOE
PRINT,JOE,C
```

Prints the contents of file JOE to the console.

```
PRINT,JOE,L
```

Prints the contents of file JOE to the line printer.

Name: P6834

Format: P6834

Purpose: To program (burn) AMI 6834 PROMS. See Section 9.

Name: RENAM

Format: RENAM,oldfilename,newfilename

Purpose: To modify the specified file's file directory entry by replacing its existing file name with a new file name.

Comments: Only the file name area of the file's file directory entry is affected.

Example:
```
RENAM,MAIN5,MAIN
```

Renames the file MAIN5 with the name MAIN.

Name: RUN

Format: programname,inputfile,outputfile,option
RUN,programname,inputfile,outputfile, option

Purpose: To execute a 6800 object program.

Comments: RUN is the default command and the "RUN", is optional.

Name: <u>VIEW</u>

Format: VIEW,inputfile

Purpose: To display portions of a file on the CRT. The edit key (↓) advances the window to include the next 20 lines. <HOME> returns control to FDOS.

Comments: See MDC software reference for other options.

Except for the DELET command, references to file names can specify the drive number by following the character string with a colon (:) index (e.g., 1).

```
AP,IND:1,OUTL:1
```

Files IND and OUTL will be on the diskette in drive number 1.

Commands referring to the line printer with the option "L" require the standard Centronics printer. Only the DPA command utilizes the high speed Dataproducts printer.

Appendix B MDC Text Editor (ED) Summary

INTRODUCTION

A text editor is a program for creating or modifying files containing text — letters, numbers, punctuation marks, and perhaps some special characters. Such files include source programs, documentation, and test data.

The MDC program named ED is a display-oriented text editor. This means that text is always visible while it is being entered or changed, and that the effect of any editing function can be seen immediately.

ED is executed just like any other MDC program. To create a new (output) file by modifying an old (input) file —

```
ED,name-of-input,name-of-output
```

To create a new file solely from keyboard input —

```
ED,,name-of-output
```

For the purposes of editing, the MDC keyboard is divided into three separate sections. The "Main Section" group resembles a typewriter. The "Edit Key" group is the 4-by-5 arrangement to the right. These keys are used to control the operation of the editor. The "Function Key" group is the top row, from BOOT to the red RESET key. None of these keys is used during an editing run.

Certain keys on the edges of the Main Section have special purposes. As on a typewriter, the SHIFT key changes other Main Section keys from lower case to upper case only while it is held down. The ALL CAPS key acts like a typewriter's Shift Lock, but it affects only letters. A red light built into this key will indicate whether it is "off" or "on" from its most recent use.

OPERATION

When ED is executed, it will divide the screen into two areas or "windows" and separate them with a line of hyphens.

The top window — 20 lines by 80 characters — is the Text Window. In this area, part of your text, as supplied from the keyboard and possibly from a floppy disk file, will always be visible.

The bottom window — 5 lines — is the Control Window. It contains information about the editing process. As explained in detail later, the entries on the first line are the current mode, the cursor line and column, the size of the Selection, the size of the accumulated deletions in E4, the capacity of the Workspace for more separate lines or total characters, an error or warning message (or "OK"), and a message telling you what the editor is doing (or what its version number is).

The remaining lines of the Control Window — labeled E2, E3, and E4 are used for searching, substitution, and error correction.

Somewhere on the screen, you will always see the "cursor" — a blinking marker. This indicates the location which will be affected by the very next editing operation, such as text entry or modification. When ED starts up, the cursor will be on Line 1 of your text, and at Column 1 within that line.

As various editing operations move the cursor around, the Control Window will always show the line and column at which the cursor is located. The text character which occupies the same screen position as the cursor is said to be "under" the cursor; its neighbors are "before" and "after" the cursor.

Four of the Edit Keys are marked with arrows, and are used to move the cursor in the directions indicated. The cursor is never allowed to move sideways off the screen. It can go no further up than the start of the Workspace, and no further down than the end of your complete text. Any attempt to move the cursor beyond these limits will result in a beep from the MDC, and an error message of "CAN'T" — the editor's standard response to an impossible request.

ED provides margin controls and tabulation settings similar to those on an electric typewriter. In addition, it can do semi-automatic adjustment of old text to fit new margins.

To set a tab stop, move the cursor to the desired position and hit C/TABS. The hyphen (or margin indicator) for the current column will change to highlight. To clear a tab stop, use exactly the same procedure, and the column indicator will go back to lowlight. To clear all the tabs at once, simply hit CS/TABS.

DATA ENTRY

To supply new text from the keyboard, you would ordinarily set CHR Mode and type in line after line of text, ending each line with the RETURN key. Each character entered will appear under the cursor, pushing the cursor (and the rest of the line) to the right.

To correct trivial typing errors during this type of text entry, use the BACKSPACE key. It will remove the character just to the left of the cursor, allowing you to re-type into that same position, or to hit BACKSPACE again. More elaborate corrections can be made with the insertion and deletion features described later.

You can also set modes in which each character entered will replace the old character which was under the cursor, rather than pushing it to the right.

To insert a small number of characters anywhere within an existing line of text, get into either CHR Mode or WRD Mode, and position the cursor at the desired point of insertion. Now enter the new characters, and the editor will push over the rest of the line to make room.

To create a whole new line in the middle of existing text, position the cursor in Column 1 of the line just below the intended point of insertion. Hit S/INS. An empty line will appear, and the rest of your text will be adjusted downward.

To create a large block of new empty lines – perhaps for the insertion of a whole paragraph – position the cursor in Column 1, get into PAGE Mode (or PAGE*), and then hit INS. Everything below the cursor will be pushed off the bottom of the screen, opening up from 2 to 20 new lines.

To delete a character anywhere in your text, get into CHR or LIN Mode, position the cursor over the character, and hit the DEL key. The character will disappear from the screen, and the rest of the line will be closed up from the right. The cursor will not move, so repeated uses of DEL will delete consecutive characters until the end of the line is reached, at which point the editor will display a "CAN'T" message.

You can inform the editor that a certain block of text should be handled as a single unit during deletion, or movement, or the search and selection processes.

This block of text is called the "Selection," and its size (in complete lines) appears in the Control Window. When the "sel" indicator is in lowlight, there is no Selection; when it is in highlight, the Selection covers at least one character.

To mark the boundaries of a Selection, put the cursor over the first character needed, and hit SEL. Depending on the current mode, this will switch to lowlight one character, word, line, or page, and move the cursor to the next position still in highlight. Now put the cursor over the other boundary of the desired Selection, and hit SEL again. Everything between the area already in lowlight and the cursor will be added to the Selection.

It is often useful to have the cursor jump forward to the next occurrence of some "string" – a specific sequence of characters. Similarly, finding a string and replacing it with another one automatically is a great convenience.

The E2 line in the Control Window is used to specify the string to be searched for. Ordinarily, the searching process is limited to the text from the current cursor position to the end of the Workspace. However, the search can be extended right through to the end of your input file by using the special combination CS/E2 instead of E2. This may cause some disk input and output to occur.

– SUMMARY OF ED KEY FUNCTIONS –

	MODE –
CHR	Character
WRD	Word
LIN	Line
PAGE	Page
S/mode	Replacement (star) vs. insertion
	CURSOR, MARGINS, AND TABS –
UP or Down	One line or page
S/UP or S/DOWN	5 lines
C/UP or C/DOWN	10 lines
CS/UP or CS/DOWN	To extreme start or end
LEFT or RIGHT	One character or word
TABS or S/TABS	To next or previous tab stop
S/RIGHT	To end of this line
S/LEFT	To Left Margin, this line
RETURN	To Left Margin, next line
LF	To Left Margin, previous line
CS/LEFT	Set Left Margin
CS/RIGHT	Set Right Margin
C/TABS	Set or clear tab stop
CS/TABS	Clear all tab stops
	TEXT ENTRY AND MODIFICATION –
BACKSPACE	Allow correction of entry
ESC	Allow special character
INS	Insert a space or a page of spaces
S/INS	Insert an End-of-Line Marker
LIN	Insert new lines automatically
DEL	Delete a character, word, or page
S/DEL	Delete through End-of-Line Marker
E4	Reverse most recent deletions
S/MOV	Move words to Left Margin, next line
	SELECTION, SEARCH, AND SUBSTITUTION –
SEL	Set a boundary of the Selection
CS/SEL	Select to end of all text
S/SEL	Forget the Selection
C/LEFT or C/RIGHT	Cursor to a boundary of the Selection
C/DEL	Delete the Selection
C/INS or C/MOV	Insert or move the Selection
WIN	Change windows
S/E2 or S/E3	Clear E2 or E3
E2	Search in Workspace or Selection
E3	Substitution in Workspace or Selection
CS/E2 or CS/E3	Process through all text
	SPECIAL OPERATIONS –
CS/DPLX	Force output of Workspace
S/DPLX	Perform normal exit processing

Appendix C MDC-140 Logic Analyzer (LA) Summary

The MDC-140 Logic Analyzer is an advanced debug tool connected as a peripheral device of the AMI Microcomputer Development Center (MDC). Features include:

- Captures 1024 Events of 40 Parallel Inputs
- Captures Data Under Control of Programmable Start on Data Content
- Delay of -1024 to +64K Clock Periods
- Setup and Display of Captured Data Under Control of MDC Software
- Display Format is User-Definable; Captured Data Can Be Displayed in a Mix of Hex, Octal, Binary, ASCII and Special Formats for Support of S6800, S6820, S2000, 8080, etc.
- Four Clock Sources
- Input Voltage Range = -15 to +15 volts
- Adjustable Input Thresholds
- Data-Dependent Output for Triggering an Oscilloscope

TIPS FOR USE OF LOGIC ANALYZER IN DEBUGGING S2000 SYSTEMS

From FDOS, call up the Logic Analyzer software and prewritten display formats:

```
!LA,68.F
```

When the CRT asks for your choice of format, type:

```
2000
```

followed by two blanks.

To monitor the S2000 STATUS output, specify the clock source as "UCF" ("user clock, falling edge") and connect the red clock-source-input lead to pin 23, CLK, on the S2000 chip (set threshold level ≈2.0VDC). This will give you four lines of CRT display per S2000 instruction cycle. You may want to trim down the oscillator R or C in your prototype to compensate for the capacitance added in by the red clock-source lead, and insulate pin 23 on the 40-pin clip to keep its capacitance out of the S2000 oscillator circuit.

(An alternative approach which will result in the display of more instruction cycles — but less information per cycle — is to clip the red clock-source lead to SYNC (pin 35) and specify the clock source as "UCR". When S2000 is in the multiplexed mode, address and opcode data will be displayed, but control outputs won't be displayed.)

If your prototype uses the S2000 K lines as Touch-Control™ inputs, insulate those pins (30-33) on the Logic Analyzer clip, too; sense the K lines with low-capacitance probes or sense them through large-valued resistors having low capacitance (i.e., small surface-area). Make sure that the Logic Analyzer is powered-up before you clip it to your prototype system; the Analyzer's input protection diodes clamp when its power is down.

Next, set the A and B qualifiers. Set the SYNC qualifier = 0 to qualify on an address (possible only if S2000 is in its multiplexed mode). "E2 ESC A" will set the B qualifier equal to the A. Set the Analyzer delay to $3FE_{16}$ if you want the qualifying event to appear at the beginning of the trace memory display.

Consult the Logic Analyzer instruction manual for further information.

DOMESTIC

Western Area

100 East Wardlow Rd., Suite 203
Long Beach, California 90807
Tel: (213) 595-4768
TWX: 910-341-7668

3800 Homestead Road
Santa Clara, California 95051
Tel: (408) 249-4550
TWX: 910-338-0018

Central Area

500 Higgins Road, Suite 210
Elk Grove Village, Illinois 60007
Tel: (312) 437-6496
TWX: 910-222-2853

Suite Number 204
408 South 9th Street
Noblesville, Indiana 46060
Tel: (317) 773-6330

29200 Vassar Ave., Suite 303
Livonia, Michigan 48152
Tel: (313) 478-9339
TWX: 810-242-2903

725 So. Central Expressway, Suite B-5
Richardson, Texas 75080
Tel: (214) 231-5721
TWX: 910-867-4766

Eastern Area

237 Whooping Loop
Altamonte Springs, Florida 32701
Tel: (305) 830-8889
TWX: 810-853-0269

1420 Providence Turnpike, Suite 220
Norwood, Massachusetts 02062
Tel: (617) 762-6141
TWX: 710-336-0073

20 Robert Pitt Drive, Room 212
Monsey, New York 10952
Tel: (914) 352-5333
TWX: 710-577-2827

Axe Wood East
Butler & Skippack Pikes, Suite 3-A
Ambler Pennsylvania 19002
Tel: (215) 643-0217
TWX: 510-661-3878

INTERNATIONAL

England

AMI Microsystems, Ltd.
108 A Commercial Road
Swindon, Wiltshire
Tel: (0793) 31345 or 25445
TLX: 851-449349

France

AMI Microsystems, S.A.R.L.
124 Avenue de Paris
94300 Vincennes
France
Tel: (01) 374 00 90
TLX: 842-670500

Holland

AMI Microsystems, Ltd.
Calandstraat 62
Rotterdam
Holland
Tel: 010-361483
TLX: 844-27402

Italy

AMI Microsystems, S.p.A.
Via Pascoli 60
20133 Milano
Tel: 29 37 45 or 2360154
TLX: 843 32644

Japan

AMI Japan Ltd.
7th Floor Daiwa Bank Building
1-6-21, Nishi-Shimbashi
Minato-ku, Tokyo 105
Tel: (501) 2241
TLX: 781-0222-5351

Korea

KMI Inchon Export Industrial Estate
Block 1
Hyo Sung-Dong Buk-ku
Inchon, Korea

West Germany

AMI Microsystems, GmbH
Rosenheimer Strasse 30/32, Suite 237
8000 Munich 80, West Germany
Tel: (SV 089) 48 30 81
TLX: 841-522743

AMI

AMERICAN MICROSYSTEMS, INC.

3800 HOMESTEAD ROAD • SANTA CLARA CA 95051 • TELEPHONE (408) 246-0330 • TWX: 910-338-0018