TECHNICO SUPER STARTER MANUAL

Version 3, April 1978

(Applies to Version 3 Monitor and IIA)

TABLE OF CONTENTS

TECHNICO SUPER STARTER MANUAL

I PREFACE

The Super Starter System provides the basis of your own personal minicomputer system - including a 2704/2708 EPROM programmer. The Super Starter System is not a demo kit, but is the basis for a powerful computing machine. Because it incorporates the TI 9900 processor, it is compatible with the TI 990/4 minicomputer and other TI 990 family products.

Before proceeding with assembly of your kit, read through the entire manual and familiarize yourself with the features of this kit. Then, carefully assemble your kit; test it as described in the manual; apply power; and begin programming.

If you have any problems with this system, carefully recheck your assembly. (Are all resistor values correct? Are all chips aligned correctly? Is your terminal connected properly?) Since critical components are pretested, misassembly errors are the most likely cause of problems. If all else fails, Rosse Corporation (the designers of this system) are more than willing to provide whatever assistance they can to solve the problem. Just call them at ~~(703) 369-2734~~ 703 471 2530

This manual has been written for a user with little or no background in programming. In order to proceed directly into the manual, the reader is assumed to understand the following:

1. Binary, octal, binary coded decimal and

hexadecimal number systems.

2. Signed and unsigned binary arithmetic.

3. Boolean logic (AND, OR, EXCLUSIVE-OR).

4. ASCII character codes.

5. Basic concepts of the Texas Instruments TMS 9900.

The Super Starter System is organized for maximum user flexibility. The basic system includes 1,024 bytes of fused link read only memory (PROM), 512 bytes of read/write memory (RAM), sixteen input bits, sixteen output bits, and eight levels of interrupt. On board expansion is provided for 2,048 bytes of erasable read-only memory (2708 EPROM), an additional 1,024 bytes of 74S472 PROM, and an additional 1,536 bytes of RAM. The system also includes the necessary EPROM programming logic to program EPROMs (TI 2708, Intel 2708, or Intel 2704). The system has an EIA RS-232 or 20 milliamp current loop interface with automatic Baud rate determination for terminals up to a 9600 Baud rate. TI 733ASR, 743 and 745 terminals are available through Technico.

As you see, the Super Starter System is an excellant beginning, but you may be interested in future expansion boards. For example, with the 16K word (32K byte) expansion RAM (part number TEC-9900-MA-32) you can even use the Super Starter System to run our powerful relocating editor and assembler. To keep informed about future developements, just complete the enclosed reply card and mail it to us. It is our intention that you be completely satisfied with the products you receive

from Technico. If for any reason you are not pleased, let us know and we will make every effort to provide immediate corrective action.

Technico is a fully franchised distributor of Texas Instruments, therefore, all parts in your system are completely factory warranted. If you find a defective component, just return the part to us for replacement. We appreciate any suggestions you may have as to how we might improve both our products and services.

Best Regards,

William E Regan, Jr.
President
TECHNICO INC.

ROSSE CORPORATION

THE SUPER STARTER DESIGNERS

Who Are We?

Rosse Corporation is a growing consulting firm located in the metropolitan Washington, D.C. area. We specialize in designing microprocessor based systems.

What Can We Do For You?

We have a strong background in both hardware and software, and can help you to realize your objectives with microprocessors. You have already purchased one of our designs - The Super Starter Kit. This kit is a good example of the quality design approach used here at Rosse Corp.

What About Experience?

We have design experience with many of the popular microprocessors, namely: F8, COSMAC, Z80, Intel 8080, TI 9900, and Motorola 6800. Not just breadboards, but real products. Members of our staff are also hgihly published in the microprocessor area. In addition to our technical know-how, we are aware of the manufacturing aspects of microprocessors, and pride ourselves on producing the right documentation to simplify manufacturing. If you will take a moment to review the monitor, I think you will agree that it is both well written and well documented.

What Next?

If you have a specific application for microprocessors, why not give us a call. Maybe we can help you to mount the microprocessor learning curve.

Best Regards,

Jim Ferry

President

Rosse Corporation

II PARTS LIST - SUPER STARTER KIT

SOCKETS (Basic Kit)

2	24 Pin
33	16 Pin
16	18 Pin
6	20 Pin
1	64 Pin (may consist of 2 20 Pin and 2 12 Pin socket strips)
1	8 Pin
9	14 Pin

RESISTORS

2	10 ohm
2	47 ohm
2	220 ohm
1	680 ohm
1	470 ohm
7	1K ohm
1	2.2K ohm
17	3.3K ohm
2	4.7K ohm
1	6.8K ohm
3	10K ohm
1	20K ohm
1	47K ohm

1	51K ohm
1	100K ohm
4	15 ohm or 10 ohm
1	500K ohm - pot

CAPACITORS

2	22 pf
1	470 pf
1	620 pf or 680 pf
1	1000 pf
24	.1 mf
1	27 mf electrolytic or47 mf
4	2.2 mf electrolytic
1	1500 pf
2	.01 mf

DIODES

3	1N4148

TRANSISTORS

5	2N3904
1	2N3906
2	2N4401 (TI S111)

INTEGRATED CIRCUITS

1	74LS00
1	74LS04
3	74LS32
1	74LS40
2	74LS74
1	74123
1	74LS148
1	74LS156
1	74LS155
2	74LS251
2	74LS259
1	74S260
1	74LS362 (9904 CLOCK)
15	74LS367
1	74LS377
1	72555
4	TMS 4042-2 RAM
1	TMS 9900
2	74S472 Monitor PROM's (U47,U49)

MISC.

1	P.C. board
2	SPST switches

1	momentary contact switch
1*	terminal connection cable
2*	power connection cables
1	.47 microh. coil (looks like a large resistor)

* Not supplied with the basic kit. Purchased seperately.

III. BUS STRUCTURE

The TMS 9900 CPU has separate address and data buses. Since the address and data words are not multiplexed on a single bus, standard memories can be used with the TMS 9900 without an external address latch.

The TMS 9900 instructions build a 16-bit address word which describes a 64K x 8 address space. The least significant bit is used inside the CPU to select the byte and the other 15 address bits are passed to external memory to access a 32K x 16 address space. Thus, a TMS 9900 system has a 16-bit data word and a 15-bit address word. Byte addressing is transparent to the memory.

The address bus is also used to select an I/O bit and to pass the external control functions (IDLE, etc.). The external control functions are not required in most applications and therefore are not implemented in the Super Starter System. The address bus is used either to address memory ($\overline{\text{MEMEN}}$ low), to address an I/O bit, or to pass an external control function. The TMS 9900 interface signals are shown in Figure III-1

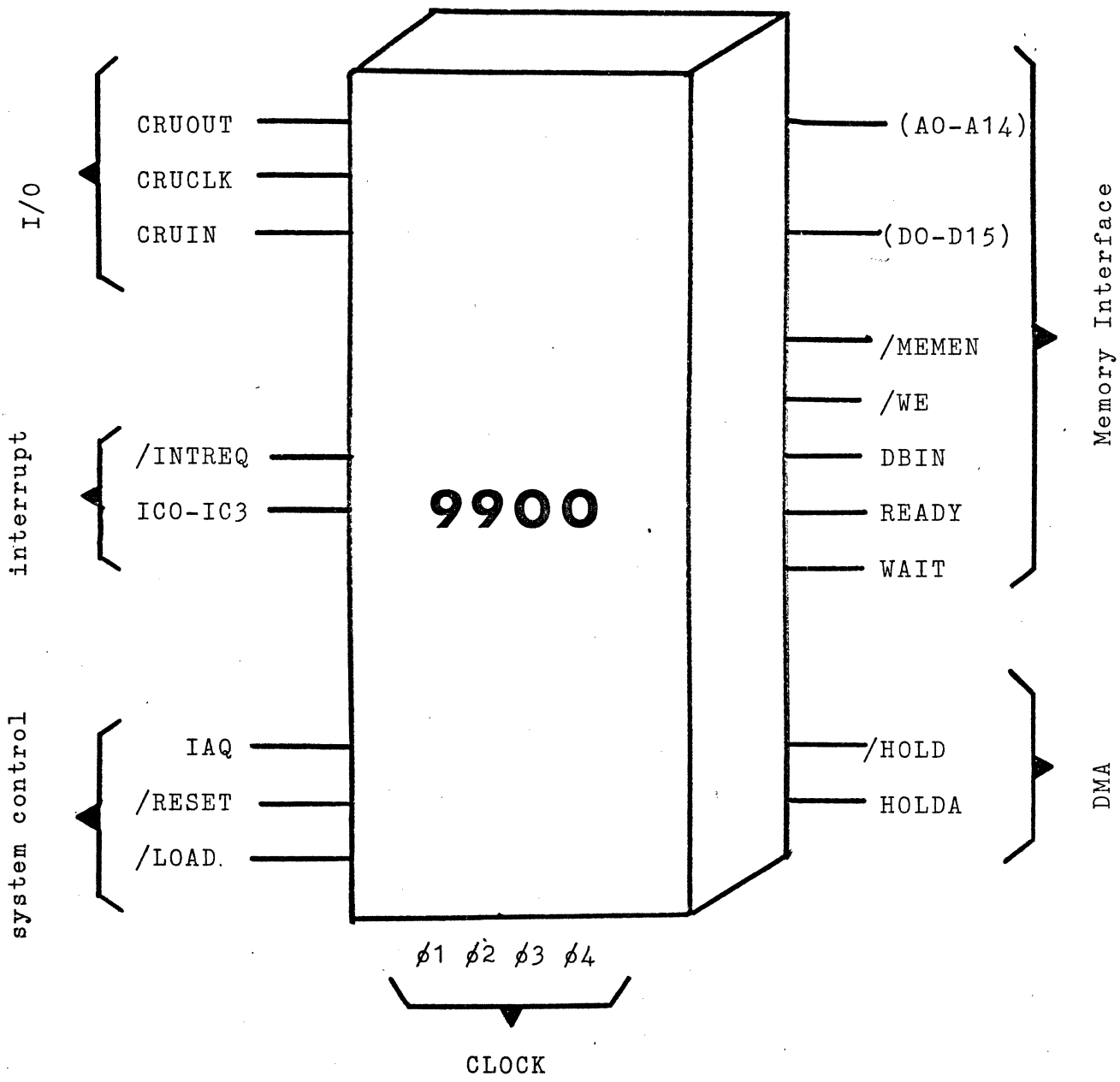
The data bus is used only to transfer data to and from the memory when $\overline{\text{MEMEN}}$ is low. The ROMs and RAMs are the only devices connected to the CPU data bus. DBIN indicates whether the data bus is the input or the output mode. The data bus is normally in the output mode (DBIN low), and the memory data outputs should be enabled only when DBIN is high.

The communications register unit (CRU) is a versatile command-driven I/O interface bus. The CRU employs three dedicated signals (CRUCLK, CRUOUT, and CRUIN) and the lower 12 bits of the address bus to interface with the CRU system. The CPU can set, reset, input, or test any bit in the CRU interface.

The CPU sets or resets an output bit by placing the bit address on the address bus, the output data bit on CRUOUT, and a clock pulse on CRUCLK. The CPU inputs or tests an input bit by placing the bit address on the address bus and testing CRUIN. Thus, CRU output operations are clocked by CRUCLK, while the CRU continuously decodes the CPU address to determine which signal is to be input to CRUIN. The current CPU instruction, however, determines whether or not the current CRUIN input is used. The Super Starter System provides 16 input and 16 output bits. One of the input bits and two of the output bits are used to control the RS-232/TTY interface and EPROM programmer.

The TMS 9900 has fifteen user interrupt levels in addition to the $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ functions. The presence of an interrupt is indicated by an external device driving $\overline{\text{INTREQ}}$ low and placing the priority code on IC0 through IC3. The Super Starter Kit provides priority encoding logic for eight unique levels of interrupt (U11,U25).

Figure III-1 TMS 9900 Interface Signals



IV. SYSTEM CONFIGURATION

A. MEMORY

The Super Starter Kit is equipped to handle three different types of memory:

PROM (74S472) - Four fusable link PROM's are available for permanent program storage. Two PROM's, which provide 512 words (1024 bytes) of program storage, are provided with the kit. These PROM's contain a powerful monitor to assist with program development.

EPROM (TMS2708/Intel 2708) - Two EPROM's are wired in parallel to provide an optional 1024 words (2048 bytes) of program storage. The two EPROM's may also be programmed using the software provided in the monitor. This provides a convenient means for saving user programs in a read only memory.

RAM (TMS4042) - Four 256 x 4 RAM's are provided with the kit, which provides 256 words (512 bytes) of read/write memory. This memory can be expanded to 1024 words (2048 bytes) by adding twelve more 256 x 4 RAM's.

The unique address decoding logic allows maximum flexibility in address assignment. A four input NAND (U6-74LS40) detects any reference to the last 2K words of memory. This signal partially enables an OR (U13-74LS32) and a one-of-four decoder (U8-74156). Address bit A4 determines whether the OR or the decoder will be enabled. The jumpers determine which memory will be address when the OR or the decoder is enabled. If these jumpers are installed as shown in the schematic, memory will be addressed as:

FC00-FFFF	ROM-1 (Monitor)
F800-FBFF	ROM-2 (IIA or Expansion PROM)
F000-F7FF	EPROM

An OR gate (U7-74S260) detects any reference to the first 1K of memory. This signal enables a second one of four decoder which determines exactly which section of memory is addressed. If the jumpers are installed as shown in the schematic, memory is addressed as:

0000-01FF	RAM-1 (Basic)
0200-03FF	RAM-2
0400-05FF	RAM-3
0600-07FF	RAM-4

The Super Starter Kit will ignore a reference to any address not shown above. The kit will also ignore addresses 0000-07FF if the jumper /FIRST K is removed. This may be useful if external RAM is added to the system.

The jumpers also allow the memory to be reorganized to suit the needs of any particular application. Some useful configurations are:

• JW1,JW2,JW3,JW4-JW11 (out)	Place EPROM at F800-FFFF
• JW5,JW6,JW7,JW8-JW11 (out) JW1(in)-JW5(out),JW2(in)-JW6(out) JW3(in)-JW7(out),JW4(in)-JW8(out)	Place EPROM at 0000-FFFF and RAM at F800-FFFF

This reorganizing is useful, but should be done carefully. Be certain that you do not enable two different memories with the same signal - this will damage the memory.

B. INPUT/OUTPUT

Two octal multiplexers (U3,U4-74LS251) and two addressable latches (U1,U2-74LS259) are used for CRU-based I/O. The I/O are addressed as bits 0-F (hex). Additional I/O may be added to the system by adding appropriate decoding logic.

One of the input bits (0) is used for RS-232/TTY input. If any application requires these bits, but does

not require the services of the monitor, the jumper may be removed, which deactivates this input.

One of the output bits (0) is used for RS-232/TTY output and one bit (1) is used to control the on-board EPROM programmer. As with the inputs, these may be disabled by removing the associated jumpers.

C. CLOCK GENERATION

The SN74LS362 clock generator (U10) provides the four-phase MOS timing signals for the TMS 9900. A single capacitor is used to determine the clock frequency. This is adequate for most applications, but if a more precise frequency is required, a crystal reference can be used.

A simple LC network is used to control the frequency overtone. The SN74LS362 also provides TTL compatible clock outputs. The RC network on the Schmitt-triggered D input provides a power-on reset fot the system in addition to the manual reset.

D. $\overline{\text{RESET}}$, $\overline{\text{LOAD}}$, AND $\overline{\text{INTREQ}}$

The $\overline{\text{RESET}}$,$\overline{\text{LOAD}}$, and $\overline{\text{INTREQ}}$ TMS 9900 inputs are used to alter the normal program execution sequence. The encoding logic (U11-74LS377,U25-74148) present the proper interrupt code to the IC0-IC3 line on the processor. It

also synchronizes the interrupt request.

The external load and reset signals are also directly input to the CPU after being synchronized. $\overline{\text{RESET}}$ is held active (low) for at least three clock cycles by the switch or the power-on RC network. $\overline{\text{LOAD}}$ is held active (low) for one instruction time as determined by the TMS 9900 IAQ output.

The load signal is used to enter the monitor. If switch one is in the load position, a load request is generated following any restart. This transfers control to the monitor since the load vector is at ROM locations FFFC-FFFF. If switch one is not in the load position, restarts use the normal restart vector at 0000-0003.

E. EPROM PROGRAMMER

A unique feature of the Super Starter is the on-board EPROM programmer for TMS 2708 or Intel 2708/2704 Erasable Read-Only Memories. The programming is enabled/disabled by switch three. When disabled, all programming requests are ignored by the hardware. When the programmer is enabled, bit 1 of the CRU output controls the programming. Another important feature is that both the EPROM's are programmed at one time. It is not necessary to program the even bytes, then the odd ones as it is

with a single EPROM programmer. Rather, a whole word is programmed at one time.

If programming is enabled (by switch three, and CRU I/O bit 1), then the processor can program any location by simply writing into it. When the write is detected (U12-74123), the address and data are held by placing the processor in wait and a program pulse is generated. After programming, the program mode can be reset to read, and the EPROMs verified. The EPROM must be programmed several times to insure data integrity. Do not continuously reprogram one location, as it may damage the EPROM. The recommended sequence is (R1,R2 preset to source and R3 to PROM destination):

```
            INCT    R2          ;advance end
            LI      R4,255      ;R4= repeat
LOOP 1      MOV     R1,R5       ;R5= start
            MOV     R3,R6       ;R6=PROM start
            ORI     R6,>F000    ;adjust for PROM
LOOP 2      MOV     *R5+,*R6+   ;Do one pass
            C       R5,R2
            JLE     LOOP 2
            DEC     R4          ;Do another pass
            JNE     LOOP 1
```

E. REAL TIME CLOCK

A real time clock oscillator is provided for software timing. The oscillator output is connected to bit (1) of the CRU input by jumper JW13. If the clock is not used the jumper can be removed. The clock can also be used to periodically interrupt the CPU, just connect the clock output to an interrupt input.

V. ELECTRICAL CHARACTERISTICS

The Super Starter Kit requires the following input power:

+5V	-	Maximum of 1.5 Amps
+12V	-	Maximum of .5 Amps
-5V	-	Maximum of .5 Amps

To program 2704/2708 EPROMs the following power must be supplied:

+28V	-	Maximum of 40 Milliamps

A power supply to power the Super Starter System plus a full 65K byte memory expansion is available (p/n TEC-9900-PP). Power ratings for this expanded unit are as follows:

+5V	-	Maximum of 5 Amps
+12V	-	Maximum of 3 Amps
-5V	-	Maximum of 2 Amps
+28V	-	Maximum of 40 Milliamps

VI. SYSTEM EXPANSION

The Super Starter Kit has been designed for ease of expansion. Since any choice of edge connector would seriously restrict the method of expansion, use of jacks was chosen instead. All of the critical signals, including those for a computer control panel, are available on 16-pin DIP sockets. The individual jacks and pin assignments are described in the paragraphs below.

The physical size of the Super Starter System, 7" x 16", is the identical size of standard wire wrap boards such as the Garry (p/n NCS-13). Flat Flexible Cable jumpers can be used to interface with this type of board to perform control functions. The TEC-9900-MA-32K byte memory add-on boards are physically also the same size. Since the TEC-9900-SS is fully buffered, memory expansion can be accomplished by merely jumping to the memory boards (p/n TEJ-99DA-12). Program loading to the TEC-9900-SS can be accomplished by interfacing a terminal or RS232 cassette tape into jack 10 of the kit. Refer to the monitor section for details regarding program loading.

A. J4 (Address Bus)

J4 1	Address Bit 0 (MSB)
J4 2	Address Bit 1
J4 3	Address Bit 2
J4 4	Address Bit 3
J4 5	Address Bit 4
J4 6	Address Bit 5
J4 7	Address Bit 6
J4 8	Address Bit 7
J4 9	Address Bit 8
J4 10	Address Bit 9
J4 11	Address Bit 10
J4 12	Address Bit 11
J4 13	Address Bit 12
J4 14	Address Bit 13
J4 15	Address Bit 14 (LSB)
J4 16	Unused

B. J9 (Data Bus)

J9 1	Data Bit 0 (MSB)
J9 2	Data Bit 1
J9 3	Data Bit 2
J9 4	Data Bit 3
J9 5	Data Bit 4
J9 6	Data Bit 5
J9 7	Data Bit 6
J9 8	Data Bit 7
J9 9	Data Bit 8
J9 10	Data Bit 9
J9 11	Data Bit 10
J9 12	Data Bit 11
J9 13	Data Bit 12
J9 14	Data Bit 13
J9 15	Data Bit 14
J9 16	Data Bit 15 (LSB)

C. J8 (Interrupt Control)

J8	1	Interrupt level 7
J8	2	Interrupt level 6
J8	3	Interrupt level 5
J8	4	Interrupt level 4
J8	5	Interrupt level 3
J8	6	Interrupt level 2
J8	7	Interrupt level 1
J8	8	Interrupt level 0 (highest priority)
J8	9 to 16	Ground (unused)

D. J6 (Control Signal Group 1)

J6	1	Ready (In)
J6	2	/HOLD (In)
J6	3	DBIN (Out)
J6	4	/WE (Out)
J6	5	/MEMEN (Out)
J6	6	HOLDA (Out)
J6	7	WAIT (Out)
J6	8	LOAD (In)
J6	9	/RESET (In)
J6	10	/RESET (Out)
J6	11	/LOAD (Out)
J6	12	IAQ (Out)
J6	13	CRUIN
J6	14	CRUOUT
J6	15	CRUCLOCK
J6	16	GND

E. J7 (Control Signal Group 2)

J7	1	/Phase one	TTL Level Processor Clocks
J7	2	/Phase two	
J7	3	/Phase three	
J7	4	/Phase four	
J7	5	Oscillator out	
J7	6	Oscillator In	
J7	7 to 16	Unused	

F. J10 (RS-232/TTY Interface)

J10	1	Pin 1	
J10	2	2	
J10	3	3	
J10	4	5	
J10	5	6	
J10	6	7	
J10	7	8	E.I.A. RS-23C Connector
J10	8	21	Pin Assignments
J10	9	22	
J10	10	23	
J10	11	24	
J10	12 to 16	unused	

G. J2 (Input Port 1/CRUIN)

J2 1	Bit 0 (LSB)	TTY IN
J2 2	Bit 1	CLOCK IN
J2 3	Bit 2	
J2 4	Bit 3	
J2 5	Bit 4	
J2 6	Bit 5	
J2 7	Bit 6	
J2 8	Bit 7	
J2 9	Bit 8	
J2 10	Bit 9	
J2 11	Bit 10	
J2 12	Bit 11	
J2 13	Bit 12	
J2 14	Bit 13	
J2 15	Bit 14	
J2 16	Bit 15 (MSB)	

H. J1 (Output Port 1/CRUOUT)

J1	1	Bit 0 (LSB) TTY OUT
J1	2	Bit 1 /PROGRAM ENABLE
J1	3	Bit 2
J1	4	Bit 3
J1	5	Bit 4
J1	6	Bit 5
J1	7	Bit 6
J1	8	Bit 7
J1	9	Bit 8
J1	10	Bit 9
J1	11	Bit 10
J1	12	Bit 11
J1	13	Bit 12
J1	14	Bit 13
J1	15	Bit 14
J1	16	Bit 15 (MSB)

I. J5 (Input Power)

J5 1 to 4, 13 to 16	+5V
J5 5 to 6, 11 to 12	+12V
J5 7, 10	-5V
J5 8, 9	+28V (used only to program EPROMs)

J. J3 (Input Ground)

J3 1 to 16	Ground

WARNING

Be careful when applying power to J3/J5. A misconnection will seriously damage the system! Also, all unused pins are grounded.

VII. MIGHTY MONITOR

The Super Starter - Mighty Monitor provides the following comprehensive set of commands:

- A Alter the contents of RAM
- B Breakpoint set/restore
- C Copy memory to memory
- D Dump memory to display or terminal
- G Go to program in memory
- H Hexadecimal Arithmetic
- I Inspect CRU bit
- L Load program from terminal
- M Modify CRU bit
- P Program EPROM
- S Snap definition
- W Workspace dump

The Mighty Monitor accepts input from and produces output for a serial Asynchronous ASCII terminal or teletypewriter. To insure maximum flexibility in the choice of a terminal, the monitor always generates two stop bits after each character and user controlled delay after each carriage return. The Baud rate of the terminal is determined automatically during the start up of the monitor. After a reset (power-on or manual) the monitor will wait for the user to enter the letter 'X'. When the letter 'X' is entered, the monitor automatically calculates the Baud rate (110 to 9600) and begins normal operation. During normal operation, the monitor prompts the user to enter a command by typing a question mark at the beginning of a new line. The first entry by the user must be one of the allowable command codes (A, B, C, D, G, H, I, L, M, P, S, W), and is followed by the arguments in hexadecimal notation. Multiple arguments are separated from one another by an arbitrary sequence of symbols or characters, except for hexadecimal digits (0-9,A-F) or carriage return. The command is terminated by any non-hexadecimal digit (including carriage return) after the last argument.

If an argument is typed with more than the required number of digits (usually four), the monitor will use only the right-most digits. This feature can be used to correct input errors. If any argument is shorter than required, the left-most digits will automatically be filled with zeros.

The monitor uses certain locations in memory to store breakpoint information, etc. The monitor memory map is shown in Figure VII-1. To operate in half-duplex mode (no character echo) change the echo flag to zero using monitor's Alter command. To insert a delay after carriage return, enter the required delay in the delay word (again using the Alter). The total carriage return delay is Delay*6 microseconds. If you do not know the delay required for your terminal, it can be determined experimentally by increasing the delay until no characters are lost after a carriage return. If you modify any of the other locations used by the monitor, the monitor may not function properly.

A detailed description of each command is provided in the following paragraphs, along with an example of its usage.

NOTE: If you are using a TI Silent 700 which is equipped for 1200 Baud operation, a special monitor is available for communication with that terminal. Inquire at Technico for further details regarding the Silent 700 monitor.

FIGURE VII-1 MEMORY MAP

Address (hex)	Contents
0-3	Interupt vector - level 0
4-7	- level 1
8-B	- level 2
C-F	- level 3
10-13	- level 4
14-17	- level 5
18-1B	- level 6
1C-1F	- level 7
20	Carriage return delay
22	Echo flag
24	Terminal speed
26	No. of words for a break
28	User instruction - one
2A	- two
2C	- three
2E	Return branch (two words)
32	Next stop
34	Stop increment
36	Maximum number of stops
38	Register bounds - first
3A	- last
3C	Memory bounds - first
3E	- last

FIGURE VII-1 MEMORY MAP (continued)

Address (hex)	Contents
40-43	XOP 0
44-47	XOP 1
48-4B	XOP 2
4C-4F	XOP 3
50-53	XOP 4
54-57	XOP 5
58-5B	XOP 6
5C-5F	XOP 7
60-63	XOP 8
64-67	XOP 9
68-6B	XOP 10
6C-6F	XOP 11
70-73	XOP 12
74-77	XOP 13
78-7B	XOP 14
7C-7F	XOP 15
80-9F	Monitor Workspace
A0-AF	XOP Workspace (only 8 registers)

FIGURE VII-1 MEMORY MAP (continued)

Address (hex)	Contents
B0	User - R0
B2	- R1
B4	- R2
B6	- R3
B8	- R4
BA	- R5
BC	- R6
BE	- R7
C0	- R8
C2	- R9
C4	- R10 (RA)
C6	- R11 (RB)
C8	- R12 (RC)
CA	- R13 (RD)
CC	- R14 (RE)
CE	- R15 (RF)

ALTER - The contents of memory may be examined and modified.

Format: A aaaa

Procedure:

1. Type "A".
2. Type the byte address (aaaa) of the memory location to be examined (in hex) followed by a space.
3. The monitor will display the contents of the specified location in hexadecimal format (followed by a hyphen).
4. If you wish to change the contents of this location, simply enter the desired hexadecimal value followed by a space or carriage return. If not, just type space or carriage return and the monitor will display the contents of the next sequential address. If a space is typed, the next value will be displayed on the current line, but if a carriage return is typed, both the next byte's address and contents are displayed on the next line.
5. Repeat steps 3-4 until all desired locations have been examined or modified. To exit this routine depress the BREAK key on the terminal (or type an ASCII NUL).

Note:

1. If the monitor was entered from a BREAKPOINT, ALTER can be used to examine or modify the

working registers. Refer to the memory map command for a definition of the addresses used.

2. ALTER can also be used to examine EPROM or PROM, but it can not be used to modify them.

Example: The following sequence will alter locations #400 and #404 with #FF. Locations #401-403 are unchanged (user's entries are underlined).

?A400 00-FF 11-_22-_33-_44-FF 55-

BREAKPOINT - A breakpoint or trap may be set in any user program that is stored in RAM. Whenever the processor encounters the substituted trap instruction, the state of the machine is saved and control is transfered back to the monitor for user action.

Format: B aaaa,n

Procedure:

1. Type "B".
2. Type the hexadecimal address (aaaa) of the location to be trapped, followed by a delimiter. (aaaa) must be a word address (even number).
3. Type the number of words (n) to be removed for the trap. This should be the number of words (1, 2, or 3) currently occupied by the trapped instruction.
4. Type space or carriage return. The monitor will remove any prior trap and then install the new trap.

Note:

1. If the existing trap is to be removed without setting a new one, the address is omitted and the command terminated by carriage return.
2. After entering the monitor from a trap, the GO command can be used to resume execution (see GO command, discussed later).

3. The contents of the user's workspace registers are saved whenever a breakpoint is encountered. The contents of the registers can be examined or modified using the ALTER command. The Monitor Memory Map shows where the active registers are saved. Note: If the workspace pointer is changed by the user program, the registers will be located at the address in the workspace pointer.
4. Relative jump instructions should not be breakpointed if a return GO is to be used or if a SNAP is established.

COPY - The contents of a block of memory may be copied into another area of memory.

Format: C ssss, eeee, dddd

Procedure:
1. Type "C".
2. Type (in hex) the starting address (ssss) followed by a delimeter, and then the ending address (eeee) followed by a delimeter of the block of memory you wish to be copied.
3. Type (in hex) the destination address (dddd) followed by a space or carriage return. For a normal copy operation, the destination address should not be within the bounds of the block of memory that you are copying.

Note:
1. The copy command can be used to set a block of memory to a specified constant. This is done in two steps. First, place the desired constant in the start location (using the ALTER command). Then perform a "C ssss, eeee-1, ssss+1", where (ssss) is the start address and (eeee) is the end address of the block.

Example:
1. The following command will copy #410-420 into #430-440.

?C410,420,430

2. To set all locations #410-41F to zero, the following commands are used.

```
?A410 11-00 22          (sets 410 = 00)
?C410,41E,411           (clears 410-41F)
```

Note that #41E is one less than the ending address #41F and #411 is one greater than the starting address #410.

DUMP - The contents of a block of memory may be listed on the printer.

Format: D ssss, eeee

Procedure:
1. Type "D".
2. Type (in hex) the starting address (ssss) followed by a delimiter and then the ending address (eeee) of the memory to be listed.
3. Terminate the command by typing a space or a carriage return. The monitor will now list the block of memory you requested, sixteen bytes per line.

Note:
1. The ending address may be omitted (and the command terminated by a carriage return), in which case the monitor assumes that the ending address is the end of memory (65535, or #FFFF).
2. The dump may be stopped at any time by depressing any key on the terminal.
3. The LOAD command can reload the program if the dump is recorded, on paper tape or other media.

Example: Both of the following examples will dump the entire memory:

```
?D0,FFFF
?D0
```

GO - Control can be transferred to a specified word in memory. Execution can also be resumed after a breakpoint trap.

Format: G aaaa

Procedure:
1. Type "G".
2. Type the hex address (aaaa) where control is to be transferred. (aaaa) must be a word address (even).
3. Terminate the command by typing a space or carriage return. The monitor will now transfer control to address (aaaa).

Note:
1. The address (aaaa) may be omitted (and the command terminated by a carriage return) in which case the monitor will assume that a trap was reached, restore the state of the machine, execute the instruction removed for the trap, and return to the point following the trap. This feature should be used only if the monitor was entered by a trap and the location being trapped does not contain a relative jump.
2. Do not set a new breakpoint, then issue a GO without an address, as this will transfer control to the wrong location..

Example: The following will transfer control to location #106.

```
?G106
```

HEXADECIMAL ARITHMETIC - Calculate the hexadecimal sum and difference of two numbers.

Format: H aaaa, bbbb

Procedure:
1. Type "H".
2. Type the two hexadecimal operands (aaaa) and (bbbb) separated by a delimiter and followed by a space or carriage return.
3. The monitor will now calculate and display (xxxx)=(aaaa)+(bbbb) and (yyyy)=(aaaa)-(bbbb) as follows:

H+=(xxxx) H-=(yyyy)

Example: This command is useful for calculating the destination address for a jump. If the jump instruction #1047 is at, say, location #1234 then the destination address is (#1234+2)+2*47. This sum is calculated in two steps as follows:

Step 1) ?H1236,47

H+=127D H-=11EF

Step 2) ?H127D,47

H+=12C4 H-=1236

Note that the jump displacement is relative to the next sequential instruction (#1236) not the jump itself.

INSPECT - A CRU bit may be displayed on the terminal.

Format: Ibb

Procedure:
1. Type "I".
2. Type the CRU bit (bb) to be tested, followed by a space.
3. The monitor will display the selected CRU bit.

Example: Display CRU bit 5 (assume it is set).

```
?I5 1
```

LOAD - A program file may be loaded into memory from paper tape or any other terminal storage media.

Format: L

Procedure:
1. Type "L".
2. Initiate the input (e.g. the paper tape).

Note:
1. The LOAD command will reload programs produced by the DUMP command. The dumped program will be reloaded into the same area of memory that it was dumped from.
2. If you do not wish the input to be listed as it is loaded, simply set locations #22, #23 to zero. This will suppress the monitor's echo.
3. The carriage return delay should be set to zero (i.e., #20, #21) prior to loading.

MODIFY - A CRU bit may be set or cleared.

Format: M bb,v

Procedure:
1. Type "M".
2. Type the desired CRU bit (bb) followed by a delimeter.
3. Type the bit value that you desire (0=clear, 1=set).

Example: Set bit 12 to 0

?M12,0

PROGRAM - Program a 2708 EPROM.

Format: P aaaa, bbbb, cccc

Procedure:

1. Type "P".
2. Type (in hex) the starting address (aaaa) of the area to be placed in EPROM followed by a delimiter, and then by the ending address (bbbb) followed by a delimiter.
3. Type the starting address of the EPROM area to be programmed followed by a space or carriage return.(0000 is the first EPROM location)
4. The monitor will now program the EPROM's.

Note:

1. The starting address of the EPROM's, for programming purposes only, is zero.
2. The monitor always programs both EPROMs. Even bytes in one and odd bytes in another.
3. To program only selected locations, place #FF in any location not to be programmed. Since the erased EPROM has #FF in all locations, this will not change the EPROM.
4. The ending address (bbbb) MUST BE EVEN.

Example: Program a copy of the monitor.

?P FC00, FFFE,0

SNAP - Snap parameters can be established.

Format: S ffff, iiii, nnnn

?R r1, r2

?M M1, M2

Procedure:

1. Type "S".
2. Type the first time a snap is desired (ffff) followed by a delimiter, then the increment between snaps required (nnnn) followed by a delimiter, and finally the total number of snaps (nnnn) followed by a delimiter.
3. When the monitor types "?R", enter the workspace registers to be snapped. If no registers are to be snapped, then type a carriage return.
4. When the monitor types "?M", enter the area of memory to be snapped. If no memory is to be snapped, then type a carriage return.

Note: Prior to establishing a snap a breakpoint must be set.

Example: The following sequence will snap registers R1-R3 and memory area #100-105 after the fourth execution of the instruction at #130. After the initial snap, it will snap every third time until a total of six snaps.

```
?B130,1          (first set trap)
?S4,3,6          (set trap)
?R1,3
?M 100,105
```

The sample output given on the next page illustrates the use of A, B, and S commands. The A command is used to enter a program into memory. This program will decrement R1, R2, and R3. The B command is used to set a Breakpoint trap at #130 (which contains a 1 word instruction). The S command specifies a snap of R1 through R3 and memory locations #100 through #105 to be taken just prior to the 4th, 7th, 10th, 13th, 16th, and 19th times that the instruction at location #130 is executed.

```
?A130 2C-06 00-01 06-06 02-02 06-06 03-03 10-10 FC-FC 8A-
?B130,1
?S4 3 6
R?1 3
M?100 105
?G130

PC=0132 WP=00B0 ST=D000
R1=00B0 R2=2B36 R3=0D38
0100: 02 03 00 01 C0 C1

PC=0132 WP=00B0 ST=D000
R1=00AD R2=2B33 R3=0D35
0100: 02 03 00 01 C0 C1

PC=0132 WP=00B0 ST=D000
R1=00AA R2=2B30 R3=0D32
0100: 02 03 00 01 C0 C1

PC=0132 WP=00B0 ST=D000
R1=00A7 R2=2B2D R3=0D2F
0100: 02 03 00 01 C0 C1

PC=0132 WP=00B0 ST=D000
R1=00A4 R2=2B2A R3=0D2C
0100: 02 03 00 01 C0 C1

PC=0132 WP=00B0 ST=D000
R1=00A1 R2=2B27 R3=0D29
0100: 02 03 00 01 C0 C1

?
```

VIII. ASSEMBLY

The Super Starter Kit is designed for easy assembly. You don't have to be a microprocessor wizard to build and test your computer. If you carefully follow the assembly instructions, your computer will operate properly - the first time that power is applied.

To be sure that you don't make assembly errors, we highly recommend that you familiarize yourself with the kit prior to assembly. The best way to do this is to study the manual before proceeding. After you have read the manual - all of it- you are set to begin. The following simple precautions will help minimize the chances of error:

- Use care in handling the integrated circuits. All of the integrated circuits (I.C.) will be seriously damaged by static discharge. Carpeted areas are a problem. Even a minor static shock will ruin most I.C.'s.

- Use the proper tools, and exercise care when soldering the components. In particular, use a low wattage iron - no more than 30 watts. Use only rosin-core solder. Acid core solder will ruin the kit. Keep the tip of your iron clean - a damp sponge is ideal for this purpose.

- Never remove or install components when power is applied to the board. If you do, you will almost surely burn out some of the I.C.'s.

- Prior to starting to assemble your kit, gather the necessary tools. The Super Starter Kit does not require an extensive set of tools. The following set should be sufficient:

    1. needle-nose pliers
    2. diagonal cutters
    3. soldering iron (25 or 30 watts)
       Do not use a soldering gun because it gets much too hot.
    4. solder (remember - use rosin-core)
    5. volt-ohmmeter or continuity checker

You are now ready to assemble the computer. Follow each of the instructions precisely, and in the order shown. All of the components are installed on the silk-screened side of the board, and are soldered on the other side. Be sure you have the board oriented with the silk-screen printing down when soldering components.

STEP 1 - Parts Verification

Separate and check all parts against the parts list of section II. If you find that any parts are missing, notify us immediately, and a replacement will be sent to you. Keep the parts separated for ease of assembly - paper cups or a small muffin tin is ideal storage.

STEP 2 - Install Sockets

Install the I.C. sockets shown below. Be certain to orient the socket properly. Each socket will have a distinctive marking to indicate pin one. Some sockets have a cut-off corner, others have a notch in the end with pin one. In any case, pin one must be aligned with the pin one indication on the printed circuit board as shown in Figure VIII-1.

( ) J1 16 pin socket
( ) J2 16 pin socket
( ) J3 16 pin socket
( ) J4 16 pin socket
( ) J5 16 pin socket
( ) J6 16 pin socket
( ) J7 16 pin socket
( ) J8 16 pin socket
( ) J9 16 pin socket
( ) J10 16 pin socket
( ) U23 64 pin socket (CPU)
( ) U22 18 pin socket (RAM)
( ) U37 18 pin socket (RAM)
( ) U18 18 pin socket (RAM)
( ) U33 18 pin socket (RAM)

( ) U10 20 pin socket (clock)

( ) U47 20 pin socket (PROM)

( ) U49 20 pin socket (PROM)

STEP 3 - Install Resistors

The resisotrs should be installed in the order indicated below. Bend the leads to fit the distance between the mounting holes, insert the leads, push the resistor snug against the board, carefully solder it in place, and then trim off the excess leads. All resistor values are in ohms, and all resistors are $\frac{1}{4}$ watt. For your convenience, the color code for each resistor is also shown. Figure VIII-2 illustrates the standard resistor color code.

( )	R1	3.3K	orange,orange,red
( )	R2	3.3K	orange,orange,red
( )	R3	3.3K	orange,orange,red
( )	R4	3.3K	orange,orange,red
( )	R5	3.3K	orange,orange,red
( )	R6	3.3K	orange,orange,red
( )	R7	3.3K	orange,orange,red
( )	R8	3.3K	orange,orange,red
( )	R9	10K	brown,black orange
( )	R10	10	brown,black,black
( )	R11	20K	red,black,orange
( )	R12	51K	green,brown,orange
( )	R13	3.3K	orange,orange,red

( )	R14	3.3K	orange,orange,red
( )	R15	47	yellow,violet,black
( )	R16	10K	brown,black,orange
( )	R17	1K	brown,black,red
( )	R18	4.7K	yellow,violet,red
( )	R19	3.3K	orange.orange,red
( )	R20	220	red,red,brown
( )	R21	220	red,red,brown
( )	R22	680	blue,grey,brown
( )	R23	10	brown,black,black
( )	R24	3.3K	orange,orange,red
( )	R25	1K	brown,black,red
( )	R26	3.3K	orange,orange,red
( )	R27	1K	brown,black,red
( )	R28	100K	brown,black,yellow
( )	R30	1K	brown,black,red (R29 not used)
( )	R31	3.3K	orange,orange,red
( )	R32	3.3K	orange,orange,red
( )	R33	4.7K	yellow,violet,red
( )	R34	6.8K	blue,gray,red
( )	R35	470	yellow,violet,brown
( )	R36	3.3K	orange,orange,red
( )	R38	3.3K	orange,orange,red (R37 will be
( )	R39	2.2K	red,red,red done later)

( )	R40	1K	brown,black,red
( )	R41	47K	yellow,violet,orange
( )	R42	3.3K	orange,orange,red
( )	R43	47	yellow,violet,black
( )	R44*	10 or 15	brown,black,black or brown,green,black
( )	R45*	10 or 15	brown,black,black or brown,green,black
( )	R46*	1K	brown,black,red
( )	R47*	10 or 15	brown,black,black or brown,green,black
( )	R48*	10 or 15	brown,black,black or brown,green,black

* Revision 'B' P.C. Board only

FIGURE VIII-1 SOCKET INSTALLATION

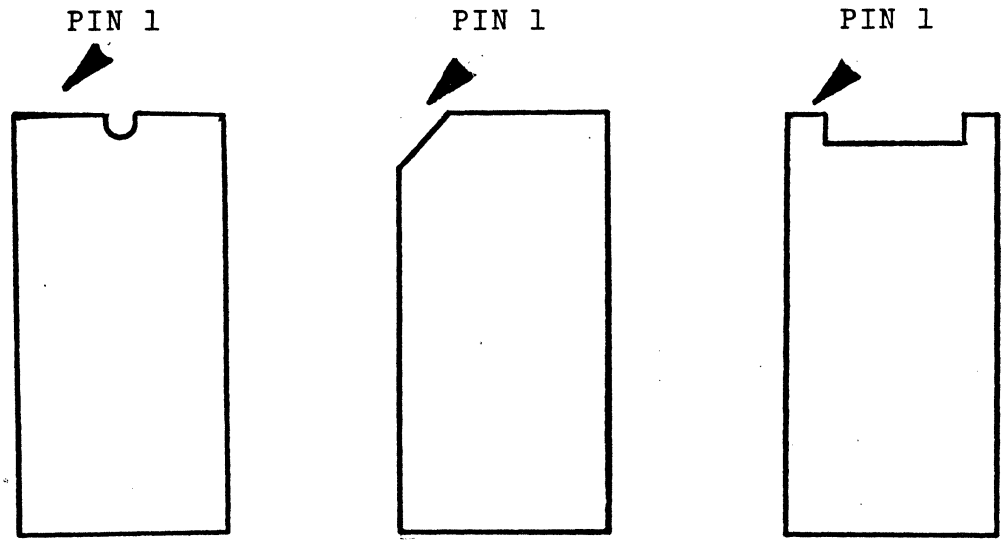
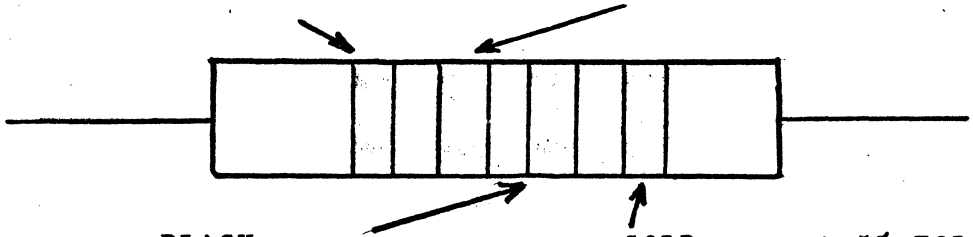


FIGURE VIII-2. RESISTOR COLOR CODE

E.I.A. COLOR CODE INDICATOR

BLACK	- 0	BLACK	- 0
BROWN	- 1	BROWN	- 1
RED	- 2	RED	- 2
ORANGE	- 3	ORANGE	- 3
YELLOW	- 4	YELLOW	- 4
GREEN	- 5	GREEN	- 5
BLUE	- 6	BLUE	- 6
VIOLET	- 7	VIOLET	- 7
GRAY	- 8	GRAY	- 8
WHITE	- 9	WHITE	- 9



BLACK	-	GOLD	= ± 5% TOL.
BROWN	- 0	SILVER	= ± 10% TOL.
RED	- 00	NO BAND	= ± 20% TOL.
ORANGE	- 000		
YELLOW	- 0000	Example:	RED RED ORANGE
GREEN	- 00000		2 2 000 =22K
BLUE	- 000000		

STEP 4 - Install Pot

Install the 500K ohm pot at location R37. This pot controls the speed of the real time clock and can be set as required by your software. It must be oriented as shown on the board. That is, the screw should be positioned according to the marking on the board.

STEP 5 - Install Capacitors and Inductor

Each capacitor should be installed in the order indicated below. Insert the leads into the board, gently push the capacitor shug against the board, carefully solder it in place, and then trim off the excess leads. All values shown below are in microfarads unless otherwise indicated. Many of these capacitors are bypass capacitors - they minimize electrical noise on the board and insure "glitch" free operation. If disc capacitors are enclosed, the value is printed on them. If color coded capacitors are included, the resistor color code is used to determine the value.

( )	C1	.1
( )	C2	.1
( )	C3	.1
( )	C4	.1
( )	C5	.1
( )	C6	.1
( )	C7	.1
( )	C8	.1
( )	C9	.1
( )	C10	.1
( )	C11	.1
( )	C12	.1
( )	C13	.1

( ) C14 .1

( ) C15 .1

( ) C16 22pf

( ) C17 27mf or 47 mf (electrolytic -

( ) C18 22pf see note below)

( ) C19 .1

( ) C20 .1

( ) C21 .1

( ) C22 620pf

( ) C23 .1

( ) C24 2.2 (electrolytic - see note below)

( ) C25 .1

( ) C26 .1

( ) C27 470pf

( ) C28 .1

( ) C29 1000pf

( ) C30 .1

( ) C31 2.2 (electrolytic - see note below)

( ) C32 .01

( ) C33 2.2 (electrolytic - see note below)

( ) C34 2.2 (electrolytic - see note below)

( ) C35 1500pf

( ) C36 .01

( ) L1 .47 microh. coil

All of the electrolytic capacitors must be properly oriented on the board. One end of the capacitor is marked with a "+". This end of the capacitor must be positioned as indicated on the board. Since these are used to filter the power, proper orientation is essential or the kit will be damaged when power is applied.

In some cases the capacitance value is printed on the capacitor in the form of a three digit number, with the first two digits indicating the number of zeroes to be added to the right (as in the resistor color code). For example, 470 indicates 47 picofarads and 104 indicates 100000 picofarads, 1.e., 0.1 microfarads. As another example, C22 is nominally 620 pf. But your kit may have a capacitor marked CK05/681K. The 681 means that the capacitor is 680 pf. This is the capacitor that you would use for C22 in this case.

STEP 6 - Processor Power

Regulator U29 was used for earlier 6V processors and is no longer required. To provide additional power filtering, install jumpers 's' and 'T'. A scrap resistor lead makes an excellent jumper.

STEP 7 - Install the Diodes

All of the diodes are the same - 4148. They must be properly oriented on the board. One end of the diode is marked with a band. This banded end must be positioned as shown on the board. Since diodes are used to prevent current flow in one direction, reversing them will burn out your kit!

( ) CR 1 4148

( ) CR 2 4148

( ) CR 3 4148

STEP 8 - Install the Transistors

There are three different types of transistors in the Super Starter Kit, so be careful to install the proper type. Each transistor has three pins, and must be properly oriented. If you look at the transistor from the side that is flat (pins facing down), the pins are (from left to right) - emitter, base, and emitter Be sure to orient the emitter pin as shown on the board. If not properly oriented, the transistor will be damaged.

( )	Q 1	2N3906
( )	Q 2	2N3904
( )	Q 3	2N3904
( )	Q 4	2N3904
( )	Q 5	2N3904
( )	Q 6	2N4401 (or TIS111)
( )	Q 7	2N4401 (or TIS111)
( )	Q 8	2N3904

STEP 9 - Install the I.C.'s

All integrated circuits must be properly positioned. Pin one of the IC is indicated by a small dot or number one in the corner, or by a notch at one end of the chip. Pin one must be positioned as shown on the board. If you purchased a fully socketed kit, first install the proper size socket, and then install the IC in the socket (Be sure to position the socket as described in Step 2.). DO NOT solder the socket with the I.C. installed. When you solder the I.C.'s be careful not to create a solder "bridge" between adjacent pins. This type of soldering error is the most common kit building error, and can be very difficult to locate. Check each joint after you solder it. Install the I.C.'s listed below.

( ) U1 74LS259
( ) U2 74LS259
( ) U3 74LS251
( ) U4 74LS251
( ) U5 74LS155
( ) U6 74LS40
( ) U7 74S260
( ) U8 74LS156
( ) U9 74LS74
( ) U10 74LS362 (socket installed in Step

( ) U11 74LS377

( ) U12 74123

( ) U13 74LS32

( ) U14 72555

( ) U18 4042 (socket installed in Step 2)

( ) U22 4042 (socket installed in Step 2)

( ) U24 74LS367

( ) U25 74LS148

( ) U26 74LS32

( ) U27 74LS00

( ) U28 74LS04

( ) U29 No Longer Used- do not install

( ) U33 4042 (socket installed in Step 2)

( ) U37 4042 (socket installed in Step 2)

( ) U38 74LS367

( ) U39 74LS367

( ) U40 74LS367

( ) U41 74LS367

( ) U42 74LS367

( ) U43 74LS367

( ) U44 74LS367

( ) U45 74LS367

( ) U47 74S472 Monitor - odd addressed bytes (socket installed in Step 2)

( )	U49	74S472 Monitor - even addressed bytes (socket installed in Step 2)
( )	U52	74LS367
( )	U53	74LS367
( )	U54	74LS367
( )	U55	74LS367
( )	U56	74LS367
( )	U57	74LS367
( )	U58	74LS32
( )	U59	74LS74

Warning: The CPU, U23, should not be installed at this time. It will be installed later after the integrity of the board has been verified.

STEP 10 - Install Expansion Memory I.C.'s

If you purchased any of the memory expansion capability, install those I.C 's and their sockets. The available expansion areas are:

( ) RAM 1

- ( ) U21 4042
- ( ) U36 4042
- ( ) U17 4042
- ( ) U32 4042

( ) RAM 2

- ( ) U20 4042
- ( ) U35 4042
- ( ) U16 4042
- ( ) U31 4042

( ) RAM 3

- ( ) U19 4042
- ( ) U34 4042
- ( ) U15 4042
- ( ) U30 4042

( ) PROM 1 - Usually reserved for "Instant Input Assembler"

( ) U46 74LS472 - odd addressed bytes

( ) U48 74LS472 - even addressed bytes

NOTE: Refer to our literature for programs that we offer in fusable link PROM. All of them are designed to run in this expansion area. Many of them, like the Instant Input Assembler, will speed up your programming tasks.

( ) EPROM

( ) U51 2708 - odd addressed bytes

( ) U50 2708 - even addressed bytes

NOTE: These two EPROMs can be programmed by the Super Starter Kit itself. Just put a blank EPROM in each socket, and then use the monitor to save your program in EPROM. Refer to the monitor section for detailed instructions.

$TEP 11 - Memory Configuration

Install the jumper wires to select the proper memory addressing. The Super Starter Kit allows you to rearrange the memory addressing allocation. The only restriction involves the PROM monitor. If you are using the PROM monitor, then the PROM monitor must be located at #FC00 and RAM must be located at #0000. If you are not planning to use the monitor, or your application requires a special address allocation, then refer to the schematic and determine for yourself what jumper configuration is required. If you want to use the standard kit configuration, then install the jumpers as follows:

( ) JW1 (in) to JW1 (out)

( ) JW2 (in) to JW2 (out)

( ) JW3 (in) to JW3 (out)

( ) JW4 (in) to JW4 (out)

( ) JW5 (in) to JW5 (out)

( ) JW6 (in) to JW6 (out)

( ) JW7 (in) to JW7 (out)

( ) JW8 (in) to JW8 (out)

( ) JW10 (in) to JW10 (out)

( ) JW11 (in) to JW11 (out)

( ) JW12 (in) to JW12 (out)

STEP 12 - Input/Output Configuration

The Super Starter Kit includes 32 bits of I/O (16 bits in and 16 bits out). The monitor uses three of these bits. We recognize that many users may not want to use the monitor, and have made provisions for removing the monitor related I/O. If you are using the monitor, install the following jumpers. If not, simply leave them out, and the monitor I/O is disabled. JW13 connects the clock to I/O bit two. It may be removed if you do not plan to use the real time clock.

( ) JW9 (in) to JW9 (out)

( ) JW13 (in) to JW13 (out)

( ) JW14 (in) to JW14 (out)

( ) JW15 (in) to JW15 (out)

STEP 13 - Install Control Switches

There are two different types of switches supplied with the kit, namely SPST and momentary contact switches. The first step is to separate them from each other. The momentary contact switch is the one which does not "latch". That is, if you move its handle it will spring back when it is released. The two SPST switches can be installed in either direction, but the momentary contact one must be properly oriented. The handle of the switch must face the processor. It is very important that the switch be installed correctly or the processor will be continually halted!

( ) SW1 - SPST

( ) SW2 - momentary contact

( ) SW3 - SPST

STEP 14 - Short Test

The board has been assembled, and before applying power, you can test for short circuits which might seriously damage the kit. Using a volt-ohmmeter or a continuity checker, check the resistance between the pins of the processor as described below. Each reading should indicate a high resistance or a very dim glow of the light. If any of them show a zero resistance or a bright lit light, then you have a short. If you find a short, you must recheck all of your connections until the problem is located and repaired. If power is applied to the kit in the presence of a short circuit, all of the I.C.'s may be damaged!

( ) Pin 1 (-5V) and Pin 26 (GND)

( ) Pin 1 (-5V) and Pin 2 (+5V)

( ) Pin 1 (-5V) and Pin 27 (+12V)

( ) Pin 2 (+5V)* and Pin 26 (GND)

( ) Pin 2 (+5V) and Pin 27 (+12V)

( ) Pin 27 (+12V) and Pin 26 (GND)

STEP 15 - Connection of Power

If the kit has no power shorts, then you are ready to apply the power. If you have located any shorts, DO NOT apply power. All of the Super Starter power is obtained via the 16-pin jacks. All of the pins of Jack J3 should be connected to the power supply ground. Jack J5 should be connected as follows:

( ) Pins 1-4, and 13-16 +5V

( ) Pins 5,6,11,12 +12V

( ) Pins 7,10 -5V

( ) Pins 8,9 +28V (optional - used only for EPROM programming)

After you have connected jacks J3 and J5 to the power supply, check to be sure that the jacks are inserted properly (pin 1 to pin 1) and that you have the proper supply input on each pin. An error here is very costly - it will ruin the entire kit!

STEP 16 - Power Check

As a further precaution before applying continuous power, we suggest that you perform the following power supply check. Place your volt-ohmmeter on the pins shown below, turn power on and then immediately turn power back off again. While power is on, check the reading and verify that it is correct. If it is not correct, then you have a construction error or you have not connected the power correctly. Correct the problem before proceeding with final checkout.

( ) +5V between Pin 16 of U8 and Pin 8 of U8

( ) -5V between Pin 1 of U23 and Pin 8 of U8

( ) +12 V between Pin 27 of U23 and Pin 8 of U8

STEP 17 - Install Processor

Turn off power and install the TMS 9900 CPU. The socket was installed earlier. Be certain to properly orient the CPU. Pin 1 should be in the corner nearest the toggle switches.

STEP 18 - Connection of Terminal

Turn off the power. Connect your terminal to input jack J10. If you have an RS-232C terminal, the pins on its connector should be connected as follows:

( ) Pin 1 (terminal) to Pin 1 of J10

( ) Pin 2 to Pin 2 of J10

( ) Pin 3 to Pin 3 of J10

( ) Pin 5 to Pin 4 of J10

( ) Pin 6 to Pin 5 of J10

( ) Pin 8 to Pin 7 of J10

( ) Pin 7 to Pin 6 of J10

If you have a TTY or other 20ma current loop interface, connect it as follows (there are no standard connector assignments - refer to the manual for your terminal):

( ) TTY IN - input to Pin 11 of J10 and return on Pin 10 of J10

( ) TTY OUT - input on Pin 8 of J10 and return on Pin 9 of J10

Apply power to the kit. If you are using a TTY, and it begins to "chatter" then reverse the output leads (Pin 8 and Pin 9).

STEP 19 - Start Monitor

To activate the monitor, reset the CPU and then type the letter 'X' on the terminal. The CPU should respond with a "?". If you cannot get the "?", you have an assembly error. First, check the small things:

( ) SW1 set correctly? (in LOAD position)

( ) SW2 oriented correctly? (handle toward CPU)

( ) Terminal wired correctly?

( ) All I.C.'s in right position?

( ) Monitor PROM in proper socket? (If they are reversed the monitor won't work.)

( ) All jumpers properly installed?

If the monitor responds with "?", then the kit is running. Try using the monitor and exploring the capabilities of your new computer. If you have further trouble and cannot get the kit running, contact the dealer that you purchased the kit from and ask for his assistance. If he cannot help you, call us at Rosse Corporation, and we will do everything we can to help you. Our number is (703) 471-7530.

IX. INSTRUCTION SET

The following notation is used to describe the TI 9900 instruction set. For further information regarding addressing modes, timing, etc. refer to the TMS 9900 Microprocessor Data Manual, which is found in section XIII.

S - General address for the source operand. Any addressing mode is acceptable. (See Figure IX-1)

D - General address for the destination operand. Any addressing mode is acceptable. (See Figure IX-1)

IOP - Immediate operand

W - Workspace register

DISP - Relative displacement

WP - Workspace pointer

PC - Program counter

ST - Status Register (See Figure IX-2)

() - Contents of address or register

→ - Replaces

FIGURE IX-1 ADDRESSING MODES

Addressing Mode	Description
Workspace Register	The contents of the indicated workspace register are the operands. (e.g. R3,R7)
Workspace Register Indirect	The contents of the indicated workspace register contain the memory address of the operand. (e.g. *R3,*R6)
Indexed	The contents of the indicated workspace register (R0 is not allowed as an index register) are added to the address enclosed in the second command word. (e.g. @2(R1),@6(R4))
Direct	The word following the instruction contains the memory address of the operand. (e.g. @6,@9)
Workspace Register Indirect with Auto Increment	The contents of the indicated workspace register contain the memory address of the operand which is automatically incremented after the access (plus 2 for word operations and plus 1 for byte operations). (e.g. *R1+,*R9+)

FIGURE IX-1 ADDRESSING MODES (continued)

Addressing Mode	Description
Immediate	The word following the instruction contains the operand.
Relative	The 8-bit displacement of the instruction is added to the updated program counter in jump instructions or to the base address in single-bit CRU instructions.

FIGURE IX-2 STATUS REGISTER

Bit	Description
0-LGT	logical greater than
1-AGT	arithmetic greater than
2-EQ	equal
3-C	carry
4-OV	overflow
5-P	odd parity
12-15	interrupt

INSTRUCTION: ADD

Format: A S,D

Opcode: A000

Status Changed: LGT,AGT,EQ,C,OV

Definition: The source operand is added to the destination operand. The sum replaces the destination operand.

Results: (S)+(D) → (D)

Notes: Use to add 16 bit numbers from:

Memory to Memory	A @SCALE,@TABLE
Register to Register	A R10,R9
Memory to Register	A @PRIME,R6
Register to Memory	A R14,@SUM

INSTRUCTION: ADD BYTES

Format: AB S,D

Opcode: B000

Status Changed: LGT,AGT,EQ,C,OV,OP

Definition: Add two 8-bit bytes. The 8-bit source operand is added to the 8-bit destination operand. If either address is a workspace register, then the left-most eight bits of that workspace register will be used.

Results: (S)+(D)→(D)

Notes: Used to add signed 8-bit numbers from:

Memory to Memory	AB	@X,@Y
Register to Memory	AB	R1,@Y
Memory to Register	AB	@X,R1
Register to Register	AB	R1,R2

INSTRUCTION: ABSOLUTE VALUE

Format: ABS S

Opcode: 0740

Status Changed: LGT,AGT,EQ,C,OV

Definition: Compute the absolute value of the source operand and replace the source operand with that result.

Results: Absolute value of (S) → (S)

Notes: Used to compute the absolute value of a 16-bit number.

```
ABS @LISTA
ABS @LISTB
```

	BEFORE	AFTER
LISTA	FFF4	000C
LISTB	000C	000C

INSTRUCTION: ADD IMMEDIATE

Format: AI W,IOP

Opcode: 0220

Status Changed: LGT,AGT,EQ,C,OV

Definition: Add the immediate value to the spacified workspace register.

Results: (W) + IOP → (W)

Notes: Add a constant to a workspace register.

AI R4,100	Add 100 to register R4
AI R11,10	Add ten to register R11

INSTRUCTION: AND IMMEDIATE

Format: ANDI W,IOP

Opcode: 0240

Status Changed: LGT,AGT,EQ

Definition: Perform a bit-by-bit logical AND operation between the workspace register and the immediate operand. Place the result in the workspace register.

Results: (W) AND IOP → (W)

Notes: Use to isolate certain bits of a workspace register.

```
ANDI 6,>F00E

Before: (R6)=>9877      1001 1000 0111 0111
Immed. operand=>F00E    1111 0000 0000 1110
After: (R6)=>9006       1001 0000 0000 0110
```

INSTRUCTION: UNCONDITIONAL BRANCH

Format: B S

Opcode: 0440

Status Changed: None

Definition: Replace PC with the source address. Effectively, transfers control to the source address.

Results: S → (PC)

Notes: This is the most flexible jump and can be used to transfer control to any location in memory. If the jump is out of range (+127, -128 words) for a relative jump instruction, use B.

Example: B @107 will cause PC to be reloaded with 107.

INSTRUCTION: BRANCH AND LINK TO SUBROUTINE

Format: BL S

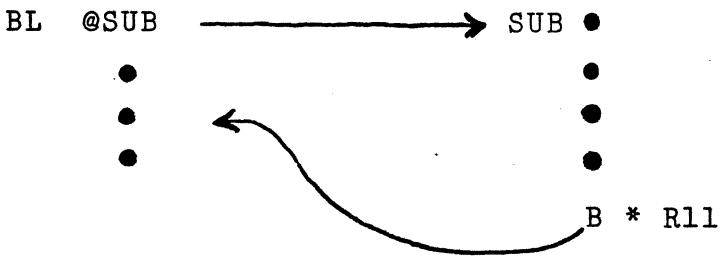
Opcode: 0680

Status Changed: None

Definition: Place source address in PC and place address of the instruction following the BL instruction in workspace register R11.

Results: (PC) → (R11)
S → (PC)

Notes: Use to transfer control to a subroutine. Return from the subroutine is accomplished with a branch indirect through register 11.



INSTRUCTION: BRANCH AND LOAD WORKSPACE POINTER

Format: BLWP S

Opcode: 0400

Status Changed: None

Definition: Place source operand into WP and the word following it into the PC. Place previous contents of WP into R13 of the new workspace, PC (address immediately following BLWP) in new R14 and ST in new R15.

Results: (S) → (WP)
(S+2) → (PC)
(original WP) → (R13)
(original PC) → (R14)
(original ST) → (R15)

Notes: Use to call a subroutine and change the workspace environment. The subroutine must return via RTWP command.

```
BLWP R4        Place (R4) in WP,(R5) in PC
BLWP @SBR      Place (SBR) in WP,(SBR+2) in PC
```

INSTRUCTION: COMPARE

Format: C S,D

Opcode: 8000

Status Changed: LGT,AGT,EQ

Definition: Compare the contents of the source operand with the contents of the destination operand and set/reset designated status register bits.

Results: Status register bits set/reset after comparison.

Notes: Use to compare 16-bit numbers from:

Memory to Memory	C @TOP,@LAST
Register to Register	C R1,R6
Memory to Register	C @BOT,R5
Register to Memory	C R7,@11

INSTRUCTION: COMPARE BYTES

Format: CB S,D

Opcode: 9000

Status Changed: LGT,AGT,EQ,OP

Definition: Compare the contents of the source operand byte with the contents of the destination operand byte and set/reset the designated status register bits.

Results: Status Register bits set/reset after comparison.

Notes: Use to compare 8-bit numbers. If a workspace register is used for S or D, the left-most 8-bits will be used.

CB R1,R2 Compare R1 (byte) with R2 (byte).

INSTRUCTION: COMPARE IMMEDIATE

Format: CI W,IOP

Opcode: 0280

Status Changed: LGT,AGT,EQ

Defintion: Compare the contents of the specified register with the immediate operand and set/reset designated status register bits.

Results: Status register bits set/reset after comparison.

Notes: Use to compare contents of workspace register with some known value and set status register bits accordingly.

CI R2,>7FFF	Compare register R2 to >7FFF
CI R3,0	Compare register R3 to zero. (A more efficient way is: MOV R3,R3)

INSTRUCTION: CLEAR

Format: CLR S

Opcode: 04C0

Status Changed: None

Definition: Replace source operand with a full 16-bit word of zeroes.

Results: (S) ←0

Notes: Use to zero workspace registers or memory locations.

```
        CLR R5              Clear register R5
        CLR @SUM            Clear location SUM
        LI  R1,X            Clear (X) to (X+10)
LOOP    CLR *R1+
        CI  R1,X+12
        JL  LOOP
```

INSTRUCTION: COMPARE ONES CORRESPONDING

Format: COC S,W

Opcode: 2000

Status Changed: EQ

Definition: When all ones in the source operand have a corresponding one in the destination workspace register, set the equal bit in the status register.

Results: EQ status bit is set/reset.

Notes: Use to check if a bit or bits in a destination workspace register are set to one. Bits correspond to the one bits in the source operand. If corresponding bits in destination are also set, the equal bit in Status Register is also set.

Assume TEST= C102 = 1100 0001 0000 0010

R8 = E306 = 1110 0011 0000 0110

Then COC @TEST,R8

Every logic one bit in TEST has a corresponding logic one bit in reg. R8; therefore the equal status bit is set

```
MASK    DATA  8000
        COC   @MASK,R1   IS SIGN IN R1 A ONE?
        JEQ   ADD        IF SO, JUMP TO ADD
```

INSTRUCTION: COMPARE ZEROES CORRESPONDING

Format: CZC S,W

Opcode: 2400

Status Changed: EQ

Definition: When the bits in the destination workspace register corresponding to the one bits in the source operand are all equal to a logic zero, set equal status bit.

Results: Set/reset status register equal bits.

Notes: Use to test single/multiple bits within a workspace register.

Assume TEST=>C102 = 1100 0001 0000 0010

R8 =>2201 = 0010 0010 0000 0001

Then CZC @TEST,R8

Every logic one bit in TEST has a corresponding logic zero in register R8; therefore, the equal status bit is set.

INSTRUCTION: DECREMENT BY ONE

Format: DEC S

Opcode: 0600

Status Changed: LGT,AGT,EQ,C,OV

Definition: Subtract one from the 16-bit source operand.

Results: (S)-1 → (S)

Notes: Used for indexing or controlling loops.

```
DEC   @TEC          TEC=TEC-1
JNE   LOOP          JUMP IF TEC NOT ZERO
```

INSTRUCTION: DECREMENT BY TWO

Format: DECT S

Opcode: 0640

Status Changed: LGT,AGT,EQ,C,OV

Definition: Subtract two from the 16-bit source operand.

Results: (S)-2 → (S)

Notes: Useful for counting and indexing full word arrays.

```
DECT   @COUNT     Subtract two from COUNT
DECT   R10        Subtract two from register 10
```

INSTRUCTION: DIVIDE

Format: DIV S,W

Opcode: 3C00

Status Changed: OV

Definition: Divide the destination operand (a 32-bit unsigned integer) by the source operand (a 16-bit unsigned integer) using integer arithmetic and place the quotient in the destination operand and the remainder in the second word of the destination operand. If the quotient exceeds 16-bits, the overflow is set.

Results: (W,W+1) / (S) → (W) quotient
(W+1) remainder

Notes: Use divide (DIV) for integer division (unsigned).

```
DIV   R3,R4     Divide registers R4,R5 by register (R3)
DIV   @SUM,2    Divide registers R2,R3 by (SUM)
```

INSTRUCTION: IDLE COMPUTER

Format: IDLE

Opcode: 0340

Status Changed: None

Definition: Place the computer in an IDLE state.

Results: Computer is IDLE.

Notes: Used to halt the processor and wait for an interrupt.

INSTRUCTION: INCREMENT BY ONE

Format: INC S

Opcode: 0580

Status Changed: LGT,AGT,EQ,C,OV

Definition: Add one to the 16-bit source operand.

Results: (S)+1 → (S)

Notes: INC @CNT(R1) increment table location selected by R1

INSTRUCTION: INCREMENT BY TWO

Format: INCT S

Opcode: 05C0

Status Changed: LGT,AGT,EQ,C,OV

Definition: Add two to the 16-bit source operand.

Results: (S)+2 → (S)

Notes: Useful for controlling word addressing of an index.

INSTRUCTION: INVERT

Format: INV S

Opcode: 0540

Status Changed: LGT,AGT,EQ

Definition: The 16-bit source operand is replaced with its one's complement.

Results: One's complement of (S) → (S)

Notes: Use this operation to "flip" the bits in some memory location or register.

```
INV  R2     Invert register R2
INV  @SUM   Invert location (SUM)
INV  *R3    Invert location in register R3
```

INSTRUCTION: JUMP EQUAL

Format: JEQ DISP

Opcode: 1300

Status Changed: None

Definition: When the equal status bit is set, the signed displacement is added to the PC.

Results: (PC) + (displacement) → PC (if EQ)

(PC) + 2 → PC (if not EQ)

Notes: Used to transfer if equal

```
C     @X,@Y
JEQ   YES     go to YES if (X) = (Y)
```

INSTRUCTION: JUMP IF GREATER THAN

Format: JGT DISP

Opcode: 1500

Status Changed: None

Definition: When the arithmetic greater than status bit is set, add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (if AGT)
(PC) + 2 → (PC) (if AGT .clear)

Notes: Used following a 16-bit arithmetic operation:

```
C     @ONE,@TWO
JGT   @OUI          go to OUI if (ONE) is
                    arithmetically greater
                    than (TWO)
```

The arithmetic greater than is the result of a signed compare, so >FFFF (-1) is <u>not</u> arithmetic greater than >7FFF, but it is logical greater than.

INSTRUCTION: JUMP ON HIGH

Format: JH DISP

Opcode: 1B00

Status Changed: None

Definition: When the logical greater than status bit is set and the equal status bit is clear then the signed dispalcement is added to the PC.

Results: (PC) + Displacement → (PC) (if LGT and not EQ)
(PC) + 2 → (PC) (if LGT clear or EQ)

Notes: Used when comparing logical or unsigned values.

```
C     @BIG,@GOOD
JH    @BAD          go to BAD if (BIG) is
                    logically greater than
                    (GOOD) (unsigned)
```

Since the logical greater than is an unsigned compare, this instruction is most often used for address comparisons.

INSTRUCTION: JUMP ON HIGH OR EQUAL

Format: JHE DISP

Opcode: 1400

Status Changed: None

Definition: When the equal status bit or the logical greater than status bit is set, the signed displacement is added to the PC.

Results: (PC) + Displacement → (PC) (if LGT or EQ)
(PC) + 2 → (PC) (if LGT clear and EQ clear)

Notes: Use to branch or transfer control when either logical greater than or equal status bits=1.

```
JHE     $+4     If SR bits 0 or 2 =1, skip
                one word.
JHE     SUB     If SR bits 0 or 2 =1, jump
                to SUB.
```

INSTRUCTION: JUMP ON LOW

Format: JL DISP

Opcode: 1A00

Status Changed: None

Definition: When the logical greater than and equal stauts bits are both reset, then th signed displacement is added to the PC.

Results: (PC) + Displacement → (PC) (If LGT and EQ are clear)

(PC) + 2 → (PC) (If LGT or EQ)

Notes: Use to transfer control when a logical or unsigned less than condition is detected.

```
C   @ONE,@TWO
JL  @GO          go to GO if (ONE) logically
                 less than (TWO) (unsigned
                 compare)
```

INSTRUCTION: JUMP ON LOW OR EQUAL

Format: JLE DISP

Opcode: 1200

Status Changed: None

Definition: When the equal status bit is set or the logical greater than is reset, then the signed dispalcement is added to the PC.

Results: (PC) + Displacement → (PC) (if LGT clear or EQ set)
(PC) + 2 → (PC) (if LGT set and EQ clear)

Notes: Use to test status register bits and transfer control if LGT=0 or EQ=1.

```
JLE     ADDNO     If SR bits 0=0 or 2=1,
                  go to ADDNO
```

INSTRUCTION: JUMP ON LESS THAN

Format: JLT DISP

Opcode: 1100

Status Changed: None

Definition: If the arithmetic greater than and equal status bits are reset then add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (If LGT and EQ reset)
(PC) + 2 → (PC) (If LGT or EQ set)

Notes: Used when comparing arithmetic values.

```
C     @A,@B
JLT   LESS        go to LESS if (A) is
                  arithmetically less than (B)
```

INSTRUCTION: UNCONDITIONAL JUMP

Format: JMP DISP

Opcode: 1000

Status Changed: None

Definition: Add the signed displacement to the PC and place the sum into the PC.

Results: (PC) + Displacement → PC

Notes: Use to transfer control unconditionally.

```
        JMP   LOOP   Begin execution at loop
        JMP   $      Remain at this location
HERE    JMP   $+4    Remain at this location
        JMP   $+4    Jump over next address
```

The destination address must be within the range +127 to -128 words. If not, use the branch (B) instruction.

INSTRUCTION: JUMP ON NO CARRY

Format: JNC DISP

Opcode: 1700

Status Changed: None

Definition: If the carry status bit is clear, add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (If no carry)
(PC) + 2 → (PC) (If carry)

Notes: Use to branch when carry cleared.

```
    JNC   YES   If carry clear, go to YES
```

Can be used to check for 16-bit carry for multi-precision arithmetic. The following will calculate (R1,R2) + (R3,R4).

```
     A     R4,R2
    JNC    GO
    INC    R1
  GO A     R3,R1
```

INSTRUCTION: JUMP ON NOT EQUAL

Format: JNE DISP

Opcode: 1600

Status Changed: None

Definition: If the equal status bit is reset, add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (If not EQ)
(PC) + 2 → (PC) (If EQ).

Notes: Used to branch when not equal.

```
A     R1,R2
JNE   X          go to X if R1 + R2 not zero
MOV   R1,R1
JNE   NO         go to NO if R1 not zero
```

INSTRUCTION: JUMP ON NO OVERFLOW

Format: JNO DISP

Opcode: 1900

Status Changed: None

Definition: When the overflow status bit is reset, add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (If no OV)
(PC) + 2 → (PC) (If OV)

Notes: Used to test arithmetic overflow.

```
    A    R1,R2
   JNO   GOOD        go to GOOD if R1+R2 does
                     not overflow
```

An overflow occurs during an add if the sign of the two operands are the same but the sign of the sum is not the same.

INSTRUCTION: JUMP ON CARRY

Format: JOC DISP

Opcode: 1800

Status Changed: None

Definition: When the carry status bit is set, add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (if carry)
(PC) + 2 → (PC) (if no carry)

Notes: Use to branch or transfer control if carry is set.

```
JOC START      If Carry, Go to Start
JOC $-2        If Carry, Go to Previous Instruction
```

INSTRUCTION: JUMP ON ODD PARITY

Format: JOP DISP

Opcode: 1C00

Status Changed: None

Definition: When the odd parity status bit is set, add the signed displacement to the PC.

Results: (PC) + Displacement → (PC) (If OP)
(PC) + 2 → (PC) (If not OP)

Notes: Used to test parity of 8-bit values.

```
MOVB    @CH,R1
JOP     ODD         go to ODD if CH is
                    odd parity
```

Note that the OP flag is only changed by byte instructions (e.g. MOVB,CB)

INSTRUCTION: LOAD COMMUNICATIONS REGISTER UNIT (OUTPUT)

Format: LDCR S,C

Opcode: 3000

Status Changed: LGT,AGT,EQ,OP (IF C <9)

Definition: Transfer the number of bits specified (C) from the source operand to consecutive CRU lines. The contents of R12 determines the least significant CRU line.

Results: (S) → CRU for C bits

Notes: Use this to output a bit pattern to CRU lines for versatile I/O. If number of bits specified is less than nine, then S is a byte address. If number of bits is nine or more, S becomes a word address. The least significant memory bit goes to the least significant memory CRU bit. If the bit count (C) is zero, then 16 bits are output. Prior to an LDCR instruction, register R12 (CRU Base Address) must be loaded with the appropriate address. With this kit, R12=0 will address bit 0.

```
LDCR   2,0      Transfer 16 bits to CRU from R2
LDCR   @NUM,8   Transfer 8 bits to CRU from NUM
```

INSTRUCTION: LOAD IMMEDIATE

Format: LI W,IOP

Opcode: 0200

Status Changed: LGT,AGT,EQ

Definition: Place the immediate operand in the specified register.

Results: IOP → (W)

Notes: Use to initialize register for counters or addresses.

```
LI R5,TABLE     LOAD R5 WITH ADDRESS OF TABLE
LI R1,10        SET R1 TO 10
LI R2,1000      LOAD REGISTER R2 WITH 1000
```

INSTRUCTION: LOAD INTERRUPT MASK IMMEDIATE

Format: LIMI IOP

Opcode: 0300

Status Changed: Interrupt Mask

Definition: Place the four least significant bits of IOP into the interrupt mask (bits 15-12 of the Status Register).

Results: IOP (15-12) → Status Register (15-12)

Notes: Used to enable or disable interrupts.

```
LIMI  0     disable all interrupts
LIMI >F     enable all interrupts
```

INSTRUCTION: LOAD WORKSPACE POINTER IMMEDIATE

Format: LWPI IOP

Opcode: 02E0

Status Changed: None

Definition: Replace contents of workspace pointer register with the beginning address of 16 contiguous words. This changes the current workspace pointer and environment.

Results: IOP → (WP)

Notes: Use to initialize the WP register to alter workspace environment.

```
LWPI  >100     Place >100 in workspace pointer
LWPI  WSP      Location WSP = Register 0
```

INSTRUCTION: MOVE WORDS

Format: MOV S,D

Opcode: C000

Status Changed: LGT,AGT,EQ

Definition: Replace destination operand with a copy of the source operand.

Results: (S) → (D)

Notes: Use to move from:

Memory to Memory	MOV@TABLE,@TEMP
Register to Register	MOV R5,R9
Register to Memory (Store)	MOV R3,@ANSWER
Memory to Register (Load)	MOV @TABLE,R8

INSTRUCTION: MOVE BYTES

Format: MOVB S,D

Opcode: D000

Status Changed: LGT,AGT,EQ,OP

Definition: Move the source byte operand to the destination byte operand. Whenever S or D is a workspace register, then the leftmost 8-bits are used.

Results: (S) → (D)

Notes:

Load Register	MOVB @X,R1
Store Register	MOVB R1,@Y
Move Memory to Memory	MOVB @X,@Y
Move Register to Register	MOVB R1,R2

INSTRUCTION: MULTIPLY

Format: MPY S,W

Opcode: 3800

Status Changed: None

Definition: Multiply the destination operand, an unsigned 16-bit integer by the source operand, an unsigned 16-bit integer. Place the product into the 32-bit (two word) destination field right justified.

Results: (W) * (S) → (W,W+1)

Notes: Use multiply (MPY) to multiply two 16-bit unsigned integers. The destination operand must be a workspace register, therefore the result will be in workspace register specified and the next one. If workspace register 15 is specified then the next memory location following the workspace area is the second half of the product.

```
MPY   *1,4     MPY reg R4 by reg R1 (indirect)
MPY   @NUM,4   MPY reg R4 by (NUM)
```

INSTRUCTION: NEGATE

Format: NEG S

Opcode: 0500

Status Changed: LGT,AGT,EQ,C,OV

Definition: Replace source operand with two's complement value of the source operand.

Results: 0-(S) → (S)

Notes: Use NEG to replace the operand with its additive inverse.

```
NEG  R7
```

The contents of workspace register R7 is replaced with its two's complement value.

INSTRUCTION: OR IMMEDIATE

Format: ORI W,IOP

Opcode: 0260

Status Changed: LGT,AGT,EQ

Definition: Perform a logical OR operation between the specified workspace register and the immediate operand. Place the result in the workspace register.

Results: (W) OR IOP → (W)

Notes: Use to perform logical OR between workspace register and some known immediate value.

Example: ORI R10,>202D

Before:	R10=>1AD5	0001 1010 1101 0101
Imed. Operand=		0010 0000 0010 1101
After:	R10=>3AFD	0011 1010 1111 1101

ORI R5,>8000 Set sign bit to one in R5

ORI R10,>F Set four LSB to one in R10

INSTRUCTION: RETURN WITH WORKSPACE POINTER

Format: RTWP

Opcode: 0380

Status Changed: All status bits set by R15, including interrupt mask.

Definition: Replace contents of WP with contents of current R13, PC with contents of R14, ST with currnet value of R15.

Results: (R13) → (WP)
(R14) → (PC)
(R15) → (ST)

Notes: Use to return from a BLWP,XOP or a hardware interrupt.

INSTRUCTION: SUBTRACT WORDS

Format: S S,D

Opcode: 6000

Status Changed: LGT,AGT,EQ,C,OV

Definition: Subtract the source operand from the destination operand and place the result in the destination operand.

Results: (D)-(S) → (D)

Notes: Use to subtract signed 16-bit integers from:

Memory to Memory	S @OLDVAL,@NEWVAL
Register to Register	S R8,R7
Register to Memory	S R10,@DIFF
Memory to Register	S @CONS,R14

INSTRUCTION: SUBTRACT BYTES

Format: SB S,D

Opcode: 7000

Status Changed: LGT,AGT,EQ,C,OV,OP

Definition: Subtract the source operand byte from the destination operand byte and place the difference in the destination operand byte.

Results: (D)-(S) → (D)

Notes: Use to subtract signed integer bytes.

SB @>501,@>503	Result in address>503
SB R1,R2	Result in upper byte of R2

INSTRUCTION: SET BIT ONE

Format: SBO DISP

Opcode: 1D00

Status Changed: None

Definition: Set the output bit to a logic one. The bit address is computed by adding bits 3-14 of R12 to the signed dispalcement.

Results: 1 → (CRU bit specified by bits 3-14 of R12 + displacement)

Notes: Use to set a particular CRU line to a logical one.

```
CLR   R12      ; Set CRU base
SBO   5        ; Set bit 5
```

The following sequence is equivalent:

```
LI    R12,30   ; Set CRU Base
SBO   -10      ; SEt bit 5
```

Bit 5 is specified because bits 3-14 of R12 is 15(R12/2) and 15+(-10) is 5.

INSTRUCTION: SET BIT ZERO

Format: SBZ DISP

Opcode: 1E00

Status Changed: None

Definition: Set output CRU bit to a logical zero. The CRU bit is determined by adding contents of bits 3-14 of R12 to the signed displacement.

Results: 0 → (CRU bit specified by bits 3-14 of R12 + displacement)

Notes: Use to get the particular CRU line to a logical zero.

```
LI    12,>280    CRU base address=>140 (R12/2)
SBZ   >28        Sets CRU address >168 (140+28)
                 to zero
SBZ   -2         Sets CRU address >13E (140-2)
                 to zero
```

INSTRUCTION: SET TO ONES

Format: SETO S

Opcode: 0700

Status Changed: None

Definition: Replace the source operand with a 16-bit word of one's.

Results: (S) ← FFFF

Notes: Use to initialize a table with -1 values instead of zeroes if your application requires such. Use to initialize refister with -1.

```
SETO  5          Set register 5 to >FFFF
SETO  @SUM       Set SUM to -1
```

INSTRUCTION: SHIFT LEFT ARITHMETIC

Format: SLA W,C

Opcode: 0A00

Status Changed: LGT,AGT,EQ,C,OV

Definition: The contents of the workspace register are shifted left the specified number of bits (C) with zeroes filling the vacated bit positions. The last bit shifted out is placed in the carry out bit. If C=0, the right four bits of register R0 are used as the shift count.

Results: (W) is shifted left the specified shift count (C).

Notes: Use to shift the contents of a workspace register left by some shift count.

```
SLA  R4,8     Shift reg R4 left 8 places
SLA  R4,2     Effectively multiply reg R4 by 4
SLA  R4,0     Shift reg R4 by contents of R0
```

Note that SLA R4,0 will shift R4 by the contents of the lower four bits of R0. If R0=17, the shift count is one because 17=10001 (binary).

INSTRUCTION: SET ONES CORRESPONDING (LOGICAL OR)

Format: SOC S,D

Opcode: E000

Status Changed: LGT,AGT,EQ

Definition: Set to logic one the bits in the destination operand that correspond to any logic one value in the source operand. This result is placed in the destination. This is effectively a logical OR operation.

Results: (S) OR (D) → (D)

Notes: Use to perform a logical OR operation. This is similar to ORI except it may be done between two general addresses.

```
Before: (PATRN1)=>E06B=1110 0000 0110 1101
        (PATRN2)=>4482=0100 0100 1000 0010

        SOC  @PATRN1,@PATRN2

After:  (PATRN1)=>E06B
        (PATRN2)=>E4EF=1110 0100 1110 1111
```

INSTRUCTION: SET ONES CORRESPONDING BYTE (LOGICAL OR)

Format: SOCB S,D

Opcode: F000

Status Changed: LGT,AGT,EQ,C

Definition: Set to a logical one the bits in the destination operand byte that correspond to any logical one in the source operand byte. This is effectively an 8-bit logical OR operation.

Results: (S) OR (D) → (D)

Notes: Use to perform an 8-bit OR.

```
SOCB  R1,@X     (X)=(X) OR R1
```

INSTRUCTION: SHIFT RIGHT ARITHMETIC

Format: SRA W,C

Opcode: 0800

Status Changed: LGT,AGT,EQ,C

Definition: Shift the contents of the specified workspace register right by the number of places specified by C. The sign bit is extended to fill the vacated bits. If C=0, then the right four bits of workspace register R0 are used for the shift count. The last bit shifted out is placed in the carry bit of the status register.

Results: (W) shifted right C places → (W)

Notes: Use to shift to the right a signed integer.

SRA R14,5

Shift right the contents of R14 by 5 places. This is a divide by 32.

INSTRUCTION: SHIFT RIGHT CIRCULAR

Format: SRC W,C

Opcode: 0B00

Status Changed: LGT,AGT,EQ,C

Definition: Shift the specified workspace register right by the specified number of places (C), with the bits being shifted out of bit 15 placed in bit 0. If C=0, the right four bits of register R0 are used as the shift count.

Results: (W) shifted right circular C places → (W).

Notes: Shift right circular some specified workspace register.

SRC R9,R5

INSTRUCTION: SHIFT RIGHT LOGICAL

Format: SRL W,C

Opcode: 0900

Status Changed: LGT,AGT,EQ,C

Definition: Shift the specified work register to the right the specified shift count filling the vacated bits with zeroes. The last bit shifted out is placed in the carry out bit. If C=0, the right four bits of register 0 are used as the shift count.

Results: (W) shifted right C places → (W)

Notes: Use to shift a workspace register right logical.

```
SRL  R10,5     Shift reg R10 right 5 places
SRL  R9,1      Effectively divide reg 9
               by 2 (unsigned)
```

INSTRUCTION: STORE COMMUNICATION REGISTER UNIT (INPUT)

Format: STCR S,C

Opcode: 3400

Status Changed: LGT,AGT,EQ,OP((<9)

Definition: Transfer number of bits specified (C) from the CRU lines addressed by R12 to the source operand. If the number of bits does not fill an entire memory word, then zeroes are added on the left. If C <9, then S is a byte address. If C >9 then S is a word address.

Results: CRU lines → (S) for C bits

Notes: Use to store contents of CRU lines in some memory location. Least significant CRU line to least significant memory bit.

If C <9 byte addressing

C ⩾9 word addressing

INSTRUCTION: STORE STATUS REGISTER

Format: STST W

Opcode: 02C0

Status Changed: None

Definition: Transfer the status register to workspace register W.

Results: Status Register → (W)

Notes: Used to transfer the status register to workspace so it can be manipulated.

```
STST  R5      R5=status
```

INSTRUCTION: STORE WORKSPACE POINTER

Format: STWP W

Opcode: 02A0

Status Changed: None

Definition: Transfer the workspace pointer to workspace register W.

Results: WP → (W)

Notes: Used to determine the address of the register file.

```
STWP  R6      R6 = address of R0
```

After execution of the above instruction, the following two instructions are the same.

```
INC     R0
INC     *R6
```

INSTRUCTION: SWAP BYTES

Format: SWPB S

Opcode: 06C0

Status Changed: None

Definition: Swap the upper byte of the source operand with the lower byte of the source operand.

Results: Swap (S) upper and (S) lower.

Notes: Used for character manipulation.

```
MOVB  @C1,R1   R1=character one
SWPB  R1       reverse bytes
MOVB  @C2,R1   R1=character two,one
```

INSTRUCTION: SET ZEROES CORRESPONDING

Format: SZC S,D

Opcode: 4000

Status Changed: LGT,AGT,EQ

Definition: Set to a logic zero the bits in the destination operand that correspond to bit positions equal to logic one in the source operand. The source is not changed. Effectively this is a logical AND with the source being inverted prior to the AND.

Results: NOT (S) AND D → D

Notes: Use to turn off flag bits or AND the contents of one's complement source and destination.

Before: (PAT1)= >3030=0011 0000 0011 0000
(PAT2)= >5511=0101 0101 0001 0001

```
SZC  @PAT1,@PAT2
```

After: (PAT1)= >3030
(PAT2)= >4501=0100 0101 0000 0001

INSTRUCTION: SET ZEROES CORRESPONDING (BYTE)

Format: SZCB S,D

Opcode: 5000

Status Changed: LGT,AGT,EQ,OP

Definition: Set to a logical zero the bits in the destination operand byte that correspond to bit positions equal to a logical one in the source byte.

Results: NOT (S) AND (D) $\rightarrow$ (D)

Notes: Useful for character or flag manipulation.

SZCB @X,@Y $Y=\overline{X}$ AND Y

INSTRUCTION: TEST BIT

Format: TB DISP

Opcode: 1F00

Status Changed: EQ

Definition: Read the specified input bit whose address is computed by adding the signed displacement to bits 3-14 of R12. Set the equal status register bit to the value read.

Results: EQ ← CRU line read

Notes: Use to read a particular CRU line and depending on the result, make appropriate decisions.

```
CLR  R12     set CRU base
TB   14      wait for bit 14 to be set
JNE  $-2
```

INSTRUCTION: EXECUTE

Format: X S

Opcode: 0480

Status Changed: None (remote instruction may, however)

Definition: The instruction at the source operand is executed.

Results: Execute (S)

Notes: Used to execute an instruction out of line, typically in a table.

```
X   @TAB(R1)    execute the instruction in
                table TAB pointed to by R1
```

INSTRUCTION: EXTENDED OPERATION

Format: XOP S,N

Opcode: 2C00

Status Changed: None

Definition: Place extended operation into execution. The (N) field indicates which XOP trap location to utilize.

Results: S → (R11) of XOP workspace
(0040+4n) → (WP)
(0042+4n) → (PC)
(WP) → (R13) of XOP workspace
(PC) → (R14) of XOP workspace
(ST) → (R15) of XOP workspace

Notes: Use to implement software routines which are used frequently, for example: floating point arithmetic
signed multiply
extended precision

The monitor uses XOP 0 as a breakpoint call. That is, a breakpoint replaces the users instruction by an XOP 0. XOP 1 and XOP 2 are used for input and output. The following will print the letter "A".

```
LETTER BYTE  'A'
       XOP  @LETTER,2
```

INSTRUCTION: EXCLUSIVE OR

Format: XOR S,W

Opcode: 2800

Status Changed: LGT,AGT,EQ

Definition: Perform a bit by bit exclusive OR of the 16-bit source operand with the 16-bit destination workspace register.

Results: (S) XOR (W) → (W)

Notes: Use to perform an exclusive OR between a workspace register and a source operand.

Assume: (R0)=>21BD = 0010 0001 1011 1101

(TC)=>E436 = 1110 0100 0011 0110

Then: XOR @TC,0

(R0)=>C58B = 1100 0101 1000 1011

INSTRUCTION: EXTERNAL CONTROL

Format: CKOF (Clock Off)

CKON (Clock On)

LREX (Load Rom/Execute)

RSET (Reset)

Opcode: 03C0

03A0

03E0

0360

Definition: These instructions can be decoded by external hardware. The TI 9900 does not perform any function when they are executed. This kit does not decode these instructions, so they should be avoided.

INSTRUCTION PATCHING: It is frequently necessary to patch a program resident in RAM. The TI 9900's addressing often becomes confusing when trying to patch programs. To assist the user, the patching tables are provided. The first gives the hexadecimal op-code and the second provides the additional digits for addressing.

For example, if a MOV *R1,@5(R2) is needed, the following steps are used:

(1) op-code = Cxxx (from Table I)

(2) xxx = 89s (from Table II)

(3) Thus, instruction = C89s = C891

TABLE I: OP-CODES

A	Axxx	add Rs to Rd
AB	Bxxx	add Rs (byte) to Rd (byte)
AI	022s	add constant to Rs
ANDI	024s	AND Rs with Rd
C	8xxx	compare Rs with Rd
CB	9xxx	compare Rs (byte) to Rd (byte)
CI	028s	compare constant with Rs
CKOF	03C0	clock-off
CKON	03A0	clock-on
COC	2aaa	compare (Rd and Rs) with Rs
CZC	2bbb	compare (Rd and Rs) with zero
DIV	3ccc	Rd=(Rd,Rd+1)/Rs, Rd+1=remainder
IDLE	0340	idle
JEQ	13yy	jump if equal
JGT	15yy	jump if greater than
JH	1Byy	jump if high
JHE	14yy	jump if high or equal
JL	1Ayy	jump if low
JLE	12yy	jump if low or equal
JLT	11yy	jump if less than
JMP	10yy	jump unconditional
JNC	17yy	jump if carry clear
JNE	16yy	jump if not equal

TABLE I: OP-CODES (continued)

JNO	19yy	jump if no overflow
JOC	18yy	jump if carry set
JOP	1Cyy	jump if odd parity
LDCR	3aaa	d-bits of Rs to CRU
LI	020s	load Rs immediate
LIMI	0300	load interrupt mask immediate
LREX	03E0	load Rom and execute
LWPI	02E0	load workspace pointer immediate
MOV	Cxxx	move Rs to Rd
MOVB	Dxxx	move Rs (byte) to Rd (byte)
MPY	3ddd	(Rd,Rd+1)=Rd times Rs
ORI	026s	OR or constant with Rs
RSET	0360	reset
RTWP	0380	return with workspace
S	6xxx	subtract Rs from Rd
SB	7xxx	subtract Rs (byte) from Rd (byte)
SBO	1Dyy	set CRU bit yy
SBZ	1Eyy	set CRU bit yy
SLA	0Ans	shift Rs left (alg.) by n
SOC	Exxx	OR Rs with Rd
SOCB	Fxxx	OR Rs (byte) to Rd (byte)
SRA	08ns	shift Rs right (alg.) by n
SRC	0Bns	shift Rs right (circ.) by n
SRL	09ns	shift Rs right (log.) by n

TABLE I: OP-CODES (continued)

STCR	3bbb	d-bits of CRU to Rs
STST	02Cs	Rs = status register
STWP	02As	Rs = workspace pointer
SZC	4xxx	Rd = Rd and not Rs
SZCB	5xxx	Rd (byte) = Rd (byte) and not Rs
TB	1Fyy	test CRU bit
XOP	2ccc	extended operation
XOR	2ddd	ex-OR Rs with Rd

	Rs	*Rs	*Rs+	@Rs	
ABS	074s	077s	076s	076s	absolute value of Rs
B	----	045s	047s	046s	branch
BL	----	069s	06Bs	06As	branch and link R11
BLWP	----	041s	043s	042s	branch and link workspace
CLR	04Cs	04Ds	04Fs	04Es	clear Rs
DEC	060s	061s	063s	062s	decrement Rs by one
DECT	064s	065s	067s	066s	decrement Rs by two
INC	058s	059s	05Bs	05As	increment Rs by one
INCT	05Cs	05Ds	05Fs	05Es	increment Rs by two
INV	054s	055s	057s	056s	invert Rs (ones comp.)
NEG	050s	051s	053s	052s	negate Rs (twos comp.)
SETO	070s	071s	073s	072s	set Rs to ones
SWPB	06Cs	06Ds	06Fs	06Es	swap bytes of Rs
X	048s	049s	04Bs	04As	execute inst. at Rs

TABLE II: ADDRESSING

		R0	R1	R2	R3	R4	R5	R6	R7		
XXXX	Rs,Rd	00s	04s	08s	0Cs	10s	14s	18s	1Cs	Rs,Rd	aaaa
	*Rs,Rd	01s	05s	09s	0Ds	11s	15s	19s	1Ds	*Rs,Rd	
	*Rs+,Rd	03s	07s	0Bs	0Fs	13s	17s	1Bs	1Fs	*Rs+,Rd	
	@Rs,Rd	02s	06s	0As	0Es	12s	16s	1As	1Es	@Rs,Rd	
	Rs,*Rd	40s	44s	48s	4Cs	50s	54s	58s	5Cs	Rs,Rd	bbbb
	*Rs,*Rd	41s	45s	49s	4Ds	51s	55s	59s	5Ds	*Rs,Rd	
	*Rs+,Rd	43s	47s	4Bs	4Fs	53s	57s	5Bs	5Fs	*Rs+,Rd	
	@Rs,*Rd	42s	46s	4As	4Es	52s	56s	5As	5Es	@Rs,Rd	
	Rs,*Rd+	C0s	C4s	C8s	CCs	D0s	D4s	D8s	DCs	Rs,Rd	cccc
	*Rs,*Rd+	C1s	C5s	C9s	CDs	D1s	D5s	D9s	DDs	*Rs,Rd	
	*Rs+,*Rd+	C3s	C7s	CBs	CFs	D3s	D7s	DBs	DFs	*Rs+,Rd	
	@Rs,*Rd+	C2s	C6s	CAs	CEs	D2s	D6s	DAs	DEs	@Rs,Rd	
	Rs,@Rd	80s	84s	88s	8Cs	90s	94s	98s	9Cs	Rs,Rd	dddd
	*Rs,@Rd	81s	85s	89s	8Ds	91s	95s	99s	9Ds	*Rs,Rd	
	*Rs+,@Rd	83s	87s	8Bs	8Fs	93s	97s	9Bs	9Fs	*Rs+,Rd	
	@Rs,@Rd	82s	86s	8As	8Es	92s	96s	9As	9Es	@Rs,Rd	

TABLE II: ADDRESSING (continued)

		R8	R9	R10	R11	R12	R13	R14	R15		
XXXX	Rs,Rd	20s	24s	28s	2Cs	30s	34s	38s	3Cs	Rs,Rd	aaaa
	*Rs,Rd	21s	25s	29s	2Ds	31s	35s	39s	3Ds	*Rs,Rd	
	*Rs+,Rd	23s	27s	2Bs	2Fs	33s	37s	3Bs	3Fs	*Rs+,Rd	
	@Rs,Rd	22s	26s	2As	2Es	32s	36s	3As	3Es	@Rs,Rd	
	Rs,*Rd	60s	64s	68s	6Cs	70s	74s	78s	7Cs	Rs,Rd	bbbb
	*Rs,*Rd	61s	65s	69s	6Ds	71s	75s	79s	7Ds	*Rs,Rd	
	*Rs+,*Rd	63s	67s	6Bs	6Fs	73s	77s	7Bs	7Fs	*Rs+,Rd	
	@Rs,*Rd	62s	66s	6As	6Es	72s	76s	7As	7Es	@Rs,Rd	
	Rs,*Rd+	E0s	E4s	E8s	ECs	F0s	F4s	F8s	FCs	Rs,Rd	cccc
	*Rs,*Rd+	E1s	E5s	E9s	EDs	F1s	F5s	F9s	FDs	*Rs,Rd	
	*Rs+,*Rd+	E3s	E7s	EBs	EFs	F3s	F7s	FBs	FFs	*Rs+,Rd	
	@Rs,*Rd+	E2s	E6s	EAs	EEs	F2s	F6s	FAs	FEs	@Rs,Rd	
	Rs,@Rd	A0s	A4s	A8s	ACs	B0s	B4s	B8s	BCs	Rs,Rd	dddd
	*Rs,@Rd	A1s	A5s	A9s	ADs	B1s	B5s	B9s	BDs	*Rs,Rd	
	*Rs+,@Rd	A3s	A7s	ABs	AFs	B3s	B7s	BBs	BFs	*Rs+,Rd	
	@Rs,@Rd	A2s	A6s	AAs	AEs	B2s	B6s	BAs	BEs	@Rs,Rd	

SOFTWARE PRODUCT ANNOUNCEMENT

Tired of writing or patching programs in hex? Then our new "Instant Input Assembler" is just what you have been waiting for. The "Instant Input Assembler" offers most standard assembler features, except for symbolic labels. The unique difference is that it operates in conversational mode. It accepts input from the operator terminal and immediately translates it to machine code. No need to edit and punch a tape first. Furthermore, the "Instant Input Assembler" is delivered in PROM so that it is always ready for use. To activate the assembler, just jump to the start of it!

To order your "Instant Input Assembler", just contact your Super Starter Kit dealer - or Technico,Inc.. The assembler is delivered in two fused link PROMs, ready to be plugged into the expansion PROM area of the Super Starter Kit. In addition, you will receive complete user documentation and a source listing of the amazing 512 word "Instant Input Assembler".

When ordering the "Instant Input Assembler" it is necessary to specify the monitor version that it is to operate with. The monitor is uniquely identified by the contents of location FC00 (hex) so just tell us the contents of that location. This is accomplished by the monitor command "D FC00,FC01".

XI. SOFTWARE

A. MONITOR

The source listing of the mighty monitor is included in this section. A review of the monitor listing will help you to understand how the TI 990 instructions are used. The monitor listing is relative addressed. That is, the loader modifies the code to operate where loaded. In the kit, the monitor is loaded at #FC00. Therefore, you must add #FC00 to the address shown in the listing to obtain the PROM address of that data. For example, STRT10 is the label of the instruction at relative #16. The actual PROM address of that word is #FC00 + 16 = FC16.

In addition to the terminal commands, the monitor provides other useful features for the programmer. During power-on the monitor establishes two XOP's (Extended Operators) to be used for terminal input/output. These XOP's can be exploited by a user program to perform input/output to the user terminal. XOP 1 is used for input, and XOP 2 is used for output. The program in Figure XI-1, entered by the Instant Input Assembler, uses these XOP's to print the message "pick a number from 1 to 5 " and then collect the user response. Notice that the Instant Input Assembler recognizes the XOP's by the mnemonics IN and OUT.

Figure XI-1 Use of XOP'x for Input/Output

Program entered via the "Instant Input Assembler".

```
?GF800

0100: 0201  LI R1,>110       ; R1=MESSAGE ADDRESS
0102: 0110
0104: 2C91  OUT *R1          ; PRINT (R1)
0106: D031  MOVB *R1+,R0     ; ADVANCE AND TEST FOR END
0108: 16FD  JNE >104         ; CONTINUE TILL END
010A: 2C41  IN R1            ; GET REPLY
010C: 10F9  JMP >100         ; REPEAT THE PROCESS
010E:       /110
0110: 0D0A  +>0D0A           ; CR,LF, THEN MESSAGE
0112: 5049  $PICK A NUMBER FROM 1 TO 5
0114: 434B
0116: 2041
0118: 204E
011A: 554D
011C: 4245
011E: 5220
0120: 4652
0122: 4F4D
0124: 2031
0126: 2054
0128: 4F20
012A: 3520
012C: 0000  +0               ; STOPPER
012E:


------- EXECUTE THE PROGRAM


?G100

PICK A NUMBER FROM 1 TO 5 3
PICK A NUMBER FROM 1 TO 5 2
PICK A NUMBER FROM 1 TO 5 1
PICK A NUMBER FROM 1 TO 5 0
```

Other routines in the monitor are also useful. Some of them are:

TYPEN — Proceed to a new line on the terminal. Uses register R4 as scratch. Called by BL @TYPEN.

DMEMN — Display the contents of register R1 as four hex digits. The value is displayed on a new line and is followed by a ":". Input in register R1. Registers R0,R4, R5, and R7 are used as scratch. Called by BL @DMEMN.

DISRG — Display contents of R5 as four hex digits. The format is "XY = dddd" where "XY" are any two characters following the call. Input in R5 and word following the call. Registers R0,R4,R5,and R7 used as scratch. Called as follows:

```
BL  @DISRG
DATA  'XY'
```

TYPEWD — Display the contents of R5 as four hex digits. Input in R5. Registers R0,R4,R5 used as scratch. Called by BL @TYPEWD.

RDNUM This is a powerful routine for accepting hex parameters from the operator. It will read one, two, or three parameters and put them in R1,R2,R3. Refer to the source listing for further details of RDNUM.

DUMP Dump memory from address in R1 to the address in R2. Registers R0,R1,R5 used as scratch. The following will dump #107 to #311 and then return to the user.

```
LI  R1,>107
LI  R2,>311
BL  @DUMP
```

BDISPS Display the leftmost byte of R5 as two hex digits preceeded by a space. Input is in R5. Registers R0,R4,R5 used as scratch. Called by BL @BDISPS.

B. SUPER STARTER GAMES

The Super Starter Game package is a set of four games that you can play against your computer. The listing of the games is included in this section. Like the monitor, the games are relative addressed. If you wish to run the games in RAM (it takes 1K words) load the first dump following the source listing and jump to the start (via G D2). If you want to put the games in EPROM, first load the second dump following the source listing into RAM. Then program it into EPROM (via PB0,7FE,0). To execute the games in EPROM just jump to them (via GF000). Be sure you load the proper dump or the games will not work. If your kit does not have 1K words of RAM, you must enter the program a piece at a time and program each segment into the EPROM. Be careful to get the addressing correct or the games won't work. To be sure you have programmed the EPROM's correctly just dump them and recheck the dump against the second dump in this section. To dump the EPROM type "DF000, F7FF".

```
                        TITL 'TMS9900 MIGHTY MONITOR (VER3 - 12/1/77)'
0000                IDTMM  IDT
0000                       DREG
                    *
                    * NOTICE: WHEN THE MONITOR IS ENTERED IT WILL
                    * AWAIT USER INPUT TO DETERMINE THE BAUD RATE
                    * OF THE TERMINAL DEVICE.  THE USER SHOULD
                    * TYPE AN 'X' TO SET THE BAUD RATE.
                    * DO NOT TYPE 'CARRIAGE RETURN' AS IT WILL
                    * NOT WORK!!!
                    *
                    * THE BASIC TMS9900 DEBUG MONITOR OFFERS THE
                    * FOLLOWING SET OF COMMANDS(PARAMETERS IN []
                    * ARE OPTIONAL):
                    *
                    * A <ADDRESS>                              ALTER
                    * B [<ADDRESS>] [<WORD COUNT>]             BREAKPOINT
                    * C <START> <END> <TARGET>                 COPY
                    * D <START>[<END>]                         DUMP
                    * G [<ADDRESS>]                           GO
                    * H <NUMBER-1> <NUMBER-2>                 HEX ARITH
                    * I <BIT>                                 INSPECT BIT
                    * L [<ADDRESS>]                           LOAD
                    * M <BIT> <VALUE>                         MODIFY BIT
                    * P <START> <END> <TARGET>                PROGRAM
                    * S <1ST> <INC> <TOTAL>                   SNAP
                    *   ?R [<REG-1> <REG-2>]
                    *   ?M [<START> <END>]
                    * W <REG>[<REG>]                           WORKSPACE DUMP
                    *
                    *
                    * EXTERNAL DEFINITIONS
                    *
                           DEF  TYPE,TYPEN,TYPEH
                           DEF  DMEMN,TYPEWD,RDNUM
                    *
                    * SYSTEM EQUIVALENCES
                    *
0001                PRG    EQU  1                  ; PROGRAM MODE
0000                TTYI   EQU  0                  ; TTY INPUT
0000                TTYO   EQU  0                  ; TTY OUTPUT
2C00                XOP0   EQU  >2C00              ; XOP-0
1000                NOOP   EQU  >1000              ; NO-OP
0080                MTRWP  EQU  >80                ; MONITOR WORKSPACE
00B0                USRWP  EQU  >B0                ; USER WORKSPACE
0090                XOPWS  EQU  >90                ; XOP WORKSPACE(8 REG.)
0020                DELAY  EQU  >20                ; DELAY WORD
0026                BREAK  EQU  >26                ; BREAKPOINT AREA
0028                BKRTN  EQU  BREAK+2            ; BREAK RETURN
000D                CR     EQU  >0D                ; CARRIAGE RETURN
0A0D                CRLF   EQU  >0A0D              ; CAR. RET., LINE FEED
001A                MAX    EQU  26                 ; (NO. OF COMM. + 1)*2
                    *
                    * THE FOLLOWING AREA OF RAM IS USED
                    * BY THE MONITOR
                    *
                    * 20  CR DELAY TIME
                    * 22  ECHO FLAG
```

```
                    * 24  TERMINAL SPEED
                    * 26  (BREAK) NO. OF WORDS FOR TRAP
                    * 28  USER INST. ONE
                    * 2A             TWO
                    * 2C             THREE
                    * 2E  RETURN BRANCH (TWO WORDS)
                    * 32  NEXT STOP
                    * 34  STOP INCREMENT
                    * 36  MAX NO. OF STOPS
                    * 38  SNAP REG - FIRST
                    * 3A             LAST
                    * 3C  SNAP MEM - FIRST
                    * 3E             LAST
                    * 40-43  XOP-0 BREAKPOINTS
                    * 44-47  XOP-1 INPUT
                    * 48-4B  XOP-2 OUTPUT
                    * 4C-4F  XOP-3
                    * 50-53  XOP-4
                    * 54-57  XOP-5
                    * 58-5B  XOP-6
                    * 5C-5F  XOP-7
                    * 60-63  XOP-8
                    * 64-67  XOP-9
                    * 68-6B  XOP-10
                    * 6C-6F  XOP-11
                    * 70-73  XOP-12
                    * 74-77  XOP-13
                    * 78-7B  XOP-14
                    * 7C-7F  XOP-15
                    * 80-9F  MONITOR WORKSPACE
                    * A0-AF  XOP WORKSPACE (ONLY 8 REGISTERS)
                    * B0  USER R0
                    * B2       R1
                    * B4       R2
                    * B6       R3
                    * B8       R4
                    * BA       R5
                    * BC       R6
                    * BE       R7
                    * C0       R8
                    * C2       R9
                    * C4       RA (R10)
                    * C6       RB (R11)
                    * C8       RC (R12-USER CRU BASE)
                    * CA       RD (R13)
                    * CC       RE (R14)
                    * CE       RF (R15)
                    *
                    *
                    * THE FOLLOWING IS MONITOR POWER UP
                    * SEQUENCE
                    *
0000 02E0 0080  START  LWPI MTRWP
0004 04CC              CLR  R12              ; SET CRU BASE
0006 1D01              SBO  PRG              ; CLEAR PROG. MODE
0008 020D 00B0         LI   R13,USRWP        ; SET USER WP
000C 0201 0040         LI   R1,>40           ; SET UP XOP VECTORS
0010 0203 036E         LI   R3,XOPTB         ; (WORKSPACE,ENTRY)
```

```
0014 02A2                STWP R2                  ; BREAK USES MON. WS
0016 CC42       STRT10   MOV  R2,*R1+
0018 0202 0090           LI   R2,XOPWS            ; OTHERS USE XOP WS
001C CC73                MOV  *R3+,*R1+
001E 16FB                JNE  STRT10
0020 0200 FFED           LI   R0,-19              ; R0=TERM. TIMER
0024 C702                MOV  R2,*R12             ; INT. LEV. 0, W.S.
0026 0201 0020           LI   R1,DELAY
002A CC41                MOV  R1,*R1+             ; CLEAR DELAY
002C 0731                SETO *R1+                ; SET ECHO
002E 1D00                SBO  TTYO                ; TTY=HIGH
0030 1F00                TB   TTYI                ; CRT?
0032 1302                JEQ  STRT20              ; NO
0034 06A0 E800           BL   @>E800              ; SETUP FOR CRT
0038 1F00       STRT20   TB   TTYI                ; WAIT FOR START
003A 13FE                JEQ  STRT20
003C 0580       STRT30   INC  R0                  ; MEASURE A BIT
003E 1F00                TB   TTYI
0040 16FD                JNE  STRT30
0042 0920       STRT40   SRL  R0,2                ; REDUCE TO BIT COUNT
0044 CC40                MOV  R0,*R1+             ; SAVE SPEED
0046 1000                NOP                      ; TO KEEP ADDRESSES PER VER2
                 *
                 * REMOVE ANY BREAKPOINTS AND THEN
                 * ENTER MONITOR
                 *
0048 05C1                INCT R1                  ; ADVANCE TO BREAK RET.
004A 0972                SRL  R2,7                ; R2=1(IT WAS 90-HEX)
                 *
                 * ROUTINE: BREAK
                 * ESTABLISH A BREAKPOINT OR SNAP AT (R1),
                 * REMOVING R2 INSTRUCTIONS AND SETTING
                 * NEXT=-1, ANY PRIOR BREAK IS REMOVED.
                 * IF OLD BREAK DOES NOT CONTAIN (XOP) IT IS
                 * NOT DISTURBED.  SINCE R1 IS  PRESET TO BKRTN, IT
                 * CAN ACT AS A BREAKPOINT REMOVAL.
                 *
004C 0203 0026   BRK      LI   R3,BREAK           ; R3=BREAK POINTER
0050 C033                MOV  *R3+,R0             ; GET NO. OF WORDS
0052 C123 0008           MOV  @8(R3),R4           ; GET RETURN
0056 6100                S    R0,R4               ; READJUST IT TO START
0058 6100                S    R0,R4
005A C154                MOV  *R4,R5              ; CHECK FOR XOP
005E             BRKXOP  EQU  $+2
005C 0285 2C00           CI   R5,XOP0
0060 1601                JNE  BRK1                ; IF NOT XOP, SKIP RESTORE
0062 C513                MOV  *R3,*R4             ; RESTORE CODE
0064 0643        BRK1    DECT R3                  ; RESET R3
0066 0742                ABS  R2                  ; IF R2=-1, R2=1
0068 CCC2                MOV  R2,*R3+             ; STORE NO. OF WORDS
006A C111                MOV  *R1,R4              ; GET INST
006C 0602                DEC  R2
006E CC60 005E           MOV  @BRKXOP,*R1+        ; PUT IN XOP
0072 CCC4        BRK2    MOV  R4,*R3+             ; SAVE INST
0074 0204 1000           LI   R4,NOOP             ; PRESET R=NOOP
0078 C082                MOV  R2,R2               ; IF R2 NOT 0, GET INST.
007A 1302                JEQ  BRK3
007C 0602                DEC  R2
```

```
007E C131              MOV  *R1+,R4
0080 0283 002E  BRK3   CI   R3,BREAK+8        ; CONT TILL THREE WORDS SET
0084 16F6              JNE  BRK2
0086 CCE0 0338         MOV  @BRANCH,*R3+      ; SET RETURN BRANCH
008A CCC1              MOV  R1,*R3+           ; PUT IN RETURN ADDRESS
008C 0713              SETO *R3               ; R2=-1
008E 1030              JMP  MTR               ; GOTO MONITOR
                *
                * ROUTINE: TYPE
                * TYPE THE RIGHT BYTE OF R4.  AFTER THAT,
                * TYPE THE LEFT BYTE IF IT IS NOT ZERO.
                *
0090 06C4       TYPE   SWPB R4                ; PUT IN LEFT BYTE
0092 2C84       TYPE1  OUT  R4                ; OUTPUT R4
0094 0A84              SLA  R4,8              ; ANOTHER CHAR?
0096 16FD              JNE  TYPE1             ; YES-TYPE IT
0098 045B       TYPEX  B    *R11               ; RETURN
                *
                * ROUTINE: GET
                * PROMPT THE OPERATOR USING (R11), THEN* GET AN UPDATED VALUE
                * TWO ENTRIES IS -1.
                *
009A C13B       GET    MOV  *R11+,R4          ; GET PROMPT
009C C68B              MOV  R11,*R10          ; SAVE RETURN
009E 06A0 0090         BL   @TYPE             ; PROMPT THE OPERATOR
00A2 0701              SETO R1                ; SET DEFAULTS
00A4 C081              MOV  R1,R2
00A6 0205 0007         LI   R5,7              ; GET USER INPUT
00AA 06A0 021E         BL   @RDNUMB
00AE C2DA              MOV  *R10,R11          ; RESET RETURN
00B0 CE81              MOV  R1,*R10+
00B2 CE82              MOV  R2,*R10+
                *
                * ROUTINE: TYPEN
                * PROCEED TO A NEW LINE ON THE TERMINAL
                * PRINT CR,LF, THEN WAIT
                *
00B4 0204 0A0D  TYPEN  LI   R4,CRLF           ; PRINT THE CRLF
00B7            CRET   EQU  $-1
00B8 10EB              JMP  TYPE
                *
                * ROUTINE: DMEMN
                * DISPLAY R1 ON A NEW LINE IN FORMAT:
                * 'XXXX:'
                *
00BA 2D         DASH   BYTE '-'
00BB 3D         EQUAL  BYTE '='
00BC C1CB       DMEMN  MOV  R11,R7            ; SAVE EXIT
00BE 06A0 00B4         BL   @TYPEN            ; GOTO NEW LINE
00C2 C141              MOV  R1,R5             ; DISPLAY R1
00C4 06A0 01BA         BL   @TYPEWD           ; DISPLAY
00C8 2CA0 02D1         OUT  @COLON            ; OUTPUT ':'
00CC 0457              B    *R7               ; EXIT
                *
                * ROUTINE: DISRG
                * DISPLAY REGISTER R5 ON CURRENT LINE.
                * TITLE OF THE DISPLAY IS IN (R11).
                *
```

```
00CE C13B        DISRG  MOV   *R11+,R4          ; GET TITLE
00D0 C1CB        DISRA  MOV   R11,R7            ; SAVE EXIT ADDRESS
00D2 06C4               SWPB  R4
00D4 06A0 0090          BL    @TYPE            ; TYPE TITLE
00D8 2CA0 00BB          OUT   @EQUAL           ; OUTPUT '='
00DC 06A0 01BA          BL    @TYPEWD          ; OUTPUT VALUE
00E0 2CA0 0287          OUT   @SPACE           ; SPACE AND EXIT
00E4 0457               B     *R7
                  *
                  * BASIC MONITOR LOOP.  QUERY OPERATOR FOR
                  * DESIRED FUNCTION; GATHER PARAMETERS; AND
                  * TRANSFER CONTROL TO APPROPRIATE ROUTINE.
                  *
00E6 02E0 0080    MTRN   LWPI  MTRWP
00EA 04CC               CLR   R12              ; RESET CRU
00EC 06A0 00B4          BL    @TYPEN           ; NEW LINE
00F0 2CA0 013E    MTR   OUT   @QUEST           ; ISSUE PROMPT
00F4 2C44               IN    R4               ; GET REPLY
00F6 0201 033C          LI    R1,TABC          ; SEARCH TABLE OF COMMANDS
00FA C281         FINDC  MOV   R1,R10           ; SAVE TABLE POINTER
00FC C171                MOV   *R1+,R5          ; GET NEXT TABLE ENTRY
00FE 13F3                JEQ   MTRN             ; IF ZERO-TABLE EXHAUSTED
0100 9144                CB    R4,R5            ; COMPARE TO USER ENTRY
0102 16FB                JNE   FINDC            ; IF NO MATCH - CONT. SEARCH
0104 06A0 0218           BL    @RDNUM           ; GET PARAMETERS
0108 0284 000D           CI    R4,CR            ;'FORCE NEW LINE IF
010C 1303                JEQ   NEWL             ;  TERMINATED BY CR OR IF
010E C01A                MOV   *R10,R0          ;  INDICATED BY P. D.
0110 0A90                SLA   R0,9
0112 1702                JNC   CONT
0114 06A0 00B4    NEWL   BL    @TYPEN
0118 C005         CONT   MOV   R5,R0            ; R5=ODD NO. IF PARAM. O.K.
011A 0810                SRA   R0,1
011C 17E4                JNC   MTRN             ; ILLEGAL ENTRY
                  *
                  * WHEN BRANCHING TO THE INDIVIDUAL COMMAND
                  * PROCESOR, THE FOLLOWING INFO IS PROVIDED:
                  *    R5=PARAM DEC, SHIFTED BY NO. OF
                  *      PARAMS INPUT
                  *    R1=PARAMETER ONE (DEFAULT BKRTN)
                  *    R2=PAREMETER TWO (DEFAULT >FFFF)
                  *    R3=PARAMETER THREE (NO SPECIFIC DEFAULT)
                  *
011E C2AA 001A           MOV   @MAX(R10),R10    ; BRANCH TO COMMAND
0122 069A                BL    *R10             ; PROCESSING ROUTINE
0124 10E0                JMP   MTRN             ; RETURN TO LOOP
                  *
                  * ROUTINE:  COPY
                  * COPY MEMORY FROM (R1) TO (R2) INTO (R3)
                  * ANY NUMBER OF BYTES MAY BE MOVED
                  *
0126 0582         COPY   INC   R2
0128 DCF1         COPY10 MOVB  *R1+,*R3+         ; MOVE ONE BYTE
012A 8081                C     R1,R2            ; TEST END
012C 16FD                JNE   COPY10           ; CONTINUE TILL DONE
012E 10E0                JMP   MTR
                  *
                  * ROUTINE:  SNAP
```

```
                    * ESTABLISH PRIOR BREAKPOINT AS A SNAP.
                    * IF NO PARAMETERS ENTERED, USE EXISTING
                    * DATA; OTHERWISE R1= FIRST SNAP, R2= SNAP
                    * INCREMENT, R3= MAXIMUM NO. OF SNAPS.
                    * IF NEW PARAMETERS ENTERED, QUERY OPERATOR
                    * TO GET REGISTERS AND MEMORY TO BE DUMPED
                    *
0130 020A 0032      SNAP   LI    R10,BREAK+12      ; R10=BREAK POINT
0134 CE81                  MOV   R1,*R10+          ; NEXT=R1
0136 CE82                  MOV   R2,*R10+          ; INC-R2
0138 CE83                  MOV   R3,*R10+          ; SET MAX.=R3
013A 06A0 009A             BL    @GET
013E                QUEST  EQU   $
013E 3F52                  TEXT  '?R'
0140 06A0 009A             BL    @GET
0144 3F4D                  TEXT  '?M'
0146 10D4                  JMP   MTR               ; BACK TO MONITOR
                    *
                    * ROUTINE:  BKIN
                    * THIS ROUTINE IS ENTERED VIA A USER BREAK.
                    * IT PRINTS WP, PC, ST. IF A SNAP ENTRY IT ALSO
                    * PRINTS REGISTERS AND MEMORY.
                    *
0148 0201 0028      BKIN   LI    R1,BKRTN          ; R1=BREAK PTR(BREAK+2)
014C 0621 000A             DEC   @10(R1)           ; NEXT=NEXT-1
0150 1303                  JEQ   BKDSP             ; IF ZERO-DISPLAY
0152 11C9                  JLT   MTRN              ; IF LESS-GOTO MONITOR
                    *
                    * ROUTINE: GO
                    * BRANCH TO (R1).  BRANCH VIA A RETURN WITH
                    * WORKSPACE.  GO ASSUMES R1 IS PRESET TO
                    * BKRTN.  R13(WP) MUST BE PRESET DURING POWER-UP
                    *
0154 C381           GO     MOV   R1,R14            ; PC=R1
0156 0380                  RTWP                    ; BRANCH
                    *
                    * AT THIS POINT, A SNAP HAS BEEN ENCOUNTERED.
                    * DISPLAY THE SELECTED REGISTERS AND MEMORY
                    *
0158 C14E           BKDSP  MOV   R14,R5            ; PRINT PC
015A 06A0 00B4             BL    @TYPEN           ; ON A NEW LINE
015E 06A0 00CE             BL    @DISRG
0162 5043                  TEXT  'PC'
0164 C14D                  MOV   R13,R5            ; PRINT WP
0166 06A0 00CE             BL    @DISRG
016A 5750                  TEXT  'WP'
016C C14F                  MOV   R15,R5            ; PRINT ST
016E 06A0 00CE             BL    @DISRG
0172 5354                  TEXT  'ST'
0174 C0A1 0012             MOV   @18(R1),R2        ; GET RD1,RD2
0178 C061 0010             MOV   @16(R1),R1
017C 1104                  JLT   BKDSP2            ; IF RD1=-1, NO REG DISP
017E 06A0 00B4             BL    @TYPEN            ; DISPLAY REGISTERS
0182 06A0 01E0             BL    @DISPW
0186 0203 003C      BKDSP2 LI    R3,BREAK+22       ; GET MD1,MD2
018A C073                  MOV   *R3+,R1
018C 0281 FFFF             CI    R1,-1             ; IF MD1=-1, NO DISP.
0190 1305                  JEQ   BKDSP3
```

```
0192 C093              MOV  *R3,R2          ;  SET THE END
0194 06A0 02AA         BL   @DUMP           ; DUMP
0198 06A0 00B4         BL   @TYPEN
019C 0201 0028  BKDSP3 LI   R1,BKRTN        ; DEC. MAX
01A0 0621 000E         DEC  @14(R1)
01A4 13A0              JEQ  MTRN            ; IF ZERO, GOTO MON.
01A6 C861 000C         MOV  @12(R1),@10(R1) ; SET NEXT=INC
01AA 000A
01AC 10D3              JMP  GO              ; RET. TO USER
                *
                * ROUTINE: BDISP
                * DISPLAY THE LEFTMOST BYTE OF R5,
                * PRECEEDED BY A SPACE
                *
01AE 2CA0 0287  BDISPS OUT  @SPACE          ; TYPE SPACE
01B2 06C5       BDISP  SWPB R5              ; PUT DATA IN LOWER BYTE
01B4 0200 0004         LI   R0,4            ; PRINT AND EXIT
01B8 1002              JMP  TYPEH
                *
                * ROUTINE: TYPEH
                * DISPLAY R5 AS A HEX DIGIT STRING
                * THE SHIFT COUNT IN R0 CONTROLS THE NO.
                * OF DIGITS PRINTED (12=4,4=2)
                *
01BA 0200 000C  TYPEWD LI   R0,12
01BE C105       TYPEH  MOV  R5,R4            ;.EXTRACT ONE NIBBLE
01C0 0B04              SRC  R4,R0
01C2 0244 000F         ANDI R4,>F           ; MASK OFF FOUR BITS
01C6 0224 0030         AI   R4,>30          ; ADJUST FOR ASCII
01CA 0284 003A         CI   R4,>3A          ; TEST 'A'-'F' AND
01CE 1102              JLT  TYPEH2          ;    IF SO-READJUST
01D0 0224 0007         AI   R4,7
01D4 06C4       TYPEH2 SWPB R4              ; TYPE
01D6 2C84              OUT  R4
01D8 0220 FFFC         AI   R0,-4           ; REDUCE SHIFT COUNT
01DC 18F0              JOC  TYPEH           ; CONT. TILL DONE
01DE 045B              B    *R11            ; EXIT
                *
                * ROUTINE: DISPW
                * DISPLAY WORKSPACE R(R1)-R(R2)
                *
01E0 C0CB       DISPW  MOV  R11,R3          ; SAVE RETURN
01E2 0241 000F         ANDI R1,>F           ; FORCE R1=0-F
01E6 6081              S    R1,R2           ; R2=NO. OF REG.
01E8 C101       DISPW1 MOV  R1,R4           ; FORM REG NAME
01EA 0224 5230         AI   R4,'R0'
01EE 0284 523A         CI   R4,'R9'+1
01F2 1102              JLT  DISPW2
01F4 0224 0007         AI   R4,7
01F8 C141       DISPW2 MOV  R1,R5           ; GET REGISTER
01FA 0A15              SLA  R5,1            ; FORM A WORD ADDRESS
01FC A14D              A    R13,R5
01FE C155              MOV  *R5,R5
0200 06A0 00D0         BL   @DISRA          ; DISPLAY REGISTER
0204 0602              DEC  R2              ; TEST FOR END
0206 1107              JLT  DISPW3          ; EXIT IF MINUS
0208 0581              INC  R1              ; ADVANCE REG. COUNT
020A 0281 0008         CI   R1,8            ; IF REG. 8, THEN
```

```
020E 16EC              JNE  DISPW1          ;    GOTO NEW LINE
0210 06A0 00B4         BL   @TYPEN
0214 10E9              JMP  DISPW1
0216 0453       DISPW3 B    *R3             ; EXIT
                *
                * ROUTINE:RDNUM
                * READ PARAMETERS AND PLACE THEM IN REGISTERS
                * R1,R2,R3.  PARAMETER DESCRIPTION IS IN R5 AND
                * IS SHIFTED RIGHT ONE POSITION FOR EACH PARAM.
                * THAT IS READ.  RDNUMA AVOIDS THE PRESET AND
                * INITIAL READ.  RDNUMB AVOIDS THE PRESET ONLY.
                * R1 IS PRESET TO BKRTN AND R2 IS PRESET TO >FFFF.
                *
0218 0201 0028  RDNUM  LI   R1,BKRTN        ; PRESET R1,R2
021C 0702              SETO R2
021E 04C4       RDNUMB CLR  R4              ; FIRST CHAR PRESET
0220 C20B       RDNUMA MOV  R11,R8          ; SAVE RETURN
0222 02A6              STWP R6              ; R6=WORKSPACE
0224 04C7              CLR  R7              ; RESET FLAG(R7)
0226 C004       STRT   MOV  R4,R0           ; TEST INPUT FOR HEX
0228 0220 FFD0         AI   R0,->30         ; R0=INPUT-'0'
022C 1117              JLT  NOHEX           ; IF MINUS, NOT HEX
022E 0280 000A         CI   R0,10           ; CHECK R0=0-9, IF SO
0232 1108              JLT  ADDIN           ;    GOTO ADDIN
0234 0220 FFF9         AI   R0,-7           ; CHECK R0=A-F, IF SO
0238 0280 000A         CI   R0,10           ;    FALL THRU TO ADDIN
023C 110F              JLT  NOHEX
023E 0280 000F         CI   R0,15
0242 150C              JGT  NOHEX
0244 C1C7       ADDIN  MOV  R7,R7           ; R0=NEXT DIG(BINARY)
0246 1603              JNE  NONEW           ; IF FIRST, PRESET VALUE
0248 0587              INC  R7              ; SET FLAG
024A 05C6              INCT R6              ; ADVANCE TO NEXT VALUE
024C 04D6              CLR  *R6             ; CLEAR NEXT VALUE
024E C0D6       NONEW  MOV  *R6,R3          ; GET VALUE
0250 0A43              SLA  R3,4            ; MULT. BY 16
0252 A0C0              A    R0,R3           ; ADD NEW DIGIT
0254 C583              MOV  R3,*R6          ; REPLACE VALUE
0256 2C44       CONT1  IN   R4              ; GET CHAR
0258 0984              SRL  R4,8            ; RIGHT JUST.
025A 10E5              JMP  STRT            ; CONTINUE SCAN
                *
                * AT THIS POINT, WE KNOW THAT THE INPUT
                * IS NOT A HEX DIGIT, SO CHECK FOR END
                * OF ENTRY AND END OF INPUT.
                *
025C C1C7       NOHEX  MOV  R7,R7           ; IF NON NULL ENTRY, THEN
025E 1306              JEQ  TSTND           ;    REVISE THE P.D.
0260 04C7              CLR  R7              ; RESET FLAG
0262 0915              SRL  R5,1            ; UPDATE P.D.
0264 C005              MOV  R5,R0           ; IF P.D.=XX...XX000X, THEN
0266 0240 000E         ANDI R0,>E           ;    RETURN TO CALLER
026A 1303              JEQ  EXIT
026C 0284 000D  TSTND  CI   R4,CR           ; IF CR, THEN RETURN
0270 16F2              JNE  CONT1
0272 0458       EXIT   B    *R8             ; RETURN TO CALLER
                *
                * ROUTINE: ALTER
```

```
                    * DISPLAY (P1); AWAIT OPERATOR UPDATE, IF ANY;
                    * INCREMENT ADDRESS AND CONTINUE.  IF THE
                    * ENTRY IS TERMINATED BY A CR, DISPLAY CURRENT
                    * ADDRESS ON A NEW LINE, THEN THE DATA BYTE.
                    * IF SPACE ENTERED, SKIP UPDATE OF THIS BYTE.
                    *
0274 C081           ALT     MOV   R1,R2             ; SAVE ADDRESS
0276 D152           ALT1    MOVB  *R2,R5            ; DISPLAY (R2)
0278 06A0 01B2              BL    @BDISP
027C 2CA0 00BA              OUT   @DASH             ; OUTPUT '-'
0280 2C44                   IN    R4                ; GET REPLY
0282 0984                   SRL   R4,8
0284 0284 0020              CI    R4,' '            ; IF ' ', SKIP UPDATE
0287                SPACE   EQU   $-1
0288 1308                   JEQ   ALT2
028A 0205 0002              LI    R5,2              ; READ FULL REPLY
028E D052                   MOVB  *R2,R1            ; SET DEFAULT
0290 06C1                   SWPB  R1
0292 06A0 0220              BL    @RDNUMA           ; GET REPLY
0296 06C1                   SWPB  R1                ; ALTER (R2)
0298 D481                   MOVB  R1,*R2
029A 0582           ALT2    INC   R2                ; ADV. ADDR POINTER
029C 0284 000D              CI    R4,CR             ; IF TERMINATED BY CR, THEN
02A0 16EA                   JNE   ALT1              ;    TYPE CURRENT ADDRESS
02A2 C042                   MOV   R2,R1
02A4 06A0 00BC              BL    @DMEMN
02A8 10E6                   JMP   ALT1
                    *
                    * ROUTINE: DUMP
                    * DUMP THE MEMORY FROM (R1) TO (R2).  IF
                    * CALLED FROM MONITOR LOOP, DUMP WILL RETURN
                    * TO MTRN; OTHERWISE IT RETURNS TO CALLING
                    * ROUTINE.
                    *
02AA C0CB           DUMP    MOV   R11,R3            ; SAVE RETURN
02AC 06A0 00BC      DUMP1   BL    @DMEMN            ; DISPLAY ADDRESS
02B0 D171           DUMP2   MOVB  *R1+,R5           ; GET NEXT BYTE
02B2 06A0 01AE              BL    @BDISPS           ; DISPLAY IT SPACE FIRST
02B6 8081                   C     R1,R2             ; CHECK END
02B8 1BAE                   JH    DISPW3            ; IF NOT END-CONTINUE
02BA C141                   MOV   R1,R5             ; IF R1=0-EXIT, IF R1 MULT 16
02BC 13AC                   JEQ   DISPW3
02BE 0AC5                   SLA   R5,12             ;    THEN DISP ADDRESS,
02C0 13F5                   JEQ   DUMP1             ;    ELSE CONTINUE
02C2 10F6                   JMP   DUMP2             ; CONTINUE DUMP
                    *
                    * ROUTINE: LOAD
                    * LOAD A MONITOR DUMP BACK TO RAM
                    *
02C4 C081           LOAD    MOV   R1,R2             ; R2=LOAD ADDRESS
02C6 0205 0002      LOAD1   LI    R5,2              ; READ VALUE
02CA 06A0 021E              BL    @RDNUMB
02CE 0284 003A              CI    R4,':'            ; IF TERM. BY ':' RESET R3
02D1                COLON   EQU   $-1
02D2 13F8                   JEQ   LOAD
02D4 06C1                   SWPB  R1                ; DATA IN LEFT BYTE
02D6 DC81                   MOVB  R1,*R2+           ; STORE ONE BYTE
02D8 10F6                   JMP   LOAD1             ; CONTINUE
```

```
                    *
                    * ROUTINE: INSPECT
                    * INSPECT A CRU BIT (R1)
                    *
02DA 0A11           INSP    SLA   R1,1             ; ALIGN FOR CRU BASE
02DC C301                   MOV   R1,R12           ; PUT IN CRU BASE
02DE 0204 3031              LI    R4,'01'          ; SET R4=0/1
02E2 1F00                   TB    0
02E4 1601                   JNE   INSP1
02E6 06C4                   SWPB  R4
02E8 2C84           INSP1   OUT   R4                ; DISPLAY THE BIT
02EA 045B                   B     *R11              ; BACK TO MONITOR
                    *
                    * ROUTINE: MODIFY
                    * MODIFY A CRU BIT (R1) TO BE (R2)
                    *
02EC 0A11           MODIF   SLA   R1,1              ; ALIGN FOR CRU BASE
02EE C301                   MOV   R1,R12           ; SET CRU BASE
02F0 C082                   MOV   R2,R2            ; TEST BIT
02F2 1302                   JEQ   MODIF1           ; IF ZERO, JUMP
02F4 1D00                   SBO   0
02F6 1001                   JMP   MODIF2
02F8 1E00           MODIF1  SBZ   0
02FA 0460 00F0      MODIF2  B     @MTR              ; BACK TO MONITOR
                    *
                    * ROUTINE: PROG
                    * PROGRAM ROM.  SOURCE IS (R1)-(R2),
                    * ROM TARGET IS (R3)
                    *
02FE 1E01           PROG    SBZ   PRG              ; ENABLE
0300 05C2                   INCT  R2
0302 0204 00C8              LI    R4,200           ; REPEAT COUNT
0306 C141           PROG1   MOV   R1,R5            ; SAVE INPUT
0308 C183                   MOV   R3,R6
030A 0266 F000              ORI   R6,>F000         ; ADJUST FOR ROM
030E CDB5           PROG2   MOV   *R5+,*R6+         ; PROG ONE WORD
0310 0207 0006              LI    R7,6             ; ALLOW DYNAMIC RAM
0314 0607           PROG3   DEC   R7               ; TO REFRESH ITSELF
0316 16FE                   JNE   PROG3
0318 8085                   C     R5,R2
031A 16F9                   JNE   PROG2            ; CONT. THIS PASS
031C 0604                   DEC   R4
031E 16F3                   JNE   PROG1            ; NEXT PASS
0320 1D01                   SBO   PRG              ; DISABLE PROG.
0322 10EB                   JMP   MODIF2           ; BACK TO MONITOR
                    *
                    * ROUTINE: HEX
                    * PRINT R1+R2 AND R1-R2
                    *
0324 C141           HEX     MOV   R1,R5            ; SUM
0326 A142                   A     R2,R5
0328 06A0 00CE              BL    @DISRG
032C 482B                   TEXT  'H+'
032E C141                   MOV   R1,R5            ; DIFFERENCE
0330 6142                   S     R2,R5
0332 06A0 00CE              BL    @DISRG
0336 482D                   TEXT  'H-'
0338 0460 00E6      BRANCH  B     @MTRN
```

```
                    *
                    * COMMAND TABLE
                    *
033C 4102           TABC   BYTE 'A',>02          ; ALTER
033E 4287                  BYTE 'B',>87          ; BREAKPOINT
0340 4388                  BYTE 'C',>88          ; COPY
0342 4406                  BYTE 'D',>06          ; DUMP
0344 4783                  BYTE 'G',>83          ; GO
0346 4884                  BYTE 'H',>84          ; HEX ARITH.
0348 4902                  BYTE 'I',>02          ; INSPECT
034A 4C83                  BYTE 'L',>83          ; LOAD
034C 4D84                  BYTE 'M',>84          ; MODIFY
034E 5088                  BYTE 'P',>88          ; PROGRAM
0350 5388                  BYTE 'S',>88          ; SNAP
0352 5786                  BYTE 'W',>86          ; WORKSPACE DUMP
0354 0000                  BYTE 0,0              ; END OF TABLE
0356 0274 004C             DATA ALT,BRK,COPY,DUMP
035A 0126 02AA
035E 0154 0324             DATA GO,HEX,INSP,LOAD,MODIF,PROG,SNAP,DISPW
0362 02DA 02C4
0366 02EC 02FE
036A 0130 01E0
036E 0148           XOPTB  DATA BKIN
0370 039E                  DATA ROUT2
0372 0376                  DATA ROUT1
0374 0000                  DATA 0
                    *
                    * XOP ROUTINES
                    * XOP-1 = INPUT
                    * XOP-2 = OUTPUT
                    *
                    *
                    * ROUTINE: ROUT1 (TERMINAL OUTPUT)
                    * OUTPUT THE BYTE AT (R11).  IF IT IS
                    * A CARRIAGE RETURN, DELAY ACCORDING TO
                    * THE VALUE (DELAY)
                    *
0376 020A 03EA      ROUT1  LI   R10,WAITA        ; R10=INDEX TO WAIT
037A D21B                  MOVB *R11,R8          ; R8=CHARACTER
037C 0209 0002             LI   R9,2             ; R9=NO STOP BITS
0380 069A           R110   BL   *R10             ; STOP BIT WAIT
0382 0609                  DEC  R9
0384 16FD                  JNE  R110
0386 0209 0008             LI   R9,8             ; R9=CHARACTER COUNT
038A 1E00                  SBZ  TTY0             ; START BIT
038C 069A                  BL   *R10             ; WAIT FOR START BIT
038E 3048           R120   LDCR R8,1             ;(22) OUTPUT ONE BIT
0390 069A                  BL   *R10             ;(16) WAIT FOR IT
0392 0918                  SRL  R8,1             ;(14) GET NEXT BIT
0394 0609                  DEC  R9               ;(10) CONTINUE TILL DONE
0396 16FB                  JNE  R120             ;(10)
0398 1D00                  SBO  TTY0             ; STOP BIT
039A 06C8                  SWPB R8               ; REPOSITION BYTE
039C 1019                  JMP  R250             ; GO CHECK BREAK, ETC.
                    *
                    * ROUTINE: ROUT2 (TERMINAL ECHO)
                    * INPUT ONE CHARACTER FROM TERMINAL AND
                    * RETURN IT IN (R11).  IF CARRIAGE RETURN
```

```
0406 04CC                CLR  R12              ; SET BASE
0408 020A 0004  T9901A   LI   R10,4             ; ECHO (1200 BAUD)
040C 1D0A       T9901B   SBO  >A
040E 1D09                SBO  >9
0410 3208                LDCR R8,8              ; OUTPUT
0412 1F0B       T9901C   TB   >B
0414 16FE                JNE  T9901C
0416 1D0B                SBO  >B
0418 C820 0022           MOV  @>22,@>22         ; ECHO?
041C 0022
041E 1308                JEQ  T9901D            ; NO-EXIT
0420 060A                DEC  R10               ; EACH ONE OUT 4 TIME
0422 16F4                JNE  T9901B
0424 04C8                CLR  R8
0426 96E0 0432           CB   @T990CR,*R11      ; CR?
042A 1602                JNE  T9901D
042C 0609                DEC  R9                ; CR DELAY
042E 16EC                JNE  T9901A
0430 0380       T9901D   RTWP ; EXIT
0432 0D0D       T990CR   DATA >0D0D             ; CR
0434 04CC       T9902    CLR  R12              ; CHAR ECHO
0436 1F0C                TB   >C
0438 16FD                JNE  T9902
043A 04C8                CLR  R8
043C 35C8                STCR R8,7
043E 1D0C                SBO  >C
0440 D6C8                MOVB R8,*R11           ; RETURN THE CHAR
0442 0288 1B00           CI   R8,>1B00
0446 1602                JNE  T9902A
0448 0460 00E6           B    @MTRN
044C C220 0022  T9902A   MOV  @>22,R8           ; ECHO?
0450 16D7                JNE  T9901             ; YES-DO IT
0452 0380                RTWP ; NO-EXIT
                 *
                 * END OF MONITOR
                 * THE LOAD VECTOR MUST BE PATCHED IN
                 *
0454                     END  START
```

```
0244 ADDIN    0274 ALT     0276 ALT1    029A ALT2    01B2 BDISP
01AE BDISPS   0158 BKDSP   0186 BKDSP2  019C BKDSP3  0148 BKIN
0028 BKRTN    0338 BRANCH  0026 BREAK   004C BRK     0064 BRK1
0072 BRK2     0080 BRK3    005E BRKXOF  02D1 COLON   0118 CONT
0256 CONT1    0126 COPY    0128 COPY10  000D CR      00B7 CRET
0A0D CRLF     00BA DASH    0020 DELAY   01E0 DISPW   01E8 DISPW1
01F8 DISPW2   0216 DISPW3  00D0 DISRA   00CE DISRG   00BC DMEMN
02AA DUMP     02AC DUMP1   02B0 DUMP2   00BB EQUAL   0272 EXIT
00FA FINDC    009A GET     0154 GO      0324 HEX    *0000 IDTMM
02DA INSP     02E8 INSP1   02C4 LOAD    02C6 LOAD1   001A MAX
02EC MODIF    02F8 MODIF1  02FA MODIF2  00F0 MTR     00E6 MTRN
0080 MTRWP    0114 NEWL    025C NOHEX   024E NONEW   1000 NOOP
0001 PRG      02FE PROG    0306 PROG1   030E PROG2   0314 PROG3
013E QUEST    0000 R0      0001 R1      000A R10     000B R11
0380 R110     000C R12     038E R120    000D R13     000E R14
000F R15      0002 R2      03A2 R210    03AC R220    03B4 R230
```

```
 03BC R240     03D0 R250     03DA R260     03DE R270     03E2 R280
 03E8 R290     0003 R3       0004 R4       0005 R5       0006 R6
 0007 R7       0008 R8       0009 R9       0218 RDNUM    0220 RDNUMA
 021E RDNUMB   0376 ROUT1    039E ROUT2    0130 SNAP     0287 SPACE
 0000 START    0226 STRT     0016 STRT10   0038 STRT20   003C STRT30
*0042 STRT40   0400 T9901    0408 T9901A   040C T9901B   0412 T9901C
 0430 T9901D   0434 T9902    044C T9902A   0432 T990CR   033C TABC
 026C TSTND    0000 TTYI     0000 TTYO     0090 TYPE     0092 TYPE1
 01BE TYPEH    01D4 TYPEH2   00B4 TYPEN    01BA TYPEWD  *0098 TYPEX
 00B0 USRWP    03EA WAITA    03EE WAITB    03F2 WAITC    2C00 XOP0
 036E XOPTB    0090 XOPWS
EDIT/ASM/LOAD?
```

```
                        TITL 'SUPER STARTER GAMES (VER 6/77)'
0000                IDTSSG IDT  'IDTSSG'
0000                        DREG                    ; DEFINE REGISTERS
                    *
                    * RAM DATA BASE
                    * ORDER IS IMPORTANT, CHANGE  WITH CARE
                    *
00B0                        AORG >B0                ; ADDRESS RAM
00B0                SEED    BSS  2                  ; RANDOM NO. SEED
00B2                BROLL   BSS  2                  ; CURRENT BANKROLL
00B4                WAGER   BSS  2                  ; CURRENT WAGER
00B6                PTOT    BSS  2                  ; PLAYER TOTAL
00B8                PACE    BSS  2                  ; PLAYER ACE COUNT
00BA                CTOT    BSS  2                  ; COMPUTER TOTAL
00BC                CACE    BSS  2                  ; COMPUTER ACE COUNT
00BE                CNT     BSS  2                  ; CARDS REMAINING
00C0                CHLD    BSS  2                  ; COMPUTER HOLD
00C2                DRW     BSS  2                  ; CARD LAST DRAWN
00C4                DECK    BSS  14                 ; THE DECK
00B6                GUESS   EQU  PTOT               ; TOTAL NO. OF GUESSES
00B8                GAMES   EQU  PACE               ; TOTAL NO. OF GAMES
00C4                NO      EQU  DECK               ; NUMBER
00B6                POINT   EQU  PTOT               ; POINT
00B8                ROLL    EQU  PACE               ; ROLL
00B6                NUM1    EQU  PTOT               ; NUMBER ONE
00B8                NUM2    EQU  PACE               ; NUMBER TWO
                    *
                    * MONITOR INTERFACE
                    *
0000                        RORG *                  ; CHANGE TO RELATIVE
0000 0460 06F4              B    @BEGIN             ; GO TO START
0000                TTY     EQU  0                  ; TTY BIT
                    *
                    * ROM DATA BASE
                    *
0004 07             BELLS   BYTE 7                  ; BELL CODE
0005 2E             DECML   BYTE '.'
0006 20             SPACE   BYTE ' '
0007 3F             QUEST   BYTE '?'
0008 40             ATSGN   BYTE '@'
0009 0D             CR      BYTE >0D                ; CARRIAGE RETURN
000A 0A             LF      BYTE >0A                ; LINE FEED
000C 03E8           BANK    DATA 1000               ; SIZE OF BANK
000E 4132 3334      LABEL   TEXT 'A23456789TJQK'
0012 3536 3738
0016 3954 4A51
001A 4B
001C 000A           TEN     DATA 10                 ; CONSTANT
001E 000D           C13     DATA 13                 ; CONSTANT
0020 0006           C6      DATA 6                  ; CONSTANT
0022 038A 0042      GTAB    DATA BLK10,'B'          ; TABLE OF SELECTIONS
0026 04F6 0046              DATA GS00,'F'
002A 05E8 0043              DATA CRP10,'C'
002E 0678 0041              DATA AD10,'A'
0032 0000                   DATA 0                  ; END OF TABLE FLAG
0034 01             ONE     BYTE 1                  ; BYTE INCREMENT
                    *
                    * PLAYER MESSAGES
```

```
                        *
0035 424C 4143  MESS00 TEXT 'BLACKJACK@'
0039 4B4A 4143
003D 4B40
003F 494E 4954  MESS01 TEXT 'INITIAL BANKROLL IS $200@'
0043 4941 4C20
0047 4241 4E4B
004B 524F 4C4C
004F 2049 5320
0053 2432 3030
0057 40
0058 5245 4144  MESS02 TEXT 'READY?@'
005C 593F 40
005F 484F 5553  MESS03 TEXT 'HOUSE LIMIT IS $100@'
0063 4520 4C49
0067 4D49 5420
006B 4953 2024
006F 3130 3040
0073 5741 4745  MESS04 TEXT 'WAGER?@'
0077 523F 40
007A 4849 543F  MESS05 TEXT 'HIT?@'
007E 40
007F 4445 414C  MESS06 TEXT 'DEALER HOLDS @'
0083 4552 2048
0087 4F4C 4453
008B 2040
008D 4445 414C  MESS07 TEXT 'DEALER BUSTED@'
0091 4552 2042
0095 5553 5445
0099 4440
009B 594F 5520  MESS08 TEXT 'YOU WIN@'
009F 5749 4E40
00A3 594F 5552  MESS09 TEXT 'YOUR BANKROLL IS $@'
00A7 2042 414E
00AB 4B52 4F4C
00AF 4C20 4953
00B3 2024 40
00B6 4445 414C  MESS10 TEXT 'DEALER TOTAL IS - @'
00BA 4552 2054
00BE 4F54 414C
00C2 2049 5320
00C6 2D20 40
00C9 594F 5520  MESS11 TEXT 'YOU LOSE@'
00CD 4C4F 5345
00D1 40
00D2 4741 4D45  MESS12 TEXT 'GAME OVER - YOU ARE BROKE!@'
00D6 204F 5645
00DA 5220 2D20
00DE 594F 5520
00E2 4152 4520
00E6 4252 4F4B
00EA 4521 40
00ED 2121 2120  MESS13 TEXT '!!! YOU BROKE THE BANK !!!@'
00F1 594F 5520
00F5 4252 4F4B
00F9 4520 5448
00FD 4520 4241
0101 4E4B 2021
```

```
0105 2121 40
0108 534F 5252  MESS14 TEXT 'SORRY, NO CREDIT@'
010C 592C 204E
0110 4F20 4352
0114 4544 4954
0118 40
0119 594F 5520  MESS15 TEXT 'YOU DRAW - @'
011D 4452 4157
0121 202D 2040
0125 594F 5552  MESS16 TEXT 'YOUR TOTAL IS - @'
0129 2054 4F54
012D 414C 2049
0131 5320 2D20
0135 40
0136 594F 5520  MESS17 TEXT 'YOU BUSTED@'
013A 4255 5354
013E 4544 40
0141 424C 4143  MESS18 TEXT 'BLACKJACK!@'
0145 4B4A 4143
0149 4B21 40
014C 4445 414C  MESS19 TEXT 'DEALER DRAWS - @'
0150 4552 2044
0154 5241 5753
0158 202D 2040
                *
015C 464F 5552  MESS20 TEXT 'FOUR DIGIT GUESS@'
0160 2044 4947
0164 4954 2047
0168 5545 5353
016C 40
016D 4755 4553  MESS21 TEXT 'GUESS NO. @'
0171 5320 4E4F
0175 2E20 40
0178 4449 4749  MESS22 TEXT 'DIGITS CORRECT - @'
017C 5453 2043
0180 4F52 5245
0184 4354 202D
0188 2040
018A 494E 2043  MESS23 TEXT 'IN CORRECT POS.- @'
018E 4F52 5245
0192 4354 2050
0196 4F53 2E2D
019A 2040
019C 4C41 5354  MESS24 TEXT 'LAST SHOT, YOU LOSE.  IT WAS - @'
01A0 2053 484F
01A4 542C 2059
01A8 4F55 204C
01AC 4F53 452E
01B0 2020 4954
01B4 2057 4153
01B8 202D 2040
01BC 3F3F 3F20  MESS25 TEXT '??? NUMBERS ONLY!@'
01C0 4E55 4D42
01C4 4552 5320
01C8 4F4E 4C59
01CC 2140
01CE 5448 4154  MESS26 TEXT 'THATS IT!@'
01D2 5320 4954
```

```
01D6 2140
01D8 594F 5552  MESS27 TEXT 'YOUR AVERAGE IS - @'
01DC 2041 5645
01E0 5241 4745
01E4 2049 5320
01E8 2D20 40
                *
01EB 5745 4C43  MESS30 TEXT 'WELCOME TO THE S.S. GAMEROOM@'
01EF 4F4D 4520
01F3 544F 2054
01F7 4845 2053
01FB 2E53 2E20
01FF 4741 4D45
0203 524F 4F4D
0207 40
0208 4348 4F4F  MESS31 TEXT 'CHOOSE YOUR GAME (BY FIRST LETTER)@'
020C 5345 2059
0210 4F55 5220
0214 4741 4D45
0218 2028 4259
021C 2046 4952
0220 5354 204C
0224 4554 5445
0228 5229 40
                *
022B 4352 4150  MESS40 TEXT 'CRAPS@'
022F 5340
0231 524F 4C4C  MESS41 TEXT 'ROLL .....@'
0235 202E 2E2E
0239 2E2E 40
023C 594F 5552  MESS42 TEXT 'YOUR POINT - @'
0240 2050 4F49
0244 4E54 202D
0248 2040
                *
024A 4143 4559  MESS50 TEXT 'ACEY DUECEY@'
024E 2044 5545
0252 4345 5940
0256 5448 4520  MESS51 TEXT 'THE PAIR - @'
025A 5041 4952
025E 202D 2040
                *
                * COMMON SUBROUTINES
                *
                *
                * ROUTINE: TYPEN
                * GO TO A NEW LINE
                *
0262                    EVEN ; MUST BE EVEN ADDRESS
0262 2CA0 0009  TYPEN   OUT  @CR
0266 2CA0 000A          OUT  @LF
026A 045B               B    *R11
                *
                * ROUTINE: WIN, LOSE
                * PRINT WIN OR LOSE MESSAGE,
                * UPDATE AND SHOW THE TOTALS
                * CHECK FOR OVERFLOW AND UNDERFLOW
                *
```

```
026C C0CB        WIN     MOV   R11,R3          ; SAVE RETURN
026E 069F                BL    *R15
0270 009B                DATA  MESS08
0272 C020 00B4           MOV   @WAGER,R0
0276 1006                JMP   SHOW
0278 C0CB        LOSE    MOV   R11,R3          ; SAVE RETURN
027A 069F                BL    *R15
027C 00C9                DATA  MESS11
027E C020 00B4           MOV   @WAGER,R0
0282 0500                NEG   R0
0284 A020 00B2   SHOW    A     @BROLL,R0       ; R0=NEW TOTAL
0288 110C                JLT   SHOW10          ; BROKE
028A 130B                JEQ   SHOW10          ; BROKE
028C 8800 000C           C     R0,@BANK        ; CHECK AGAINST BANK LIMIT
0290 150B                JGT   SHOW20          ; BIG WINNER
0292 C800 00B2           MOV   R0,@BROLL       ; SAVE NEW BANKROLL
0296 069F                BL    *R15            ; DISPLAY IT
0298 00A3                DATA  MESS09
029A C160 00B2           MOV   @BROLL,R5
029E C203                MOV   R3,R8           ; SETUP RETURN FOR DISP
02A0 101E                JMP   DISPA           ; DISPLAY AND EXIT
02A2 069F        SHOW10  BL    *R15            ; BROKE
02A4 00D2                DATA  MESS12
02A6 1008                JMP   SHOW40
02A8 069F        SHOW20  BL    *R15            ; BANK BROKE
02AA 00ED                DATA  MESS13
02AC 0201 0014           LI    R1,20           ; RING THE BELLS
02B0 2CA0 0004   SHOW30  OUT   @BELLS          ; RING THE BELLS!
02B4 0601                DEC   R1
02B6 16FC                JNE   SHOW30
02B8 0460 06F4   SHOW40  B     @BEGIN          ; RESTART
                 *
                 * ROUTINE: TYPED
                 * TYPE A DIGIT IN R4
                 *
02BC 0284 000A   TYPED   CI    R4,10           ; CHECK FOR TWO DIGITS
02C0 1108                JLT   TYPE3           ; JUMP IF NOT
02C2 04C3        DISPD   CLR   R3              ; DISPLAY TWO DEC. DIG.
02C4 3CE0 001C           DIV   @TEN,R3
02C8 0A83                SLA   R3,8
02CA A103                A     R3,R4
02CC 0224 3000   TYPE2   AI    R4,>3000        ; ADJUST SECOND DIGIT
02D0 2C84                OUT   R4              ; TYPE DIGIT
02D2 0224 0030   TYPE3   AI    R4,>30          ; ADJUST FOR ASCII
02D6 06C4                SWPB  R4
02D8 2C84                OUT   R4
02DA 045B                B     *R11
                 * ROUTINE: DISP
                 * DISPLAY THE CONTENTS OF R5
                 *
02DC C20B        DISP    MOV   R11,R8
02DE 0201 03E8   DISPA   LI    R1,1000         ; SETUP DIVISOR
02E2 C105        DISP10  MOV   R5,R4
02E4 04C3                CLR   R3              ; (R3,R4)=INPUT
02E6 3CC1                DIV   R1,R3           ; R3=INPUT/DIVISOR
02E8 C103                MOV   R3,R4
02EA 04C3                CLR   R3
02EC 3CE0 001C           DIV   @TEN,R3         ; R4=NEXT DIGIT
```

```
02F0 06A0 02BC          BL    @TYPED
02F4 04C0               CLR   R0
02F6 3C20 001C          DIV   @TEN,R0
02FA C040               MOV   R0,R1
02FC 16F2               JNE   DISP10
02FE 0458               B     *R8
                  *
                  * ROUTINE: RANDOM
                  * GENERATE A RANDOM NUMBER
                  * N(I)=N(I-1)*C [MOD 2**16]
                  * C=R*2**S+1=5*2**3+1=41
                  * PERIOD=2**(16-3)=2**13
                  *
0300 0029         GEN     DATA 41                ; GENERATOR
0302 C020 00B0    RANDOM  MOV  @SEED,R0          ; GET SEED
0306 1601               JNE  RAND10              ; IF ZERO - CORRECT IT
0308 05C0               INCT R0
030A 3820 0300    RAND10  MPY  @GEN,R0           ; (R0,R1)=NEXT NO.
030E C801 00B0          MOV  R1,@SEED          ; RESET SEED
0312 045B               B    *R11              ; EXIT
                  *
                  * ROUTINE: WAIT
                  * WAIT FOR OPERATOR GO AHEAD, AND
                  * RANDOMIZE THE GENERATOR
                  *
0314 C0CB         WAIT    MOV  R11,R3            ; SAVE RETURN
0316 069F               BL   *R15              ; READY?
0318 0058               DATA MESS02
031A 0201 00C8          LI   R1,200            ; PRESET BANKROLL
031E C801 00B2          MOV  R1,@BROLL         ; PRESET BANKROLL
0322 04E0 00BE          CLR  @CNT              ; CLEAR COUNTER
0326 C2C3               MOV  R3,R11            ; RESET RET. FOR ALT. ENTRY
0328 C0CB         WAITA   MOV  R11,R3            ; SAVE RETURN
032A 06A0 0302    WAIT10  BL   @RANDOM           ; GEN. NO.
032E 1F00               TB   TTY               ; WAIT FOR OP. INPUT
0330 13FC               JEQ  WAIT10
0332 2C44               IN   R4                ; GET INPUT
0334 0453               B    *R3
                  *
                  * ROUTINE: GETWG
                  * GET WAGER
                  *
0336 C0CB         GETWG   MOV  R11,R3            ; SAVE RETURN
0338 069F         GETW10  BL   *R15              ; ASK FOR INPUT
033A 0073               DATA MESS04
033C 04C1               CLR  R1                ; CLEAR TOTAL
033E 2C44         GETW20  IN   R4                ; GET INPUT
0340 0984               SRL  R4,8              ; RIGHT JUSTIFY
0342 0224 FFD0          AI   R4,->30           ; REMOVE ASCII BIAS
0346 1108               JLT  GETW30
0348 0284 0009          CI   R4,9
034C 1505               JGT  GETW30
034E 3860 001C          MPY  @TEN,R1           ; (R1,R2)=R1*10
0352 A084               A    R4,R2             ; R2=NEW VALUE
0354 C042               MOV  R2,R1
0356 10F3               JMP  GETW20
0358 0281 0064    GETW30  CI   R1,100            ; TEST SIZE
035C 1B06               JH   GETW40            ; TOO BIG
```

```
035E 8801 00B2          C     R1,@BROLL         ; CHECK AGAINST ASSETS
0362 1506               JGT   GETW50
0364 C801 00B4          MOV   R1,@WAGER         ; SAVE IT
0368 0453               B     *R3               ; EXIT
036A 069F       GETW40  BL    *R15
036C 005F               DATA  MESS03
036E 10E4               JMP   GETW10
0370 069F       GETW50  BL    *R15              ; REFUSE CREDIT
0372 0108               DATA  MESS14
0374 10E1               JMP   GETW10
                *
                * ROUTINE: MESS
                * DISPLAY THE MESSAGE WHOSE ADDRESS
                * FOLLOWS THE CALL
                *
0376 C08B       MESS    MOV   R11,R2            ; SAVE RETURN
0378 06A0 0262          BL    @TYPEN            ; NEW LINE
037C C072               MOV   *R2+,R1           ; R1=MESSAGE ADDR
037E 2C91       MSS10   OUT   *R1               ; OUTPUT CHARACTER
0380 0581               INC   R1                ; ADVANCE TO NEXT
0382 9811 0008          CB    *R1,@ATSGN        ; END?
0386 16FB               JNE   MSS10             ; NO-CONTINUE
0388 0452               B     *R2               ; EXIT
                *
                * GAME-1 (BLACKJACK)
                *
038A 069F       BLK10   BL    *R15              ; SIGN-ON MESSAGE
038C 0035               DATA  MESS00
038E 069F               BL    *R15              ; GIVE OUT BANKROLL
0390 003F               DATA  MESS01
0392 06A0 0314          BL    @WAIT             ; WAIT FOR GO
0396 06A0 0336  BLK20   BL    @GETWG            ; GET WAGER
039A 0201 00B6          LI    R1,PTOT           ; CLEAR TOTALS
039E 04F1       BLK30   CLR   *R1+
03A0 0281 00BE          CI    R1,CNT
03A4 16FC               JNE   BLK30
03A6 06A0 0456          BL    @DLR              ; GET DEALER HOLD
03AA 06A0 0410          BL    @PLAY             ; GET PLAYERS TWO
03AE 06A0 0410          BL    @PLAY
03B2 06A0 0456          BL    @DLR              ; GET DEALER SHOW
03B6 069F       BLK40   BL    *R15              ; HIT?
03B8 007A               DATA  MESS05
03BA 2C44               IN    R4                ; GET INPUT
03BC 0984               SRL   R4,8              ; RIGHT JUSTIFY
03BE 0284 004E          CI    R4,'N'            ; NO?
03C2 1306               JEQ   BLK45
03C4 0284 0059          CI    R4,'Y'            ; IF NOT YES?, ASK AGAIN
03C8 16F6               JNE   BLK40
03CA 06A0 0410          BL    @PLAY             ; GET HIT
03CE 10F3               JMP   BLK40             ; ASK AGAIN
03D0 069F       BLK45   BL    *R15              ; SHOW HOLD CARD
03D2 007F               DATA  MESS06
03D4 2CA0 00C1          OUT   @CHLD+1           ; PRINT CARD
03D8 C060 00BA  BLK50   MOV   @CTOT,R1          ; IF CTOT<16 - HIT
03DC 0281 0010          CI    R1,16
03E0 1503               JGT   BLK70
03E2 06A0 0456  BLK60   BL    @DLR              ; DEALER HIT
03E6 10F8               JMP   BLK50
```

```
03E8 0281 0016  BLK70   CI    R1,22             ; IF CTOT>21 - BUST
03EC 1105               JLT   BLK100
03EE 069F               BL    *R15              ; DEALER BUST
03F0 008D               DATA  MESS07
03F2 06A0 026C  BLK80   BL    @WIN              ; A WINNER
03F6 10CF               JMP   BLK20             ; CONTINUE
03F8 8060 00B6  BLK100  C     @PTOT,R1          ; COMPARE SCORES
03FC 15FA               JGT   BLK80             ; TRY AGAIN
03FE 069F               BL    *R15              ; DEALER TOTAL
0400 00B6               DATA  MESS10
0402 C120 00BA          MOV   @CTOT,R4          ; SHOW TOTAL
0406 06A0 02C2          BL    @DISPD
040A 06A0 0278  BLK110  BL    @LOSE             ; A LOSER
040E 10C3               JMP   BLK20             ; CONTINUE
                *
                * ROUTINE: PLAY
                * GET A CARD FOR PLAYER
                * ADJUST SCORE ACCORDING TO CARDS HELD
                * CHECK FOR BUST
                *
0410 C0CB       PLAY    MOV   R11,R3            ; SAVE RETURN
0412 069F               BL    *R15              ; DRAW-
0414 0119               DATA  MESS15
0416 06A0 0488          BL    @GET              ; GET CARD
041A A801 00B6          A     R1,@PTOT          ; ADD TO TOTAL
041E A802 00B8          A     R2,@PACE          ; ADD UP ACES TOO
0422 0201 0015  PLAY10  LI    R1,21             ; TEST SCORE
0426 8060 00B6          C     @PTOT,R1
042A 1508               JGT   PLAY20            ; BUST (MAYBE)
042C 1311               JEQ   PLAY40            ; BLACKJACK
042E 069F               BL    *R15              ; SCORE 1-20
0430 0125               DATA  MESS16
0432 C120 00B6          MOV   @PTOT,R4          ; PRINT TOTAL
0436 C2C3               MOV   R3,R11            ; SETUP FOR CALL TO DISP
0438 0460 02C2          B     @DISPD            ; CALL AND EXIT
043C 0620 00B8  PLAY20  DEC   @PACE             ; IF NO ACES-BUST
0440 1104               JLT   PLAY30
0442 6820 001C          S     @TEN,@PTOT        ; REDUCE ACE FROM 11 TO 1
0446 00B6
0448 10EC               JMP   PLAY10            ; RETEST
044A 069F       PLAY30  BL    *R15              ; BUST
044C 0136               DATA  MESS17
044E 10DD               JMP   BLK110            ; A LOSER
0450 069F       PLAY40  BL    *R15              ; BLACKJACK
0452 0141               DATA  MESS18
0454 10CE               JMP   BLK80
                *
                * ROUTINE: DLR
                * DRAW ONE FOR THE DEALER
                * ADJUST TOTAL SCORE
                *
0456 C0CB       DLR     MOV   R11,R3            ; SAVE RETURN
0458 C020 00BA          MOV   @CTOT,R0          ; IF FIRST CALL,
045C 1302               JEQ   DLR5              ;   DON'T SHOW DRAW
045E 069F               BL    *R15              ; DEALER DRAWS
0460 014C               DATA  MESS19
0462 06A0 0488  DLR5    BL    @GET              ; GET CARD
0466 A801 00BA          A     R1,@CTOT          ; UPDATE TOTAL
```

```
046A A802 00BC           A     R2,@CACE          ; UPDATE ACES TOO
046E 0201 0015  DLR10    LI    R1,21             ; TEST SCORE
0472 8060 00BA           C     @CTOT,R1
0476 1501                JGT   DLR30             ; BUST (MAYBE)
0478 0453       DLR20    B     *R3               ; EXIT WITH NEW SCORE
047A 0620 00BC  DLR30    DEC   @CACE             ; IF ACES, REDUCE SCORE
047E 11FC                JLT   DLR20             ; IF NOT, EXIT
0480 6820 001C           S     @TEN,@CTOT
0484 00BA
0486 10F3                JMP   DLR10             ; RETEST
                 *
                 * ROUTINE: GET
                 * GET ONE CARD.  IF NOT FIRST CALL
                 * PRINT IT ALSO
                 *
0488 C34B        GET     MOV   R11,R13           ; SAVE RETURN
048A 0620 00BE           DEC   @CNT              ; CHECK CARDS LEFT
048E 150A                JGT   GET10             ; IF NONE, RESHUFFLE
0490 0201 00C4           LI    R1,DECK           ; CLEAR DECK COUNT
0494 04F1        GET5    CLR   *R1+              ; CLEAR COUNTERS
0496 0281 00D2           CI    R1,DECK+14
049A 11FC                JLT   GET5
049C 0201 0033           LI    R1,51             ; RESET COUNT
04A0 C801 00BE           MOV   R1,@CNT
04A4 06A0 0302  GET10    BL    @RANDOM           ; GET RANDOM NO.
04A8 04C0                CLR   R0                ; FORCE PROPER RANGE
04AA 3C20 001E           DIV   @C13,R0
04AE D021 00C4           MOVB  @DECK(R1),R0      ; ANY LEFT?
04B2 C801 00C2           MOV   R1,@DRW           ; SAVE DRAWN CARD
04B6 0980                SRL   R0,8
04B8 0280 0003           CI    R0,3
04BC 15F3                JGT   GET10             ; NO-RETRY
04BE B860 0034           AB    @ONE,@DECK(R1)    ; UPDATE CARD COUNT
04C2 00C4
04C4 D121 000E           MOVB  @LABEL(R1),R4
04C8 0581                INC   R1                ; ADJUST FOR J,Q,K
04CA 0281 000A           CI    R1,10
04CE 1102                JLT   GET15
04D0 0201 000A           LI    R1,10
04D4 04C2        GET15   CLR   R2                ; CHECK FOR ACE
04D6 0984                SRL   R4,8
04D8 0284 0041           CI    R4,'A'
04DC 1603                JNE   GET20
04DE 0582                INC   R2                ; FLAG AS ACE
04E0 0201 000B           LI    R1,11             ; CHANGE VALUE
04E4 C020 00BA  GET20    MOV   @CTOT,R0          ; PRINT IF NOT FIRST
04E8 1303                JEQ   GET30
04EA 06C4                SWPB  R4                ; OUTPUT
04EC 2C84                OUT   R4
04EE 045D                B     *R13              ; EXIT
04F0 C804 00C0  GET30    MOV   R4,@CHLD          ; SAVE HOLD CARD
04F4 045D                B     *R13              ; EXIT
                 *
                 * GAME - 2 (FOUR DIGIT GUESS)
                 *
04F6 04E0 00B8  GS00     CLR   @GAMES            ; CLEAR GAME TOTAL
04FA 04E0 00B6           CLR   @GUESS            ; CLEAR GUESS TOTAL
04FE 069F        GS05    BL    *R15              ; SIGN-ON
```

```
0500 015C                DATA MESS20
0502 06A0 0314           BL   @WAIT              ; WAIT FOR START
0506 05A0 00B8           INC  @GAMES             ; UPDATE NO. OF GAMES
050A 04C2                CLR  R2                ; GENERATE NUMBER
050C 06A0 0302  GS10     BL   @RANDOM           ; GET NO.
0510 04C0                CLR  R0                ; FORCE RANGE 0-9
0512 3C20 001C           DIV  @TEN,R0
0516 C881 00C4           MOV  R1,@NO(R2)        ; SAVE NO
051A C0C2                MOV  R2,R3             ; CHECK FOR DUP
051C 0643        GS20    DECT R3
051E 1104                JLT  GS30
0520 88C1 00C4           C    R1,@NO(R3)
0524 13F3                JEQ  GS10              ; DUPLICATE
0526 10FA                JMP  GS20
0528 05C2        GS30    INCT R2                ; DIGIT O.K.
052A 0282 0008           CI   R2,8              ; CONTINUE TILL ALL DONE
052E 16EE                JNE  GS10
0530 0205 0001           LI   R5,1              ; R1=GUESS COUNT
0534 04CD        GS35    CLR  R13               ; R6=CORRECT POSITION
0536 04C7                CLR  R7                ; R7=JUST CORRECT
0538 05A0 00B6           INC  @GUESS            ; INC. TOTAL GUESS
053C 069F        GS40    BL   *R15              ; ASK FOR GUESS
053E 016D                DATA MESS21
0540 C105                MOV  R5,R4             ; PRINT GUESS NO.
0542 06A0 02BC           BL   @TYPED
0546 2CA0 0007           OUT  @QUEST            ;.PRINT '?'
054A 04C2                CLR  R2                ; GET FOUR DIGIT GUESS
054C 2C44        GS50    IN   R4                ; GET INPUT
054E 0984                SRL  R4,8              ; RIGHT JUSTIFY
0550 0224 FFD0           AI   R4,->30           ; REMOVE ASCII BIAS
0554 1130                JLT  GS100             ; NOT A DIGIT
0556 0284 000A           CI   R4,10             ; CHECK RANGE
055A 152D                JGT  GS100             ; AGAIN NOT A DIGIT
055C 8884 00C4           C    R4,@NO(R2)        ; CORRECT POS?
0560 1601                JNE  GS60
0562 058D                INC  R13
0564 0203 0008  GS60     LI   R3,8              ; CORRECT?
0568 88C4 00C2  GS70     C    R4,@NO-2(R3)
056C 1601                JNE  GS80
056E 0587                INC  R7
0570 0643        GS80    DECT R3
0572 16FA                JNE  GS70
0574 05C2                INCT R2                ; FINISHED GUESSING?
0576 0282 0008           CI   R2,8
057A 16E8                JNE  GS50              ; IF NOT, CONTINUE
057C 028D 0004           CI   R13,4             ; A WINNER?
0580 131D                JEQ  GS110
0582 069F                BL   *R15              ; SHOW RESULTS
0584 0178                DATA MESS22
0586 C107                MOV  R7,R4             ; DISPLAY TOTAL
0588 06A0 02BC           BL   @TYPED
058C 069F                BL   *R15
058E 018A                DATA MESS23
0590 C10D                MOV  R13,R4            ; DISPLAY SECOND TOTAL
0592 06A0 02BC           BL   @TYPED
0596 0585                INC  R5                ; UPDATE GUESS COUNT
0598 0285 0010           CI   R5,16             ; BUST?
059C 11CB                JLT  GS35
```

```
059E 069F              BL   *R15              ; A LOSER
05A0 019C              DATA MESS24
05A2 04C2              CLR  R2               ; SHOW ANSWER
05A4 C122 00C4 GS90    MOV  @NO(R2),R4
05A8 06A0 02BC         BL   @TYPED
05AC 05C2              INCT R2
05AE 0282 0008         CI   R2,8
05B2 11F8              JLT  GS90
05B4 1005              JMP  GS120            ; START OVER
05B6 069F      GS100   BL   *R15             ; ILLEGAL ENTRY
05B8 01BC              DATA MESS25
05BA 10C0              JMP  GS40
05BC 069F      GS110   BL   *R15             ; A WINNER
05BE 01CE              DATA MESS26
05C0 069F      GS120   BL   *R15             ; DO BATTING AVERAGE
05C2 01D8              DATA MESS27
05C4 C160 00B6         MOV  @GUESS,R5        ; R5=GUESS
05C8 3960 001C         MPY  @TEN,R5          ; (R5,R6)=GUESS*10
05CC 3D60 00B8         DIV  @GAMES,R5        ; R5=(GUESS/GAMES)*10
05D0 04C4              CLR  R4               ; (R4,R5)=(GUESS/GAMES)*10
05D2 3D20 001C         DIV  @TEN,R4          ; R4=GUESS/GAMES, R5=REMAINDER
05D6 06A0 02BC         BL   @TYPED           ; PRINT R4
05DA 2CA0 0005         OUT  @DECML           ; PRINT '.'
05DE 0225 0030         AI   R5,'0'
05E2 06C5              SWPB R5
05E4 2C85              OUT  R5               ;'PRINT DIGIT
05E6 108B              JMP  GS05
                *
                * GAME-3 (CRAPS)
                *
05E8 069F      CRP10   BL   *R15             ; SIGN ON
05EA 022B              DATA MESS40
05EC 069F              BL   *R15             ; GIVE OUT MONEY
05EE 003F              DATA MESS01
05F0 06A0 0314         BL   @WAIT            ; READY?
05F4 06A0 0336 CRP20   BL   @GETWG           ; GET WAGER
05F8 04E0 00B6         CLR  @POINT           ; CLEAR POINT
05FC 069F      CRP30   BL   *R15             ; ROLL
05FE 0231              DATA MESS41
0600 06A0 0262         BL   @TYPEN           ; NEW LINE
0604 06A0 0328         BL   @WAITA           ; WAIT TO GO
0608 3860 0020         MPY  @C6,R1
060C C101              MOV  R1,R4            ; R4=DIG ONE
060E C042              MOV  R2,R1
0610 0A31              SLA  R1,3             ; RANDOMIZE
0612 3860 0020         MPY  @C6,R1
0616 0581              INC  R1               ; FORCE RANGE 1-6
0618 0584              INC  R4
061A C004              MOV  R4,R0            ; CALC TOTAL
061C A001              A    R1,R0
061E C800 00B8         MOV  R0,@ROLL         ; SAVE IT
0622 0A81              SLA  R1,8
0624 A101              A    R1,R4
0626 06A0 02CC         BL   @TYPE2
062A C020 00B6         MOV  @POINT,R0
062E 1617              JNE  CRP50            ; JUMP IF NOT FIRST
0630 C020 00B8         MOV  @ROLL,R0
0634 0280 0007         CI   R0,7             ; 7=WINNER
```

```
0638 131C                JEQ   CRP70
063A 0280 000B           CI    R0,11              ; 11=WINNER
063E 1319                JEQ   CRP70
0640 0280 0004           CI    R0,4               ; 2,3=LOSER
0644 1113                JLT   CRP60
0646 0280 000C           CI    R0,12              ; 12=LOSER
064A 1310                JEQ   CRP60
064C 069F                BL    *R15               ; SHOW POINT
064E 023C                DATA  MESS42
0650 C120 00B8           MOV   @ROLL,R4
0654 C804 00B6           MOV   R4,@POINT
0658 06A0 02C2           BL    @DISPD
065C 10CF                JMP   CRP30              ; CONTINUE
065E C060 00B8  CRP50    MOV   @ROLL,R1           ; CHECK POINT
0662 8040                C     R0,R1
0664 1306                JEQ   CRP70              ; YES, WINNER
0666 0281 0007           CI    R1,7               ; 7=LOSER
066A 16C8                JNE   CRP30
066C 06A0 0278  CRP60    BL    @LOSE              ; A LOSER
0670 10C1                JMP   CRP20              ; CONTINUE
0672 06A0 026C  CRP70    BL    @WIN               ; A WINNER
0676 10BE                JMP   CRP20
                *
                * GAME-4 (ACEY DUECEY)
                *
0678 069F        AD10    BL    *R15               ; SIGN ON
067A 024A                DATA  MESS50
067C 069F                BL    *R15               ; GIVE OUT MONEY
067E 003F                DATA  MESS01
0680 06A0 0314           BL    @WAIT              ; WAIT TO GO
0684 069F        AD20    BL    *R15               ; LABEL THE PAIR
0686 0256                DATA  MESS51
0688 0720 00BA           SETO  @CTOT              ; SET DISP. FLAG
068C 06A0 0488           BL    @GET               ; GET ONE
0690 C820 00C2           MOV   @DRW,@NUM1
0694 00B6
0696 1603                JNE   AD21               ; SKIP IF NOT ACE
0698 C820 06AE           MOV   @THRT,@NUM1        ; NUM1=13
069C 00B6
069E 2CA0 0006   AD21    OUT   @SPACE             ; OUTPUT SPACE
06A2 06A0 0488           BL    @GET               ; GET TWO
06A6 C060 00C2           MOV   @DRW,R1            ; R1=DRAW
06AA 1602                JNE   AD22               ; JUMP IF NOT ACE
06AE             THRT    EQU   $+2                ; ADDRESS OF 13
06AC 0201 000D           LI    R1,13              ; RESET AS 13
06B0 8801 00B6   AD22    C     R1,@NUM1
06B4 1506                JGT   AD30
06B6 C820 00B6           MOV   @NUM1,@NUM2
06BA 00B8
06BC C801 00B6           MOV   R1,@NUM1
06C0 1002                JMP   AD40
06C2 C801 00B8   AD30    MOV   R1,@NUM2
06C6 06A0 0336   AD40    BL    @GETWG             ; WAGER
06CA C020 00B4           MOV   @WAGER,R0          ; IF ZERO, NO BET
06CE 13DA                JEQ   AD20
06D0 069F                BL    *R15               ; DRAW
06D2 0119                DATA  MESS15
06D4 06A0 0488           BL    @GET               ; GET CARD
```

```
06D8 8820 00C2          C     @DRW,@NUM1
06DC 00B6
06DE 1503               JGT   AD60              ; O.K. SO FAR
06E0 06A0 0278  AD50    BL    @LOSE             ; LOSER
06E4 10CF               JMP   AD20
06E6 8820 00B8  AD60    C     @NUM2,@DRW
06EA 00C2
06EC 12F9               JLE   AD50
06EE 06A0 026C          BL    @WIN              ; A WINNER
06F2 10C8               JMP   AD20
                *
                * CONTROL LOOP
                *
06F4 02E0 0080  BEGIN   LWPI  >80               ; USE MONITOR WORKSPACE
06F8 020F 0376          LI    R15,MESS          ; PRESET R15
06FC 04CC               CLR   R12               ; PRESET CRU BASE
06FE 069F               BL    *R15              ; SHOW CHOICES
0700 01EB               DATA  MESS30
0702 069F               BL    *R15
0704 0035               DATA  MESS00
0706 069F               BL    *R15
0708 015C               DATA  MESS20
070A 069F               BL    *R15
070C 022B               DATA  MESS40
070E 069F               BL    *R15
0710 024A               DATA  MESS50
0712 069F       BEG10   BL    *R15
0714 0208               DATA  MESS31
0716 2C44               IN    R4                ; GET CHARACTER
0718 0984               SRL   R4,8              ; RIGHT JUSTIFY
071A 0201 0022          LI    R1,GTAB
071E C0B1       BEG20   MOV   *R1+,R2
0720 13F8               JEQ   BEG10             ; NO MORE
0722 8C44               C     R4,*R1+
0724 16FC               JNE   BEG20
0726 0452               B     *R2               ; GO TO CHOICE
0728                    END   BEGIN             ; END OF SYSTEM
```

```
0678 AD10     0684 AD20     069E AD21     06B0 AD22     06C2 AD30
06C6 AD40     06E0 AD50     06E6 AD60     0008 ATSGN    000C BANK
0712 BEG10    071E BEG20    06F4 BEGIN    0004 BELLS    038A BLK10
03F8 BLK100   040A BLK110   0396 BLK20    039E BLK30    03B6 BLK40
03D0 BLK45    03D8 BLK50   *03E2 BLK60    03E8 BLK70    03F2 BLK80
00B2 BROLL    001E C13      0020 C6       00BC CACE     00C0 CHLD
00BE CNT      0009 CR       05E8 CRP10    05F4 CRP20    05FC CRP30
065E CRP50    066C CRP60    0672 CRP70    00BA CTOT     00C4 DECK
0005 DECML   *02DC DISP     02E2 DISP10   02DE DISPA    02C2 DISPD
0456 DLR      046E DLR10    0478 DLR20    047A DLR30    0462 DLR5
00C2 DRW      00B8 GAMES    0300 GEN      0488 GET      04A4 GET10
04D4 GET15    04E4 GET20    04F0 GET30    0494 GET5     0338 GETW10
033E GETW20   0358 GETW30   036A GETW40   0370 GETW50   0336 GETWG
04F6 GS00     04FE GS05     050C GS10     05B6 GS100    05BC GS110
05C0 GS120    051C GS20     0528 GS30     0534 GS35     053C GS40
054C GS50     0564 GS60     0568 GS70     0570 GS80     05A4 GS90
0022 GTAB     00B6 GUESS   *0000 IDTSSG   000E LABEL    000A LF
```

```
 0278 LOSE     0376 MESS     0035 MESS00   003F MESS01   0058 MESS02
 005F MESS03   0073 MESS04   007A MESS05   007F MESS06   008D MESS07
 009B MESS08   00A3 MESS09   00B6 MESS10   00C9 MESS11   00D2 MESS12
 00ED MESS13   0108 MESS14   0119 MESS15   0125 MESS16   0136 MESS17
 0141 MESS18   014C MESS19   015C MESS20   016D MESS21   0178 MESS22
 018A MESS23   019C MESS24   01BC MESS25   01CE MESS26   01D8 MESS27
 01EB MESS30   0208 MESS31   022B MESS40   0231 MESS41   023C MESS42
 024A MESS50   0256 MESS51   037E MSS10    00C4 NO       00B6 NUM1
 00B8 NUM2     0034 ONE      00B8 PACE     0410 PLAY     0422 PLAY10
 043C PLAY20   044A PLAY30   0450 PLAY40   00B6 POINT    00B6 PTOT
 0007 QUEST    0000 R0       0001 R1      *000A R10      000B R11
 000C R12      000D R13     *000E R14      000F R15      0002 R2
 0003 R3       0004 R4       0005 R5     *0006 R6       0007 R7
 0008 R8      *0009 R9       030A RAND10   0302 RANDOM   00B8 ROLL
 00B0 SEED     0284 SHOW     02A2 SHOW10   02A8 SHOW20   02B0 SHOW30
 02B8 SHOW40   0006 SPACE    001C TEN      06AE THRT     0000 TTY
 02CC TYPE2    02D2 TYPE3    02BC TYPED    0262 TYPEN    00B4 WAGER
 0314 WAIT     032A WAIT10   0328 WAITA    026C WIN
EDIT/ASM/LOAD?
```

SUPER STARTER GAMES
LOAD INTO RAM AT INDICATED ADDRESSES
BEGIN EXECUTION BY "G D2"

```
00D2: 04 60 07 C6 07 2E 20 3F 40 0D 0A 00 03 E8
00E0: 41 32 33 34 35 36 37 38 39 54 4A 51 4B 00 00 0A
00F0: 00 0D 00 06 04 5C 00 42 05 C8 00 46 06 BA 00 43
0100: 07 4A 00 41 00 00 01 42 4C 41 43 4B 4A 41 43 4B
0110: 40 49 4E 49 54 49 41 4C 20 42 41 4E 4B 52 4F 4C
0120: 4C 20 49 53 20 24 32 30 30 40 52 45 41 44 59 3F
0130: 40 48 4F 55 53 45 20 4C 49 4D 49 54 20 49 53 20
0140: 24 31 30 30 40 57 41 47 45 52 3F 40 48 49 54 3F
0150: 40 44 45 41 4C 45 52 20 48 4F 4C 44 53 20 40 44
0160: 45 41 4C 45 52 20 42 55 53 54 45 44 40 59 4F 55
0170: 20 57 49 4E 40 59 4F 55 52 20 42 41 4E 4B 52 4F
0180: 4C 4C 20 49 53 20 24 40 44 45 41 4C 45 52 20 54
0190: 4F 54 41 4C 20 49 53 20 2D 20 40 59 4F 55 20 4C
01A0: 4F 53 45 40 47 41 4D 45 20 4F 56 45 52 20 2D 20
01B0: 59 4F 55 20 41 52 45 20 42 52 4F 4B 45 21 40 21
01C0: 21 21 20 59 4F 55 20 42 52 4F 4B 45 20 54 48 45
01D0: 20 42 41 4E 4B 20 21 21 21 40 53 4F 52 52 59 2C
01E0: 20 4E 4F 20 43 52 45 44 49 54 40 59 4F 55 20 44
01F0: 52 41 57 20 2D 20 40 59 4F 55 52 20 54 4F 54 41
0200: 4C 20 49 53 20 2D 20 40 59 4F 55 20 42 55 53 54
0210: 45 44 40 42 4C 41 43 4B 4A 41 43 4B 21 40 44 45
0220: 41 4C 45 52 20 44 52 41 57 53 20 2D 20 40 46 4F
0230: 55 52 20 44 49 47 49 54 20 47 55 45 53 53 40 47
0240: 55 45 53 53 20 4E 4F 2E 20 40 44 49 47 49 54 53
0250: 20 43 4F 52 52 45 43 54 20 2D 20 40 49 4E 20 43
0260: 4F 52 52 45 43 54 20 50 4F 53 2E 2D 20 40 4C 41
0270: 53 54 20 53 48 4F 54 2C 20 59 4F 55 20 4C 4F 53
0280: 45 2E 20 20 49 54 20 57 41 53 20 2D 20 40 3F 3F
0290: 3F 20 4E 55 4D 42 45 52 53 20 4F 4E 4C 59 21 40
02A0: 54 48 41 54 53 20 49 54 21 40 59 4F 55 52 20 41
02B0: 56 45 52 41 47 45 20 49 53 20 2D 20 40 57 45 4C
02C0: 43 4F 4D 45 20 54 4F 20 54 48 45 20 53 2E 53 2E
02D0: 20 47 41 4D 45 52 4F 4F 4D 40 43 48 4F 4F 53 45
02E0: 20 59 4F 55 52 20 47 41 4D 45 20 28 42 59 20 46
02F0: 49 52 53 54 20 4C 45 54 54 45 52 29 40 43 52 41
0300: 50 53 40 52 4F 4C 4C 20 2E 2E 2E 2E 2E 40 59 4F
0310: 55 52 20 50 4F 49 4E 54 20 2D 20 40 41 43 45 59
0320: 20 44 55 45 43 45 59 40 54 48 45 20 50 41 49 52
0330: 20 2D 20 40 2C A0 00 DB 2C A0 00 DC 04 5B C0 CB
0340: 06 9F 01 6D C0 20 00 B4 10 06 C0 CB 06 9F 01 9B
0350: C0 20 00 B4 05 00 A0 20 00 B2 11 0C 13 0B 88 00
0360: 00 DE 15 0B C8 00 00 B2 06 9F 01 75 C1 60 00 B2
0370: C2 03 10 1E 06 9F 01 A4 10 08 06 9F 01 BF 02 01
0380: 00 14 2C A0 00 D6 06 01 16 FC 04 60 07 C6 02 84
0390: 00 0A 11 08 04 C3 3C E0 00 EE 0A 83 A1 03 02 24
03A0: 30 00 2C 84 02 24 00 30 06 C4 2C 84 04 5B C2 0B
03B0: 02 01 03 E8 C1 05 04 C3 3C C1 C1 03 04 C3 3C E0
03C0: 00 EE 06 A0 03 8E 04 C0 3C 20 00 EE C0 40 16 F2
03D0: 04 58 00 29 C0 20 00 B0 16 01 05 C0 38 20 03 D2
03E0: C8 01 00 B0 04 5B C0 CB 06 9F 01 2A 02 01 00 C8
03F0: C8 01 00 B2 04 E0 00 BE C2 C3 C0 CB 06 A0 03 D4
```

```
0400: 1F 00 13 FC 2C 44 04 53 C0 CB 06 9F 01 45 04 C1
0410: 2C 44 09 84 02 24 FF D0 11 08 02 84 00 09 15 05
0420: 38 60 00 EE A0 84 C0 42 10 F3 02 81 00 64 1B 06
0430: 88 01 00 B2 15 06 C8 01 00 B4 04 53 06 9F 01 31
0440: 10 E4 06 9F 01 DA 10 E1 C0 8B 06 A0 03 34 C0 72
0450: 2C 91 05 81 98 11 00 DA 16 FB 04 52 06 9F 01 07
0460: 06 9F 01 11 06 A0 03 E6 06 A0 04 08 02 01 00 B6
0470: 04 F1 02 81 00 BE 16 FC 06 A0 05 28 06 A0 04 E2
0480: 06 A0 04 E2 06 A0 05 28 06 9F 01 4C 2C 44 09 84
0490: 02 84 00 4E 13 06 02 84 00 59 16 F6 06 A0 04 E2
04A0: 10 F3 06 9F 01 51 2C A0 00 C1 C0 60 00 BA 02 81
04B0: 00 10 15 03 06 A0 05 28 10 F8 02 81 00 16 11 05
04C0: 06 9F 01 5F 06 A0 03 3E 10 CF 80 60 00 B6 15 FA
04D0: 06 9F 01 88 C1 20 00 BA 06 A0 03 94 06 A0 03 4A
04E0: 10 C3 C0 CB 06 9F 01 EB 06 A0 05 5A A8 01 00 B6
04F0: A8 02 00 B8 02 01 00 15 80 60 00 B6 15 08 13 11
0500: 06 9F 01 F7 C1 20 00 B6 C2 C3 04 60 03 94 06 20
0510: 00 B8 11 04 68 20 00 EE 00 B6 10 EC 06 9F 02 08
0520: 10 DD 06 9F 02 13 10 CE C0 CB C0 20 00 BA 13 02
0530: 06 9F 02 1E 06 A0 05 5A A8 01 00 BA A8 02 00 BC
0540: 02 01 00 15 80 60 00 BA 15 01 04 53 06 20 00 BC
0550: 11 FC 68 20 00 EE 00 BA 10 F3 C3 4B 06 20 00 BE
0560: 15 0A 02 01 00 C4 04 F1 02 81 00 D2 11 FC 02 01
0570: 00 33 C8 01 00 BE 06 A0 03 D4 04 C0 3C 20 00 F0
0580: D0 21 00 C4 C8 01 00 C2 09 80 02 80 00 03 15 F3
0590: B8 60 01 06 00 C4 D1 21 00 E0 05 81 02 81 00 0A
05A0: 11 02 02 01 00 0A 04 C2 09 84 02 84 00 41 16 03
05B0: 05 82 02 01 00 0B C0 20 00 BA 13 03 06 C4 2C 84
05C0: 04 5D C8 04 00 C0 04 5D 04 E0 00 B8 04 E0 00 B6
05D0: 06 9F 02 2E 06 A0 03 E6 05 A0 00 B8 04 C2 06 A0
05E0: 03 D4 04 C0 3C 20 00 EE C8 81 00 C4 C0 C2 06 43
05F0: 11 04 88 C1 00 C4 13 F3 10 FA 05 C2 02 82 00 08
0600: 16 EE 02 05 00 01 04 CD 04 C7 05 A0 00 B6 06 9F
0610: 02 3F C1 05 06 A0 03 8E 2C A0 00 D9 04 C2 2C 44
0620: 09 84 02 24 FF D0 11 30 02 84 00 0A 15 2D 88 84
0630: 00 C4 16 01 05 8D 02 03 00 08 88 C4 00 C2 16 01
0640: 05 87 06 43 16 FA 05 C2 02 82 00 08 16 E8 02 8D
0650: 00 04 13 1D 06 9F 02 4A C1 07 06 A0 03 8E 06 9F
0660: 02 5C C1 0D 06 A0 03 8E 05 85 02 85 00 10 11 CB
0670: 06 9F 02 6E 04 C2 C1 22 00 C4 06 A0 03 8E 05 C2
0680: 02 82 00 08 11 F8 10 05 06 9F 02 8E 10 C0 06 9F
0690: 02 A0 06 9F 02 AA C1 60 00 B6 39 60 00 EE 3D 60
06A0: 00 B8 04 C4 3D 20 00 EE 06 A0 03 8E 2C A0 00 D7
06B0: 02 25 00 30 06 C5 2C 85 10 8B 06 9F 02 FD 06 9F
06C0: 01 11 06 A0 03 E6 06 A0 04 08 04 E0 00 B6 06 9F
06D0: 03 03 06 A0 03 34 06 A0 03 FA 38 60 00 F2 C1 01
06E0: C0 42 0A 31 38 60 00 F2 05 81 05 84 C0 04 A0 01
06F0: C8 00 00 B8 0A 81 A1 01 06 A0 03 9E C0 20 00 B6
0700: 16 17 C0 20 00 B8 02 80 00 07 13 1C 02 80 00 0B
0710: 13 19 02 80 00 04 11 13 02 80 00 0C 13 10 06 9F
0720: 03 0E C1 20 00 B8 C8 04 00 B6 06 A0 03 94 10 CF
0730: C0 60 00 B8 80 40 13 06 02 81 00 07 16 C8 06 A0
0740: 03 4A 10 C1 06 A0 03 3E 10 BE 06 9F 03 1C 06 9F
0750: 01 11 06 A0 03 E6 06 9F 03 28 07 20 00 BA 06 A0
0760: 05 5A C8 20 00 C2 00 B6 16 03 C8 20 07 80 00 B6
0770: 2C A0 00 D8 06 A0 05 5A C0 60 00 C2 16 02 02 01
0780: 00 0D 88 01 00 B6 15 06 C8 20 00 B6 00 B8 C8 01
0790: 00 B6 10 02 C8 01 00 B8 06 A0 04 08 C0 20 00 B4
```

```
07A0: 13 DA 06 9F 01 EB 06 A0 05 5A 88 20 00 C2 00 B6
07B0: 15 03 06 A0 03 4A 10 CF 88 20 00 B8 00 C2 12 F9
07C0: 06 A0 03 3E 10 C8 02 E0 00 80 02 0F 04 48 04 CC
07D0: 06 9F 02 BD 06 9F 01 07 06 9F 02 2E 06 9F 02 FD
07E0: 06 9F 03 1C 06 9F 02 DA 2C 44 09 84 02 01 00 F4
07F0: C0 B1 13 F8 8C 44 16 FC 04 52 04 60 05 F0 1E 01
?
```

SUPER STARTER GAMES - PROM VERSION
LOAD PROGRAM INTO RAM AT >B0, THEN PROGRAM
INTO A PROM BY "P B0,7FF,0". PROGRAM IS
EXECUTED IN PROM BY "G F000".

```
00B0: 04 60 F6 F4 07 2E 20 3F 40 0D 0A 00 03 E8 41 32
00C0: 33 34 35 36 37 38 39 54 4A 51 4B 00 00 0A 00 0D
00D0: 00 06 F3 8A 00 42 F4 F6 00 46 F5 E8 00 43 F6 78
00E0: 00 41 00 00 01 42 4C 41 43 4B 4A 41 43 4B 40 49
00F0: 4E 49 54 49 41 4C 20 42 41 4E 4B 52 4F 4C 4C 20
0100: 49 53 20 24 32 30 30 40 52 45 41 44 59 3F 40 48
0110: 4F 55 53 45 20 4C 49 4D 49 54 20 49 53 20 24 31
0120: 30 30 40 57 41 47 45 52 3F 40 48 49 54 3F 40 44
0130: 45 41 4C 45 52 20 48 4F 4C 44 53 20 40 44 45 41
0140: 4C 45 52 20 42 55 53 54 45 44 40 59 4F 55 20 57
0150: 49 4E 40 59 4F 55 52 20 42 41 4E 4B 52 4F 4C 4C
0160: 20 49 53 20 24 40 44 45 41 4C 45 52 20 54 4F 54
0170: 41 4C 20 49 53 20 2D 20 40 59 4F 55 20 4C 4F 53
0180: 45 40 47 41 4D 45 20 4F 56 45 52 20 2D 20 59 4F
0190: 55 20 41 52 45 20 42 52 4F 4B 45 21 40 21 21 21
01A0: 20 59 4F 55 20 42 52 4F 4B 45 20 54 48 45 20 42
01B0: 41 4E 4B 20 21 21 21 40 53 4F 52 52 59 2C 20 4E
01C0: 4F 20 43 52 45 44 49 54 40 59 4F 55 20 44 52 41
01D0: 57 20 2D 20 40 59 4F 55 52 20 54 4F 54 41 4C 20
01E0: 49 53 20 2D 20 40 59 4F 55 20 42 55 53 54 45 44
01F0: 40 42 4C 41 43 4B 4A 41 43 4B 21 40 44 45 41 4C
0200: 45 52 20 44 52 41 57 53 20 2D 20 40 46 4F 55 52
0210: 20 44 49 47 49 54 20 47 55 45 53 53 40 47 55 45
0220: 53 53 20 4E 4F 2E 20 40 44 49 47 49 54 53 20 43
0230: 4F 52 52 45 43 54 20 2D 20 40 49 4E 20 43 4F 52
0240: 52 45 43 54 20 50 4F 53 2E 2D 20 40 4C 41 53 54
0250: 20 53 48 4F 54 2C 20 59 4F 55 20 4C 4F 53 45 2E
0260: 20 20 49 54 20 57 41 53 20 2D 20 40 3F 3F 3F 20
0270: 4E 55 4D 42 45 52 53 20 4F 4E 4C 59 21 40 54 48
0280: 41 54 53 20 49 54 21 40 59 4F 55 52 20 41 56 45
0290: 52 41 47 45 20 49 53 20 2D 20 40 57 45 4C 43 4F
02A0: 4D 45 20 54 4F 20 54 48 45 20 53 2E 53 2E 20 47
02B0: 41 4D 45 52 4F 4F 4D 40 43 48 4F 4F 53 45 20 59
02C0: 4F 55 52 20 47 41 4D 45 20 28 42 59 20 46 49 52
02D0: 53 54 20 4C 45 54 54 45 52 29 40 43 52 41 50 53
02E0: 40 52 4F 4C 4C 20 2E 2E 2E 2E 2E 40 59 4F 55 52
02F0: 20 50 4F 49 4E 54 20 2D 20 40 41 43 45 59 20 44
0300: 55 45 43 45 59 40 54 48 45 20 50 41 49 52 20 2D
0310: 20 40 2C A0 F0 09 2C A0 F0 0A 04 5B C0 CB 06 9F
0320: F0 9B C0 20 00 B4 10 06 C0 CB 06 9F F0 C9 C0 20
0330: 00 B4 05 00 A0 20 00 B2 11 0C 13 0B 88 00 F0 0C
0340: 15 0B C8 00 00 B2 06 9F F0 A3 C1 60 00 B2 C2 03
0350: 10 1E 06 9F F0 D2 10 08 06 9F F0 ED 02 01 00 14
0360: 2C A0 F0 04 06 01 16 FC 04 60 F6 F4 02 84 00 0A
0370: 11 08 04 C3 3C E0 F0 1C 0A 83 A1 03 02 24 30 00
0380: 2C 84 02 24 00 30 06 C4 2C 84 04 5B C2 0B 02 01
0390: 03 E8 C1 05 04 C3 3C C1 C1 03 04 C3 3C E0 F0 1C
03A0: 06 A0 F2 BC 04 C0 3C 20 F0 1C C0 40 16 F2 04 58
03B0: 00 29 C0 20 00 B0 16 01 05 C0 38 20 F3 00 C8 01
03C0: 00 B0 04 5B C0 CB 06 9F F0 58 02 01 00 C8 C8 01
03D0: 00 B2 04 E0 00 BE C2 C3 C0 CB 06 A0 F3 02 1F 00
03E0: 13 FC 2C 44 04 53 C0 CB 06 9F F0 73 04 C1 2C 44
03F0: 09 84 02 24 FF D0 11 08 02 84 00 09 15 05 38 60
?
```

```
0400: F0 1C A0 84 C0 42 10 F3 02 81 00 64 1B 06 88 01
0410: 00 B2 15 06 C8 01 00 B4 04 53 06 9F F0 5F 10 E4
0420: 06 9F F1 08 10 E1 C0 8B 06 A0 F2 62 C0 72 2C 91
0430: 05 81 98 11 F0 08 16 FB 04 52 06 9F F0 35 06 9F
0440: F0 3F 06 A0 F3 14 06 A0 F3 36 02 01 00 B6 04 F1
0450: 02 81 00 BE 16 FC 06 A0 F4 56 06 A0 F4 10 06 A0
0460: F4 10 06 A0 F4 56 06 9F F0 7A 2C 44 09 84 02 84
0470: 00 4E 13 06 02 84 00 59 16 F6 06 A0 F4 10 10 F3
0480: 06 9F F0 7F 2C A0 00 C1 C0 60 00 BA 02 81 00 10
0490: 15 03 06 A0 F4 56 10 F8 02 81 00 16 11 05 06 9F
04A0: F0 8D 06 A0 F2 6C 10 CF 80 60 00 B6 15 FA 06 9F
04B0: F0 B6 C1 20 00 BA 06 A0 F2 C2 06 A0 F2 78 10 C3
04C0: C0 CB 06 9F F1 19 06 A0 F4 88 A8 01 00 B6 A8 02
04D0: 00 B8 02 01 00 15 80 60 00 B6 15 08 13 11 06 9F
04E0: F1 25 C1 20 00 B6 C2 C3 04 60 F2 C2 06 20 00 B8
04F0: 11 04 68 20 F0 1C 00 B6 10 EC 06 9F F1 36 10 DD
0500: 06 9F F1 41 10 CE C0 CB C0 20 00 BA 13 02 06 9F
0510: F1 4C 06 A0 F4 88 A8 01 00 BA A8 02 00 BC 02 01
0520: 00 15 80 60 00 BA 15 01 04 53 06 20 00 BC 11 FC
0530: 68 20 F0 1C 00 BA 10 F3 C3 4B 06 20 00 BE 15 0A
0540: 02 01 00 C4 04 F1 02 81 00 D2 11 FC 02 01 00 33
0550: C8 01 00 BE 06 A0 F3 02 04 C0 3C 20 F0 1E D0 21
0560: 00 C4 C8 01 00 C2 09 80 02 80 00 03 15 F3 B8 60
0570: F0 34 00 C4 D1 21 F0 0E 05 81 02 81 00 0A 11 02
0580: 02 01 00 0A 04 C2 09 84 02 84 00 41 16 03 05 82
0590: 02 01 00 0B C0 20 00 BA 13 03 06 C4 2C 84 04 5D
05A0: C8 04 00 C0 04 5D 04 E0 00 B8 04 E0 00 B6 06 9F
05B0: F1 5C 06 A0 F3 14 05 A0 00 B8 04 C2 06 A0 F3 02
05C0: 04 C0 3C 20 F0 1C C8 81 00 C4 C0 C2 06 43 11 04
05D0: 88 C1 00 C4 13 F3 10 FA 05 C2 02 82 00 08 16 EE
05E0: 02 05 00 01 04 CD 04 C7 05 A0 00 B6 06 9F F1 6D
05F0: C1 05 06 A0 F2 BC 2C A0 F0 07 04 C2 2C 44 09 84
0600: 02 24 FF D0 11 30 02 84 00 0A 15 2D 88 84 00 C4
0610: 16 01 05 8D 02 03 00 08 88 C4 00 C2 16 01 05 87
0620: 06 43 16 FA 05 C2 02 82 00 08 16 E8 02 8D 00 04
0630: 13 1D 06 9F F1 78 C1 07 06 A0 F2 BC 06 9F F1 8A
0640: C1 0D 06 A0 F2 BC 05 85 02 85 00 10 11 CB 06 9F
0650: F1 9C 04 C2 C1 22 00 C4 06 A0 F2 BC 05 C2 02 82
0660: 00 08 11 F8 10 05 06 9F F1 BC 10 C0 06 9F F1 CE
0670: 06 9F F1 D8 C1 60 00 B6 39 60 F0 1C 3D 60 00 B8
0680: 04 C4 3D 20 F0 1C 06 A0 F2 BC 2C A0 F0 05 02 25
0690: 00 30 06 C5 2C 85 10 8B 06 9F F2 2B 06 9F F0 3F
06A0: 06 A0 F3 14 06 A0 F3 36 04 E0 00 B6 06 9F F2 31
06B0: 06 A0 F2 62 06 A0 F3 28 38 60 F0 20 C1 01 C0 42
06C0: 0A 31 38 60 F0 20 05 81 05 84 C0 04 A0 01 C8 00
06D0: 00 B8 0A 81 A1 01 06 A0 F2 CC C0 20 00 B6 16 17
06E0: C0 20 00 B8 02 80 00 07 13 1C 02 80 00 0B 13 19
06F0: 02 80 00 04 11 13 02 80 00 0C 13 10 06 9F F2 3C
0700: C1 20 00 B8 C8 04 00 B6 06 A0 F2 C2 10 CF C0 60
0710: 00 B8 80 40 13 06 02 81 00 07 16 C8 06 A0 F2 78
0720: 10 C1 06 A0 F2 6C 10 BE 06 9F F2 4A 06 9F F0 3F
0730: 06 A0 F3 14 06 9F F2 56 07 20 00 BA 06 A0 F4 88
0740: C8 20 00 C2 00 B6 16 03 C8 20 F6 AE 00 B6 2C A0
0750: F0 06 06 A0 F4 88 C0 60 00 C2 16 02 02 01 00 0D
0760: 88 01 00 B6 15 06 C8 20 00 B6 00 B8 C8 01 00 B6
0770: 10 02 C8 01 00 B8 06 A0 F3 36 C0 20 00 B4 13 DA
0780: 06 9F F1 19 06 A0 F4 88 88 20 00 C2 00 B6 15 03
```

```
0790: 06 A0 F2 78 10 CF 88 20 00 B8 00 C2 12 F9 06 A0
07A0: F2 6C 10 C8 02 E0 00 80 02 0F F3 76 04 CC 06 9F
07B0: F1 EB 06 9F F0 35 06 9F F1 5C 06 9F F2 2B 06 9F
07C0: F2 4A 06 9F F2 08 2C 44 09 84 02 01 F0 22 C0 B1
07D0: 13 F8 8C 44 16 FC 04 52 06 A0 14 CE 48 2B C1 41
07E0: 61 42 06 A0 14 CE 48 2D 04 60 14 E6 41 02 42 87
07F0: 43 88 44 06 47 83 48 84 49 02 4C 83 4D 84 50 88
?
```

The TMS-9900 MICROPROCESSOR: Used in Technico Systems

Unmatched. In word size. Instruction set. Addressing capabilities. TI's 16-bit TMS9900 microprocessor.

Powerful enough to be the heart of a minicomputer. Ideal for terminals. Instrumentation. Machine control. Scores of OEM applications. Destined to become today's and tomorrow's design standard.

Because the TMS9900 microprocessor represents more than just a single device. It introduces a new family concept allowing full design flexibility. Enabling you to move freely and easily over your entire range of applications. Now. And in the future. With less redesign. Less software reinvestment. Less relearning. Less obsolescence.

Improved System Cost/Performance Compared to 8-bit μPs, TI's TMS9900 microprocessor provides these unmatched savings:

- 30% faster execution time
- 50% savings in program coding
- 40% savings on system interface costs
- 50% more efficient interrupt handling
- 20% reduction in memory bit requirements

These benefits stem from the TMS9900's advanced features:

16-bit instruction word with full 16-bit data precision.

Operation at 3.3 MHz clock rate.

Full minicomputer instruction set including Hardware multiply and divide.

Advanced memory-to-memory architecture that locates general-purpose register files in memory.

Separate 16-bit address, data, I/O and interrupt buses.

Fully Compatible Software
The 9900/990 software has been tested and proven in more than 1000 systems. Any software you develop for the TMS9900 can be used with the 990 and 990/10 minicomputers — or the SBP9900 and TMS9980 microprocessors. In fact, any software developed for the TMS9900 can be used with any other family member — at present and in the years ahead.

More 9900 Family Soon
The TMS9900 is just the beginning. Future family circuits, all software compatible: SBP9900, an I^2L microprocessor designed to handle military temperature ranges. TMS9980, an N-channel μP with an 8-bit data bus for smaller systems.

Also coming are 9900 peripheral support circuits: TMS9901 programmable systems interface. TMS9902 asynchronous communication controller. TMS9903 synchronous communications controller. And the TIM9904 low power Schottky TTL 4-phase clock generator.

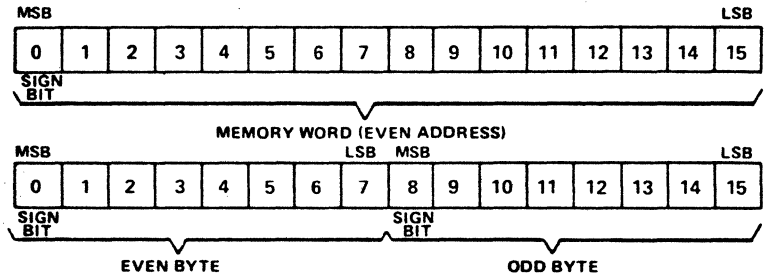
INSTRUCTION SET

The instruction set of the TMS9900 contains 69 commands, including multiplication and division, which may be divided into seven principal groups:

INSTRUCTION SET SUMMARY (69 INSTRUCTIONS)	
Arithmetic (16)	ADD (W, B, Imm), SUB (W, B), COMPARE (W, B, Imm), INCR (1,2), DECR (1,2), ABS, NEG, MPY, DIV
Program Control (20)	BRANCH (LINK, LOAD WP), JUMP, JUMP CONDITIONAL (12) RETURN, IDLE, EXECUTE, EXTENDED OPERATION
Data Control (14)	MOVE (W, B) LOAD (Imm, WP, ST), STORE (ST, WP), SWAP BYTES, CLR, SETO, SOC (W, B), SZC (W, B)
Logical (6)	ANDI, ORI, INV, COC, CZC, XOR
Shifts (4)	SRA, SRL, SRC, SLA
I/O (5)	LDCR, STCR, TB, SBO, SBZ
External (4)	RESET, CKON, CKOFF, LREX

ARCHITECTURE

The memory word of the TMS9900 is 16 bits long. Each word is also defined as **2 bytes of 8 bits.** The instruction set of the TMS9900 allows **both word and byte operands.** Thus, all memory locations are on even address boundaries, and byte instructions can address either the even or odd byte. The memory space is 65,536 bytes or 32,768 words. The word and byte formats are shown below.

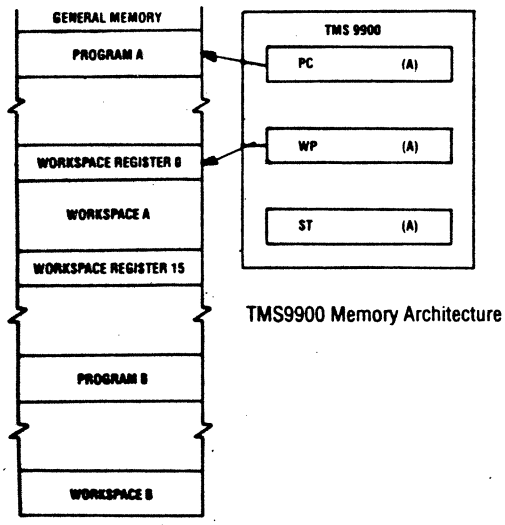


ADDRESSING MODE	DESCRIPTION
Workspace Register	The contents of the indicated workspace register are the operand.
Workspace Register Indirect	The contents of the indicated workspace register contain the memory address of the operand.
Indexed	The contents of the indicated workspace register are added to the address enclosed in the second command word.
Direct (S or D Equals 0)	The word following the instruction contains the memory address of the operand.
Workspace Register Indirect with Auto Increment	The contents of the indicated workspace register contain the memory address of the operand which is automatically incremented after the access (plus 2 for word operations and plus 1 for byte operations).
Immediate	The word following the instruction contains the operand.
Relative	The 8-bit displacement of the instruction is added to the update program counter in jump instructions or to the base address in single-bit CRU instructions.

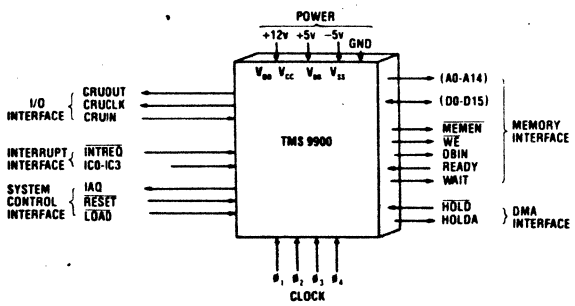
Addressing Modes
A program can use seven different modes of addressing.

	32 BITS	16 BITS				8 BITS			
COMPARE!	IBM 360	TECHNICO TI 9900	DEC LSI-11	DataGen NOVA	National PACE	Z1LOG Z-80	INTEL 8080	Materola 6800	MOS 6502
WORD SIZE (Register)	4 BYTES	2 BYTES	2 BYTES	2 BYTES	1 BYTE	1 BYTE	1 BYTE	1 BYTE	1 BYTE
NUMBER OF ACCUMULATORS	16	16	7	4	4	1	1	2	1
NUMBER OF SEPARATE SETS OF REGISTERS	1	MANY	1	1	1	2	1	1	1
ACCUMULATOR ARITHMETIC CAPABILITY	to 4.3 billion	to 65,535	to 65,535	to 65,535	to 65,535	to 255	to 255	to 255	to 255
NUMBER OF INDEX REGISTERS	15	15	7	2	2	2	0	1	2
MAXIMUM INDEX REG. ADDRESS VALUE	16,777,216	65,535	65,535	32,767	65,535	65,535	—	65,535	255
MAXIMUM DISPLACEMENT FROM INDEX REGISTER	+4096	+65,535, -65,535	+65,535, -65,535	+127, -128	+127, -128	+127, -128	—	+255	+65,535
MEMORY ADDRESSING LEVEL	BYTE	BYTE	BYTE	word	word	BYTE	BYTE	BYTE	BYTE
MEMORY TO MEMORY DATA MOVEMENT	YES	YES	YES	no	no	YES	no	no	no
HARDWARE MULTIPLY AND DIVIDE	YES	YES	optional	YES	no	no	no	no	no
SINGLE CHIP CENTRAL PROCESSING UNIT (CPU)	no	YES	no	YES	YES	YES	YES	YES	YES
SOFTWARE COMPATIBLE MINICOMPUTER FAMILY	no	YES	YES	YES	no	no	no	no	no
BIT ADDRESSABLE COMMUNICATIONS REG. UNIT	no	YES	no	no	no	no	no	no	no
EASY and INEXPENSIVE to INTERFACE to	no	YES	no	YES	YES	YES	YES	YES	YES

©TECHNICO



TMS9900 Memory Architecture



TMS9900 Microprocessor CPU

INTERRUPTS

The TMS9900 employs 16 interrupt levels with the highest priority level 0 and lowest level 15. Level 0 is reserved for the RESET function, and all other levels may be used for external devices. The external levels may also be shared by several device interrupts, depending upon system requirements.

The Program Counter (PC) contains the address of the next instruction to be executed. As each instruction is executed, the PC is automatically updated.

The Workspace Pointer (WP) contains the memory address of the first sixteen consecutive memory words in the workspace. Thus, the processor has access to sixteen 16-bit registers. When a different set of registers is required, the program simply reloads the Workspace Pointer with the address of the new workspace. This results in a significant reduction in processor overhead when a new set of registers is required.

The Status Register (SR) contains flag bits which indicate results of the most recent arithmetic or logical operations performed. The SR also contains the 4-bit interrupt mask level.

INPUT/OUTPUT

The TMS9900 can input and output data by three distinct methods. A dedicated method of performing I/O, utilizing a separate I/O port called the Communications Register Unit (CRU), may be preferred in a majority of applications because of its easy interfacing capabilities.

TECHNICO INCORPORATED
9130 RED BRANCH RD.
COLUMBIA, MD. 21045
PHONE 301-596-4100

COMPARE 16 BIT COMPUTERS

TECHNICO vs. HEATH

An attractive vacuum formed chassis may be added to any Technico System at a cost of $179; however, a modular design is used which does not require a mother board or chassis. Get a more powerful, more flexible system from Technico based on the Texas Instruments TMS-9900 16 Bit Microprocessor, and save $500 to $1,000.

TECHNICO SUPER SYSTEM 16

Includes serial and parallel interface, E-PROM programmer, CPU and Memory — can be programmed in Hex. No power supply or terminal. Hardware multiply divide is included. Order P/N TK-1.

Power Supply is added so that system is a functioning microcomputer. Only a terminal or keyboard and video board is needed for operation. Order P/N TK-1-PC.

The Instant Input Assembler can be added in ROM for only $49. Provides assembly language capability. Order P/N TK-1-IA.

Allows 12K Byte user area with assembler, editor, linking loader. Allows 10K Byte with Basic. Additional 8K Bytes of memory only $100. Order TK-2-18K.

A 4800 Baud digital cassette is used for program loading. It also provides 80,000 Bytes of memory storage. Storage time for an 8K program under 3 minutes. Reliable, fast storage on digital cassette. Price — $199. Order P/N T-9948-C.

Technico Chassis Capacity

Technico chassis will hold CPU plus 65K Byte of memory. Floppy disk controller, digital cassette interface and up to 7-RS232 or 20 ma current loop interfaces, 48 Bits of parallel input and output and a video color graphics board with Keyboard Interface. A 2708 and/or 2716 E-PROM programmer can also be included.

TWO BYTES ARE BETTER THAN ONE

COMPARE $ PRICE $

SAVE $$$

IN STORE PRICE	TECHNICO SYSTEM 16	HEATH* H-11
MINIMUM KIT	$299	$1,350
WITH POWER SUPPLY AND I-O	$442	$1,550
WITH ASSEMBLY LANGUAGE	$491	$1,845
WITH MEMORY FOR FULL SOFTWARE	$968	$2,140

*FOR COMPLETE COMPARISON SEE HEATH AND SEND FOR TECHNICO PRICE LIST

Circle Inquiry No.

TECHNICO INCORPORATED

9130 RED BRANCH RD.
COLUMBIA, MD. 21045
PHONE 301-596-4100

1-800-638-2893

HEATH H-11

Contains power supply and limited chassis but **no Interface** for terminal; there is no way to enter data or programs. CPU **does not** include hardware multiply divide. The H-11-6 costs an extra $159. (pg. 6 & 7*)

An H-11-2 parallel interface and H-11-5 serial interface cost $200. (pg. 7*)

An H-11-1 at $295 must be added to bring the system to the minimum memory to run any software. (pg. 6*)

Another H-11-1 at $295 must be added to bring the memory to the size recommended by Heath (page 6, 3rd column, last para.*) DOES NOT INCLUDE ANY TERMINAL DEVICE

The only method for program loading described in Heath catalog is a 50 character/sec paper tape reader H10. Punch operates at 10 characters per sec. Estimated time to punch an 8K program is over 25 minutes. COST — $370 (page 8*)

Heath Chassis Capacity. (pg. 6)

(Col. 1, para 3*, and Pg. 6, col. 1, para. 1*) With KD11F Board and H-11-2 and H-11-5 there is space for only 20K words (40K Byte) of memory **not** 65K Bytes. No space for additional RS232 interfaces. No space for floppy disk controller. No color graphics. No digital or audio cassett interface.

***ALL P-C BOARDS ARE AVAILABLE ASSEMBLED AND TESTED OR AS TEC-KITS™**
BUY TECHNICO PRODUCTS FROM YOUR LOCAL COMPUTER STORE!

***Source Heath Catalog Christmas 1977 issue**

TECHNICO
INCORPORATED

9130 RED BRANCH RD.
COLUMBIA, MD. 21045
PHONE 301-596-4100

CALL TOLL FREE 1-800-638-2893

TECHNICO PRODUCT DESCRIPTION

Technico products are **available assembled** and tested and in most cases, **unassembled** in TEC-KIT™ form. Due to Technico's previous experience in supplying hi-rel computer components to the aerospace and defense industry, only **full spec** manufacturer **warranteed parts** rated and guaranteed over the full temperature range are used in TEC-KITS.™ The PC board material is the same as that used in the aerospace industry. **All boards** are **socketed** for easy repair and servicing. Domestic prices range from **$299 to** over **$5000** and **all boards** are **compatible** to form whatever level system the user may desire or be able to afford. Any boards purchased as TEC-KITS™ and assembled by the user may be returned initially and **factory tested** and repaired **for** a **flat $25.00 fee.**

The Technico Super Starter System is a **low cost** start towards owning the **most powerful personal computer** (the Super Starter 16) on the market today. It is the most powerful because it **uses** the **16 bit TMS9900** microprocessor. It is **low cost because** of a **unique modular design** which allows operations either without a chassis or with a chassis which does not require an expensive mother board. The bus structure is universal and allows easy interface to other Technico boards or S100 or S50 boards. A **chassis** with fan switches cables and connectors **may be added** to the system **at any time.**

Unlike other competing personal computing systems, however, **a chassis is not required for operation.** In the case of the Imsai, for example, of the first $1000 spent on a system, almost 40% of the price goes for chassis and mother board and the resulting system does not have enough memory to run full software. **Technico** is dedicated to giving maximum computing power per dollar.

For a **reasonable price** you can **own** an **18K byte Technico system capable** of accepting full **Basic** or an **Assembler Editor Linkage loader** and still leaving a **large 6, 8, or 10K byte user area,** depending on the software loaded.

The **Technico Super Basic** is the most extensive and **fastest** basic now offered in a personal computing system. Using the Kilobaud magazine **Benchmark Program No. 7,** which is the most difficult of their Benchmarks, as a test, the fastest time previously reported was by **Ohio Scientific** who ran the benchmark in **51 seconds.** The slowest time was by **Southwest Tech, at 235 seconds,** which is almost four minutes. All other systems were reported somewhere between these two times. The **Technico** System 16 ran the benchmark in **13 seconds,** thats **400% faster than OSI** and almost **2000% faster than Southwest Tech.** In more complex programs, the difference in speed of execution is even more drastic! The **reason for** this dramatic **performance** in increase in **speed** and **efficiency of** the TMS 9900 processor and the ease and efficiency of programming it provides to the talented programmer. Since the memory is BYTE or WORD addressable, and because of the large number of accumulators & registers the **Basic requires less code** and hence **less memory** than comparable basics on other processors.

The TMS9900 truly provides **mini-computer performance** in a microprocessor because it was designed to be the CPU of a data processing system, not to be a controller or logic replacement device like the 8 bit machines. The 9900 **is a miniaturized version of the discrete PC board** CPU **used by** Texas Instruments in **their 990/10 minicomputer.** TI merely took their existing CPU design and reduced it to a single N-MOS Silicon chip, which includes hardware multiply and divide. The TMS9900 not only **copies the architecture of the PDP11,** but adds the bit manipulation features of the 990/10 which makes it easy to interface to and to use in process control and data applications. Since the chip is a reproduction of the 990/10, it is software compatible with this larger machine and the smaller 990/4 TI minicomputer. As you can see from the TMS9900 comparison sheet with its 16 bit format, 16 accumulators and 15 registers, **the TMS9900 looks more like an IBM 360 than** it does **an 8080 or Z80 or 6502.** But yet, it is priced in the Technico system at a price competitive with the older 8 bit machines.

The **9900 will not be obsolete** as will today's 8 bit processors. Approximately **13 years ago TI introduced the 7400** series of integrated circuits and said they would produce a compatible family of IC's. At the time, the RTL and DTL circuits were popular. **Today,** 13 years later, the 7400 series **is the industry standard** and still in wide use and has not been obsoleted. TI **announced** a family of software compatible processors **in 1977,** starting with **the TMS9900.** So far the family has been expanded from the original 2.3 mhz version and the present 3.3 mhz version to I2L version which is approved for space flight use & in the future will reach 10 mhz speeds. A low cost 8 bit and single chip version (the 9940) with RAM, E-Prom on the chip are also available. It is reported that the 79 model year Chrysler will use the TMS9900 as its lean burn computer. TI buble memory boards and analog boards are planned to be compatible with the TMS9900. If the experience with the 7400 is any indication we can truly expect that **the 9900 system** you invest in today **may be around until 1990.** So get started with your system of the 90's, the Technico 9916 System, it's the best value in microcomputing.

The **Technico Super Starter** System is designed to **serve two purposes. One** is to be the **CPU** and peripheral interface **for** the powerful **system 16,** which can be **expanded to 65K bytes** with **dual floppy disks, digital cassettes, video graphics, six RS232 interfaces** and **over 192 bits of IO. A powerful systems monitor** and fully buffered data and address lines and **serial and parallel interface** are included on the Super Starter Board, together with the capacity for 2K of memory. Each additional card, by adding memory or IO, enhances an already operating system. The **super starter system** can **also** be an initial **stand alone microcomputer** with which **a person can learn** about **microcomputing** without spending a fortune.

Unlike other competing systems where the initial learning device must be thrown away and a new investment made to get a real working computer, there are **no throw away boards** or peripheral devices intended in a technico system. A user merely decides which level of system his knowledge, budget or application dictates. Any system **may be expanded** at a later date **without sacrificing** the **initial investment** that is made. For example there are many learning devices on the market in the price range of the super starter system which allow programming in hex. Unfortunately after a few weeks of hex programming the user's knowledge outgrows the kit and the entire system with hex keypad is discarded and the investment lost. This is not the intent of the super starter system. It is intended to become part of the system 16. The super starter system contains the powerful **16 bit TMS9900** microprocessor with **hardware multiply and divide, 16 accumulators, 15 index registers** and **separate 16 bit data and address lines.** The **board has capacity** for **2k bytes each of Ram, rom and E-Prom a parallel and serial RS232 and 20 milli amp current loop.**

It incorporates **8 vectored interrupts** and also contains as a free bonus, an **on board E-Prom 2708 programmer.** A 2716 programmer can be added at minimal cost. Because of the ease of interfacing to the TMS9900, the RS232 interface contains only a few transistors, and under the software control of the monitor it is **completely adjustable up to 9600 baud** by merely hitting reset and carriage return.

This makes the interface almost universal and allows the unit to communicate with any terminal or RS232 or 20 milliamp device up to 9600 baud. The **monitor** also contains **advanced commands,** such as **break point** and **snap,** so that the operating registers and memory locations can be displayed on the terminal during an operating program, thus eliminating the need for a front panel. A user program can be programmed into E-Prom with merely a "p" command from the monitor. In the prom area of the board **Rom based mini assembler,** which is called the **Instant Input Assembler** , can be inserted. This allows the programmer to work in assembly language. It also **converts** the mnemonic instructions of the TMS9900 **to Hex,** one instruction at a time, therefore, being both an excellent programming and learning tool. The memory of **512 bytes** which is provided **in** the **initial price** of the system can be expanded to 2K on the board. Additional memory in 8K byte increments can be added to 32K byte capacity memory add on boards allowing memory expansion to 65K bytes, not including mass storage or memory mapping. Of the 2K each of RAM, ROM and E-Prom (6K bytes total) capacity of the super starter board, 512 bytes of Ram and 1K bytes of Prom, which is the powerful systems monitor, are included in the initial price.

Additional RAM is purchased separately. In the remaining 1K byte ROM area, either the instant Input Assembler or expanded monitor can reside. The expanded monitor is used to control and provide reliable timing for the Technico **4800 baud digital cassette** (part No. T9948-C) which can be used for **program loading** and **program data storage.** The IIA can then be stored on digital cassette as can the other software, such as the assembler editor linking loader and basic.

The recommended system configuration to run either the assembler, editor, linking loader or basic is the Super Starter System with 2K bytes of memory and expanded monitor. To this is added a 16K byte memory add on board giving the system a RAM area of 18K bytes. This system in Tec-Kit form is part No.TK-2-18K or as an assembled and tested systems part No. TAS-18K. It is recommended that the 4800 baud cassette part No. T9948-C be hooked into the system to provide rapid program loading and storage. Since the cassette is not interfaced through the RS232 interface, the RS232 interface is available for connection to any standard terminal. By using this **cassette system, program loading is** approximately 400% faster than audio cassette and **2400% faster than paper tape. If the user does not have a terminal** the color graphics board can be combined with the system and **hooked to a TV or video monitor.** The interface to a television is done through an FCC approved device connected to the antenna. The memory of the system can be expanded with additional RAM boards. The input can be expanded by adding a board with six RS232 interfaces. Floppy disks can be added for expanded memory. An E-Prom board can be added for use with ROM based software. A 192 Bit IO board can be added for expanded IO. Because of the number of compatible boards and peripheral devices available and the fact that they can be used inside or outside a chassis, as well as be purchased on TEC-KITS™ or assembled and tested, the user is provided maximum flexibility to meet his desires, application or budget.

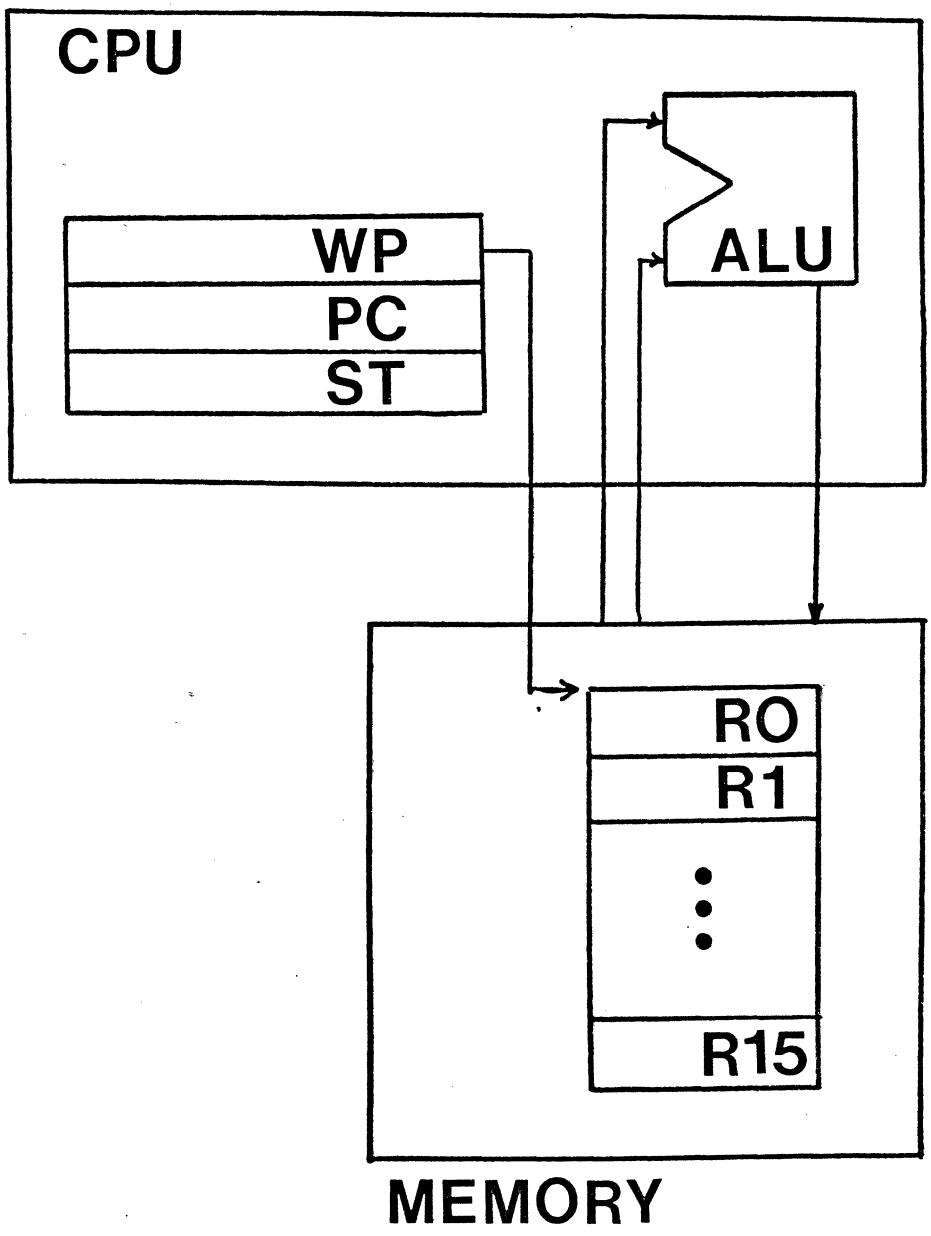
XIV. TI 9900 OVERVIEW

A. INTRODUCTION

The Texas Instruments 9900 is not the first 16-bit microprocessor to be intorduced, but it is probably the most powerful one. The architecture of the 9900 is unlike that of most other microprocessors (8 or 16 bits). It is more like that of a minicomputer. In fact, the 9900 is identical to the 990 microcomputer offered by T.I. This section provides an overview of the TI 9900 from a programming viewpoint. Combined with the detailed instruction descriptions in section IX you have all the tools to begin writing code.

As we already mentioned, the TI 9900 is a 16-bit microprocessor. Its architecture is vastly different from the simpler 8-bit microprocessors. One difference is that the registers are contained in memory. The only registers within the processor itself are: the program counter, status register, and a pointer to the registers in memory. The overall architecture is shown in FIGURE XIV-1. The program counter contains the address of the current instruction. The workspace pointer (WP) is a 16-bit register which holds the address of the first register in memory. The sixteen general registers R0-R16 are contained

Figure XIV-1 TMS9900 Architecture



in the sixteen sequential locations addressed by the WP.

For easy reference, the entire 9900 instruction set is described in detail in section IX and summarized at the end of that section.

Computations in the TI 9900 are performed between the registers, between the registers and memory, or between two memory locations. The memory of the 9900 is addressed by byte or word. The processor always references a word because the least significant address bit is not available as an external pin on the processor. Internally, however, you can address either words (two consecutive bytes, the address of the first one is even), or bytes. All instructions are stored as consecutive words. The addressing modes of the TI 9900 are:

(1) immediate - The operand is contained in the word following the instruction. For example,

```
LI   R1,>1234   ;  load R1 with 1234 (hex)
```

will load register R1 with the value 1234 (in hexadecimal notation). The symbol '>' indicates to the assembler that the value is hexadecimal, not decimal.

(2) register - The operand is contained in one of the general registers (R0-R15). These registers are actually in memory. The address of register 'x' is WP+2*x, where WP is the contents of the workspace pointer. You should be careful to preset WP at the beginning of your program. If not set properly, the registers may be located on top of your program, which will cause serious programming problems.

(3) register indirect - The operand is contained in the memory location whose address is contained in one of the general registers. For example:

```
MOV     *R1,R2      ; R2=(R1)
```

will load register R2 with the memory location whose address is contained in R1.

(4) register indirect, auto increment - The operand is contained in the memory location whose address is contained in one of the general

registers. After execution of the instruction, the register is incremented by one or two. If the instruction is a byte instruction (e.g. MOVB), then the register is incremented by one. If the instruction is a word instruction (e.g. MOV) the register is incremented by two. For example:

```
MOV     *R1+,R2
```

will load register R2 with the memory location whose address is contained in R1. After the move, register R1 is incremented by two since MOV is a word reference.

(5) indexed - The operand is contained in the memory location whose address is obtained by adding a constant to the contents of one of the general registers. If the register R0 is used, the operand address is merely the constant. To move the contents of a variable, called VAR, to register R1 we can use:

```
MOV     @VAR,R1
```

In this case, no index register was specified so the assembler assumes the R0 (no index) is desired. The following instruction:

```
MOV     @10(R1),R2
```

will load R2 with the memory location addressed by the contents of R1 plus 10.

(6) relative - Relative addressing is used to obtain the destination address for most of the 9900's jump instructions. To obtain the final destination address, the second byte of the instruction is multiplied by two and added to the address of the next sequential instruction. The addition is performed using two's complement arithmetic. This allows the programmer to transfer control to an address within the range of -254 to +256 of the present instruction. Since all instructions are stored as words (two bytes), we can transfer control to a word within the range of -127 to +128 of the present instruction. An example of relative addressing is:

```
JMP   +10
```

This instruction will transfer control to the address of the next sequential instruction plus 20 (10*2). If the jump were at >1200, this would transfer control to address >1216.

As you can see, the 9900's instruction set is more complicated than the run of the mill microprocessor. All of the op-codes are one word long. If immediate, indirect, or indexed addressing is used, the constant is stored in the word(s) following the op-code. The constant for the source operand is stored in the first word following the op-code and the constant for the destination operand is stored in the next available word. This means the 9900 instructions are one to three words long, or two to six bytes. The following six byter will transfer the contents of variable VAR1 to VAR2:

```
MOV   @VAR1,@VAR2      ; VAR2=VAR1
```

B. SUBROUTINE LINKAGE

Unlike many machines, the 9900 does not use a stack to hold subroutine return addresses. Instead, the processor saves the return address in general register R11. For example, the following instruction will save the address of BACK in R11 and will transfer control to ROUT:

```
            BL  @ROUT          ; call ROUT
BACK        :
            :
            :
```

To return from the subroutine, all you need to do is jump to the contents of R11 (B *R11).

If one subroutine must call another, the first subroutine must first save the contents of R11, since the new return address will be placed in R11 - thus destroying the old return address. There are several different ways to approach this problem. The first, and simplest, method is to save the return address in one of the general registers. For example, if ROUT is called as indicated above and must then call ROUT2, the sequence below can be used:

```
MOV   R11,R1          ; save return address
BL    @ROUT2          ; call next subroutine
.
.
.
B     *R1             ; exit
```

If you have only two or three levels of subroutine, this may be the most efficient approach. However, in larger systems there are usually too many levels of subroutines to store all the return addresses in the registers. In that case, the return address can be saved in RAM. One way to do that is:

```
MOV   R11,@TEMP       ; save return
```

To exit the subroutine, the following two instructions are used:

```
MOV   @TEMP,R11       ; get return
B     *R11            ; exit
```

The major disadvantage of this technique is that four words of instruction memory are required for the exit sequence, not to mention the word used to hold the return address. This is rather wasteful of memory. If the program

is always to be run in RAM (never put in PROM/ROM storage), an alternate entry/exit sequence is:

```
            MOV   R11,@EX+2       ; save return in exit branch
            .
            .
            .
EX          B     @0              ; exit
```

This time we saved the return address in the second word of the branch instruction, thus eliminating the move. The disadvantage here is that the program modifies itself. This means that the program can never be placed in ROM. Most microprocessor programs are eventually stored in ROM so this sequence couldn't be used. In fact, I would normally not recommend using any self-modifying techniques. However, if you are writing a quick and dirty routine, to be run only from RAM, this approach works well.

There is yet another way to save the return address. We can put it on the stack. What stack, you say? Because of the flexible modes of addressing, creation of a software stack is a very simple task. During the initial start of the program, we load one of the general registers, let's say R15, with the address of the first location of the stack. Then, an entry can be placed on the stack with

the following move:

```
        MOV   R11,*R15+          ; stack R11
```

The stack pointer is incremented after the store, so the stack builds up instead of down as in other micros. To retrieve an entry from the stack, the following instructions are used:

```
        DECT  R15                ; R15=R15-2
        MOV   *R15,R11           ; get the top entry
```

The stack could also be used to save some of the other general registers that would be used by the subroutine. If the subroutine requires a number of registers, another approach is to use the Branch and Link Workspace Pointer (BLWP). This instruction is also a subroutine call, but before performing the call it resets the workspace pointer. This means that the subroutine has a whole new set of registers to work with - without having to store the old ones! This instruction is very valuable, but should be used with discretion because it requires more memory. More memory for the call and sixteen words more memory for the new set of registers.

C. PASSING PARAMETERS

There are many differnt methods for passing data to to subroutines - in the registers, following the subroutine call, or addresses following the subroutine call. Since the return address of the routine is already in one of the general registers (R11), passing parameters or their addresses following the call is especially useful with the 9900. For example, consider the floating point subroutines called FMUL and FADD which are the multiply and add floating point routines, respectively. Each one requires three parameters, the address of which can be placed after the subroutine call. If this approach is used with the 9900, the following sequence is used to calculate X1=X2*X3+X4:

```
        BL      @FMUL           ; TMP=X2*X3
        DATA    X2
        DATA    X3
        DATA    TMP
        BL      @FADD           ; X1=TMP+X4
        DATA    TMP
        DATA    X4
        DATA    X1
```

Before we can manipulate the parameters, it would be necessary to place then in the registers, perhaps. This is easily accomplished by the following:

```
MOV   *R11+,R1        ; R1=address of param 1
MOV   *R11+,R2        ; R2=address of param 2
MOV   *R11+,R3        ; R3=address of param 3
```

Notice how the indirect with auto increment addressing mode avoids the need for intermediate increments.

D. RETURNING RESULTS

Many subroutines must return results to the calling program. The easiest way is to return the result in one of the general registers. This works fine if the subroutine is called via a BL instruction. On the other hand, if a BLWP (or XOP - which will be discussed later) is used, the calling routine uses a different set of registers than the subroutine. Therefore, if we place the results in the registers, they will be lost when control is returned to the calling program since the workspace pointer will be reset. Since the 9900's registers are located in memory, there is a simple way around this problem. Let's assume that we want to return a value in R0 and R1 - in

the old workspace. When the BLWP is executed, the old workspace pointer is saved in R13. Using this fact, we can create a sequence to store values in the previous workspace:

```
MOV   R0,*R13          ; old R0=new R0
MOV   R1,@2(R13)       ; old R1=new R1
```

As you see, the old register "1" is the same as memory location R13+2*1. That location may be addressed by @1+1(R13). R0 is a special case since @0(R13) is the same as *R13.

E. BYTE OPERATIONS

Although the 9900 is primarily a 16-bit processor, it can still handle most byte operations. There are a few aspects of the byte operations that can be confusing. First, whenever a register is addressed in the byte mode, the left byte of the register is used (not the right byte). Second, whenever the processor references memory it reads a full word. The proper byte of that word is selected within the processor. This means that it is not necessary for the processor to supply the external memory addressing circuitry with the least significant address bit - so it

does not. If you examine the hardware carefully you will note that there are only fifteen address bits. The missing bit is the least significant address bit. It is unnecessary because the processor performs the byte selection.

Recognizing the special byte addressing operation, you will quickly discover that the 9900 can cope with byte operands nearly as well as it can with full word operands. To add the contents of byte B1 to B2 we can use:

```
AB    @B1,@B2          ; B2=B2+B1
```

F. EXTENDED OPERATIONS

The TI 9900 offers a unique instruction - Extended Operation (XOP). The XOP execution is similar to the BLWP, but the target address is determined by the XOP transfer vectors - there are sixteen possible XOPs and the source operand is placed in R11 of the new workspace. For example, the following:

```
XOP   @X,15
```

will perform an extended operation 15 and will place the address of variable X in the new R11. The workspace pointer and address for extended operation 15 is in memory

locations 7C-7F. For other extended operations, the extended operation transfer vector is stored in location 40 + 4*I through 43 + 4*Il.

The monitor uses three extended operations. Refer to the monitor description fro details of the monitor XOP.

G. MULTIPLY/DIVIDE

One of the truly unique operations offered in the 9900 is the hardware multiply and divide. Notice, however, that they require unsigned operands. This is different than the other instructions, which use two's complement operands. We can easily form a signed two's complement multiply. If X1 and X2 are two arbitrary numbers, then X1*X2's sign is the exclusive-or of the signs of X1 and X2. Using this fact we can devise the routine to perform signed multiply. The sequence in Figure XIV_2 will calculate X3=X1*X2.

The multiply operation produces a 32-bit result (in R1,R2 for the example above), but does not affect any of the condition bits (thats why the test can be performed before the multiply). After the multiply, the result can be converted back to two's complement. Since you will often use the result for some further add/subtract operation, only the lower word of the product was

Figure XIV-2 Signed Multiply

X1 = address #200

X2 = address #202

X3 = address #204

```
?GF800

0100: C060  MOV @>200,R1        ; R1=X1
0102: 0200
0104: C0E0  MOV @>202,R3        ; R3=X2
0106: 0202
0108: C081  MOV R1,R2           ; R2(SIGN)=SIGN OF X1*X2
010A: 2883  XOR R3,R2
010C: 0741  ABS R1              ; GET RID OF SIGNS
010E: 0743  ABS R3
0110: C082  MOV R2,R2           ; TEST SIGN OF ANSWER
0112: 3843  MPY R3,R1           ; (R1,R2)=X1*X2 (MAGNITUDE)
0114: 1501  JGT >118            ; CORRECT THE SIGN
0116: 0502  NEG R2
0118: C802  MOV R2,@>204        ; X3=X1*X2 (LOWER 16 BITS)
011A: 0204
011C:
```

converted. If you need to convert both words, its a bit more difficult. The following sequence will not work:

```
NEG   R2
NEG   R3
```

Why not? If R2=1 and R3=1, then the two's complement of (R2,R3) is >FEFF. However, the two's complement of 1 is FF. So you see that the above sequence would yield >FFFF instead of the required >FEFF. The solution is to take the one's complement of R2 except in the case where R3=0. The required code is:

```
      INV   R2          ; R2=one's comp. of R2
      NEG   R3          ; R3=-R3
      JNE   ZRO         ; if R3=0, adjust R2
      INC   R2          ; R2=two's comp. of R2
ZRO   .
      .
      .
```

A similar approach can be used to construct a signed divide. The sign of X1/X2 is again the exclusive-or of X1,X2. If X1 and X2 are both 16-bit two's complement variables, then the routine in Figure XIV-3 will calculate X1/X2.

Figure XIV-3 Signed Divide

X1 = address 200

X2 = address 202

```
?GF800

0100: C0A0  MOV @>200,R2        ; R2=X1
0102: 0200
0104: C0E0  MOV @>202,R3        ; R3=X2
0106: 0202
0108: C102  MOV R2,R4           ; R4(SIGN)=SIGN OF X1/X2
010A: 2903  XOR R3,R4
010C: 0742  ABS R2              ; GET RID OF SIGNS
010E: 0743  ABS R3
0110: 04C1  CLR R1              ; CLEAR UPPER NUMERATOR BITS
0112: 3C43  DIV R3,R1           ; R1=(R1,R2)/R3
0114: C104  MOV R4,R4           ; CORRECT SIGN
0116: 1501  JGT >11A
0118: 0501  NEG R1
011A: C801  MOV R1,@>202        ; X2=X1/X2
011C: 0202
011E:
```

As you may have observed in that sequence, the divide operation divides a 32-bit operand by a 16-bit operand. Since we used only a 16-bit operand, the operand is placed in the lower register of the pair of registers and the upper register of the pair is cleared. If we want to use the full divide capability, the routine must be recoded as shown in Figure XIV-4.

The multiply is restricted to integer operands, but that does not mean you cannot use it to perform fractional operations. The approach is called scaling. Lets take a sample case. If the decimal point of X1 is at the extreme right and the decimal point of X2 is at the extreme left, then the decimal point of X1*X2 is between the two registers. Using this approach, we can multiply ABC by .75:

```
CON     DATA  >C000        ; constant of .75 (decimal
                            at left)
        MOV   @ABC,R1       ; get operand
        MPY   @CON,R1       ; (R1=integer part, R2=
                            fraction part)
```

In the beginning of this discussion, I indicated that it was unusual that the multiply was unsigned. Yet, we can turn this into an asset. Consider the problem of

Figure XIV-4 Full Divide

X1 = address #200 to 203

X2 = address #204 to 207

```
?GF800

0100: C060  MOV @>200,R1      ; (R1,R2)=X1
0102: 0200
0104: C0A0  MOV @>202,R2
0106: 0202    .
0108: C0E0  MOV @>204,R3      ; R3=X2
010A: 0204
010C: C101  MOV R1,R4         ; R4(SIGN)=SIGN OF X1/X2
010E: 2903  XOR R3,R4
0110: 0743  ABS R3            ; GET RID OF SIGN OF X2
0112: 0741  ABS R1            ; GET RID OF SIGN OF X1
0114: 1503  JGT >11C          ; IF X1 MINUS, INVERT LOWER HALF
0116: 0502  NEG R2
0118: 1301  JEQ >11C          ; IF R2 NONZERO, ADJUST R1
011A: 0601  DEC R1
011C: 3C43  DIV R3,R1         ; R1=X1/X2
011E: C104  MOV R4,R4         ; CORRECT SIGN
0120: 1501  JGT >124
0122: 0501  NEG R1
0124: C801  MOV R1,@>204      ; X2=X1/X2
0126: 0204
0128:
```

creating a double precision multiply (32-bits times 32-bits). If we consider unsigned numbers only (signs can be handled as in the previous examples), then a 32-bit multiply (which produces a 64-bit result) can be formed using four single precision multiplies. Figure XIV-5 illustrates the concept. We just use what is commonly called "cross multiply" techniques. Before presenting the double precision multiply, lets look at the double precision add which is an integral part of the multiply routine. To calculate (R1,R2)=(R1,R2) + (R3,R4) we can use the following (all values are assumed to be unsigned):

```
        A     R4,R2          ; add lower half
        JNC   L1             ; if Cy, correct upper
        INC   R1
L1      A     R3,R1          ; add upper half
```

Now, using this same concept for the subproduct additions, we can create the 32-bit multiply routine shown in Figure XIV-6.

H. ARITHMETIC

The advanced instruction set of the TI 9900 opens up a new microprocessor application area - signal processing. Because of the mathematics involved, most signal processing

Figure XIV-5 Multiprecision multiply

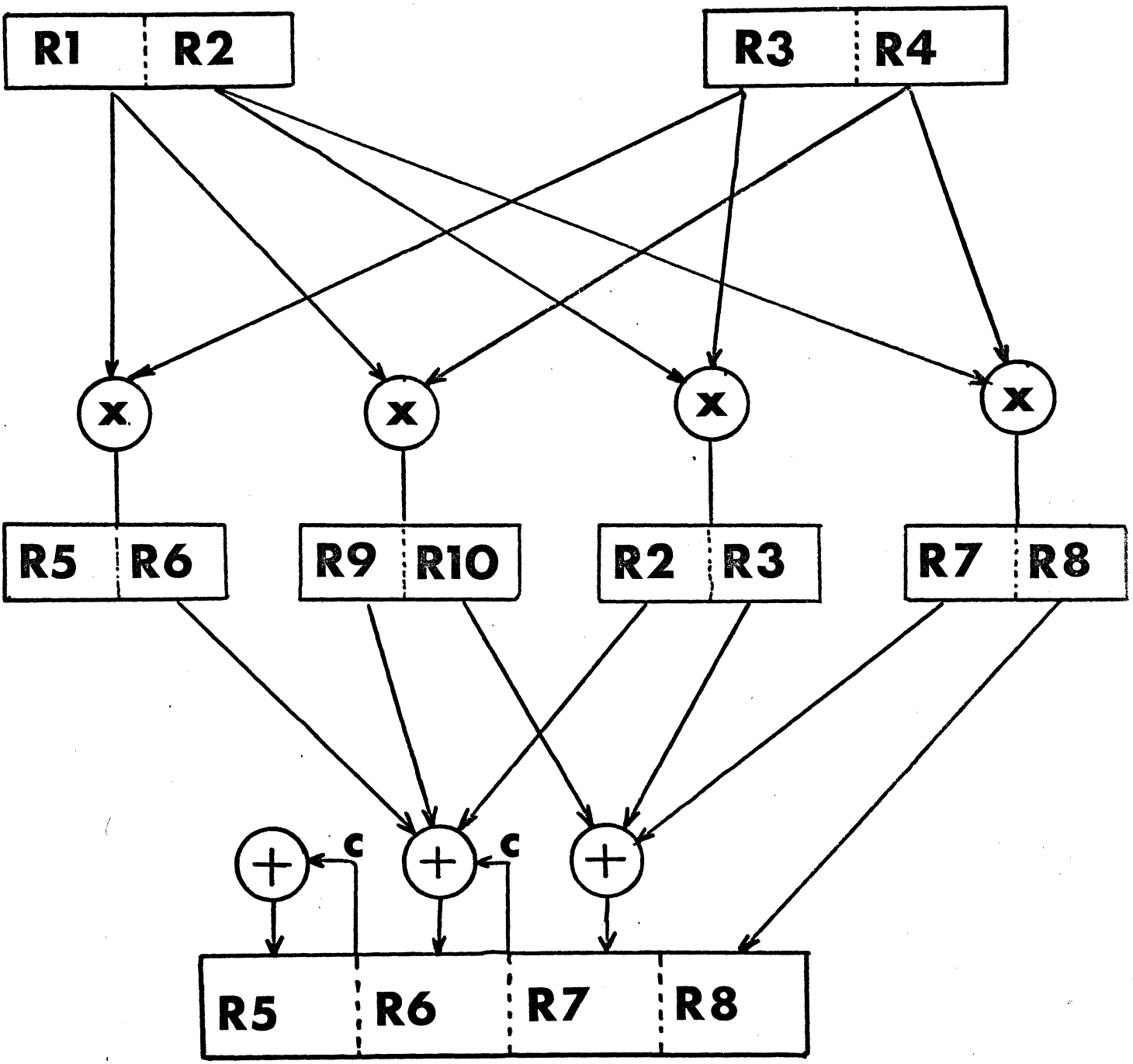


Figure XIV-6 Double Precision Multiply

```
?GF800

0100: C141   MOV R1,R5       ; R5,R6 = R1*R3
0102: 3943   MPY R3,R5
0104: C1C2   MOV R2,R7       ; R7,R8 = R2*R4
0106: 39C4   MPY R4,R7
0108: C241   MOV R1,R9       ; R9,R10 = R1*R4
010A: 3A44   MPY R4,R9
010C: 38C2   MPY R2,R3       ; R3,R4 = R2*R3
010E: 04C0   CLR R0          ; R0=CARRY ACCUMULATOR
0110: A1C3   A R3,R7
0112: 1701   JNC >116
0114: 0580   INC R0
0116: A1CA   A R10,R7
0118: 1701   JNC >11C
011A: 0580   INC R0
011C: 04C1   CLR R1          ; R1=CARRY ACCUMULATOR
011E: A182   A R2,R6
0120: 1701   JNC >124
0122: 0581   INC R1
0124: A189   A R9,R6
0126: 1701   JNC >12A
0128: 0581   INC R1
012A: A180   A R0,R6         ; ADD IN FIRST CARRY
012C: 1701   JNC >130
012E: 0581   INC R1
0130: A141   A R1,R5         ; ADD IN SECOND CARRY
0132:
```

tasks cannot be done with the off-the-shelf microprocessor. The 9900 certainly cannot handle all of the signal processing applications, but it can tackle a few of them.

Most signal processing algorithms use the SIN, COS, or other trigonometric functions as an integral part of the filter computation. One trigonometric algorithm - ideally suited to the 9900, is the CORDIC (COordinate Rotation Digital Computer) algorithm. Although you may not recognize it, it is the same algorithm used in many of the hand calculators. We will see later why the TI 9900 is ideally suited for the CORDIC procedure.

The CORDIC algorithm relies on a few very simple mathematical facts. First, any given angle (we will restrict the angle to 0-90°) can be represented as a sum/difference of a set of base angles. Mathematically this can be expressed as:

$$A = \sum d_i \, a_i \qquad \text{where } d_i = \pm 1$$

a_i = base angle

This identity is certainly not true for any random selection of base angles, but yoy can intuitively see the 90°, 45°, 22.5°, ... is one possible base set. The second cornerstone of this algorithm is a pair of trigonometric identities:

$$SIN\ (a+b) = (SIN(a) + TAN(b)COS(a))\ COS(b)$$

$$COS\ (a+b) = (COS(a) - TAN(b)SIN(a))\ COS(b)$$

Now, if we have a given angle represented as a sum/difference of a set of base angles - which are as yet unspecified - then we can devise a simple process for calculating the SIN and COS of that angle:

$$X_0 = 0$$

$$Y_0 = 1$$

$$X_i = X_{i-1} + TAN(d_i a_i) * Y_{i-1}$$

$$Y_i = Y_{i-1} - TAN(d_i a_i) * X_{i-1}$$

After executing the above procedure, we don't really have the SIN and COS. Instead, we have $X_n = R_n\ SIN(0)$ and $Y_n = R_n\ COS(0)$, where the constant R_n is $1/(COS(d_i a_i) * \ldots * COS(d_n a_n))$. So far, we have nothing to cheer about because our algorithm involves many more multiplies than a simple Taylor series. But, the plot thickens. If we define the base angles as:

$$a_i = TAN^{-1}(\tfrac{1}{2}^{i-1})$$

then

$$TAN(a_i) = \tfrac{1}{2}^{i-1}$$

This means that all of the multiply operations can be reduced to a right shift. We must, of course, prove that all angles can be represented as a sum of our base angles or the whole algorithm collapses. I will not do so here, but it can be done rather easily. Now, if we use base angles as I defined above, the algorithm may be restated as:

$$V_o = -0$$
$$X_o = 0$$
$$Y_o = 1/Rn = .60725$$
$$X_i = X_{i-1} - SIGN(V_{i-1})*Y_{i-1}/2^{i-1}$$
$$Y_i = Y_{i-1} + SIGN(V_{i-1})*X_{i-1}/2^{i-1}$$
$$Vi = V_{i-1} - SIGN(V_{i-1})*(ATAN(1/2^{i-1}))$$

If we store the ArcTan values in a table, then this algorithm requires only shift, add, and subtract. The shift operation requires a variable shift constant. This is why the algorithm fits nicely in the 9900. If the shift count is stored in R0, the variable shift can be performed by a single 9900 instruction:

```
SRA     R1,R0          ; shift R1 right by (R0)
```

Since the SIN and COS are fractional values, we must scale the input to our routine. To keep matters simple, we scale the angle so that R1=angle*256. This provides 8-bits of integer and 8-bits of fraction. We scale the X,Y values so that X=SIN*32768, and Y=COS*32768. This provides 16-bits of signed fraction. The entire algorithm is shown in Figure XIV-7. The input angle is in R1, and the outputs are in R2 and R3. This subroutine calculates both the SIN and COS. The TAN can be calculated by one additional divide. As you see, this algorithm is a very fast and efficient way to obtain the trigonometric values.

Figure XIV-7 CORDIC Routine

```
?GF800

0100: 04C2  CLR R2            ; X=0
0102: 0203  LI R3,19898       ; Y=6072526*2**15
0104: 4DBA
0106: 04C4  CLR R4            ; X0=0
0108: C143  MOV R3,R5         ; Y0=Y
010A: 04C0  CLR R0            ; SHIFT=0
010C: 04C6  CLR R6            ; COUNT=0
010E: 0501  NEG R1            ; V=-V
0110: C041  MOV R1,R1         ; TEST SIGN OF ANGLE
0112: 1105  JLT >11E          ; JUMP IF MINUS
0114: 6085  S R5,R2           ; C=C-Y/2**I
0116: A0C4  A R4,R3           ; Y=Y+X/2**I
0118: 6066  S @>140(R6),R1    ; V=V-ATAN(1/2**I)
011A: 0140
011C: 1004  JMP >126          ; CONTINUE
011E: A085  A R5,R2           ; X=X+Y/2**I
0120: 60C4  S R4,R3           ; Y=Y-X/2**I
0122: A066  A @>140(R6),R1    ; V=V+ATAN(1/2**I)
0124: 0140
0126: 0580  INC R0            ; UPGRADE SHIFT COUNT
0128: 05C6  INCT R6           ; UPGRADE ANGLE INDEX
012A: C102  MOV R2,R4         ; R4=X/2**I
012C: 0804  SRA R4,R0
012E: C143  MOV R3,R5         ; R5=Y/2**I
0130: 0805  SRA R5,R0
0132: 0280  CI R0,12          ; END?
0134: 000C
0136: 16EC  JNE >110
0138: 045B  B *R11            ; RETURN TO CALLER

013A:       /140              ; ENTER CONSTANTS
0140: 2D00  +11520            ; ATAN (1/1)*256
0142: 1A90  +6800             ; ATAN (1/2)*256
0144: 0E09  +3593             ; ATAN (1/4)*256
0146: 0720  +1824             ; ATAN (1/8)*256
0148: 0394  +916              ; ATAN (1/16)*256
014A: 01CA  +458              ; ATAN (1/32)*256
014C: 00E5  +229              ; ATAN (1/64)*256
014E: 0073  +115              ; ATAN (1/128)*256
0150: 0039  +57               ; ATAN (1/256)*256
0152: 001D  +29               ; ATAN (1/512)*256
0154: 000E  +14               ; ATAN (1/1024)*256
0156: 0007  +7                ; ATAN (1/2048)*256
```