As you are now the owner of this document which should have come to you for free, please consider making a donation of £1 or more for the upkeep of the (Radar) website which holds this document. I give my time for free, but it costs me money to bring this document to you. You can donate here https://blunham.com/Misc/Texas

Many thanks.

Please do not upload this copyright pdf document to any other website. Breach of copyright may result in a criminal conviction.

This Acrobat document was generated by me, Colin Hinson, from a document held by me. I requested permission to publish this from Texas Instruments (twice) but received no reply. It is presented here (for free) and this pdf version of the document is my copyright in much the same way as a photograph would be. If you believe the document to be under other copyright, please contact me.

The document should have been downloaded from my website https://blunham.com/, or any mirror site named on that site. If you downloaded it from elsewhere, please let me know (particularly if you were charged for it). You can contact me via my Genuki email page: https://www.genuki.org.uk/big/eng/YKS/various?recipient=colin

You may not copy the file for onward transmission of the data nor attempt to make monetary gain by the use of these files. If you want someone else to have a copy of the file, point them at the website. (https://blunham.com/Misc/Texas). **Please do not point them at the file itself as it may move or the site may be updated.**

It should be noted that most of the pages are identifiable as having been processed by me.

I put a lot of time into producing these files which is why you are met with this page when you open the file.

If you find missing pages, pages in the wrong order, anything else wrong with the file or simply want to make a comment, please drop me a line (see above).

It is my hope that you find the file of use to you.

Colin Hinson
In the village of Blunham, Bedfordshire.

TEXAS INSTRUMENTS

TM 990

TM 990/1481 High–Performance CPU Modules

Volume II

MICROPROCESSOR SERIES™

October 1980

TABLE OF CONTENTS

TABLE OF CONTENTS

APPENDIX A

WIRING TELETYPE MODEL 3320/5JE FOR TM 990/1481

A-1 GENERAL

Figure A-1 shows the wiring configuration required to connect a 3320/5JE Teletype in a 20 mA current loop with a TM 990/1481. Other teletypewriter models may require different connections; therefore, consult the manufacturer for correct wiring of other models.

CAUTION

Note the 117 Vac connection at pins 1 and 2. Be sure that this voltage is not accidently wired to the TM 990/1481 board.

A-2 CONNECTIONS

The following assumes that the teletypewriter is wired as it came from the factory.

(1) Locate the 151411 terminal block at the left rear (viewed from the rear) of the machine (Figure A-1).

(2) Move the white/blue wire from terminal 4 to terminal 5 on the terminal block.

(3) Move the brown/yellow wire from terminal 3 to terminal 5 on the terminal block.

(4) Move the purple wire from terminal 8 to terminal 9 on the terminal block (for 20 mA neutral signaling).

(5) Locate the power resistor behind the teletype power supply. Remove the blue wire from the 750 ohm tap and connect it to the 1450 ohm tap, as shown in Figure A-2.

(6) Check pins 3, 4, 6, and 7 at terminal strip 151411. Voltage to ground must be zero with power applied. If not, do not connect to the TM 990/1481.

NOTE

For teletypewriter operation a jumper must be installed between E1 and E2. At all other times, this jumper should be removed.

A-3 TROUBLESHOOTING

If the printer continues to chatter after the RESET switch on the TM 990/1481 has been activated, reverse connections 6 and 7 at the terminal strip.

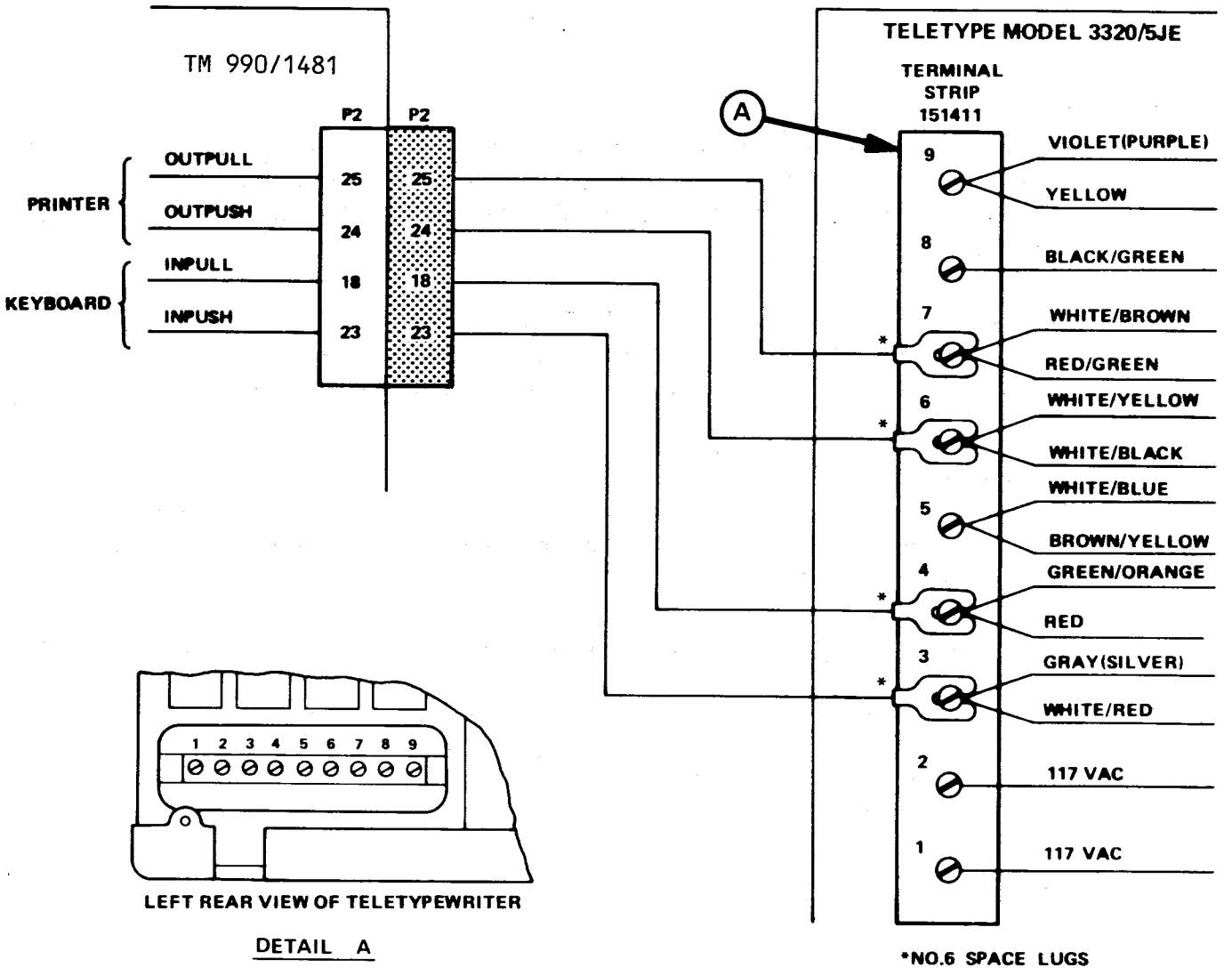
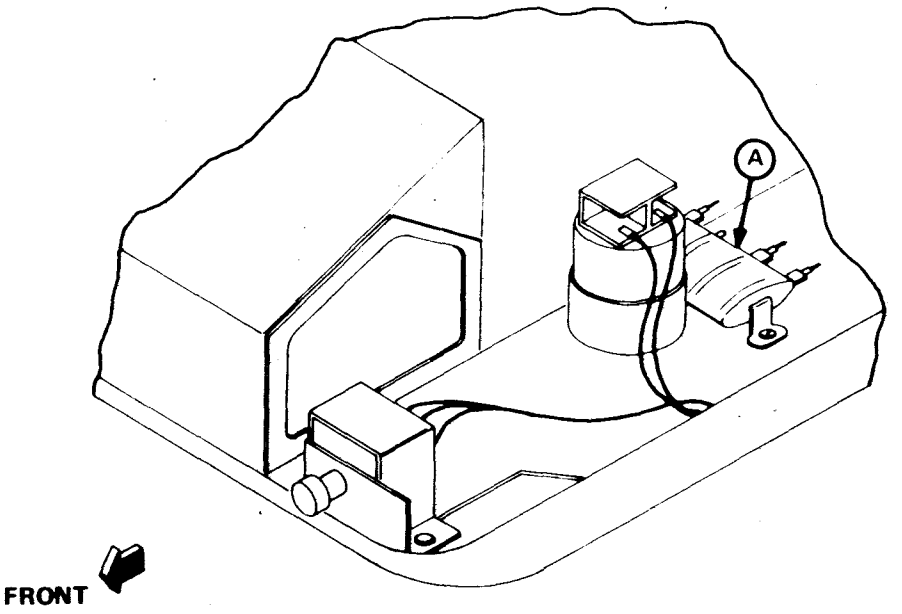
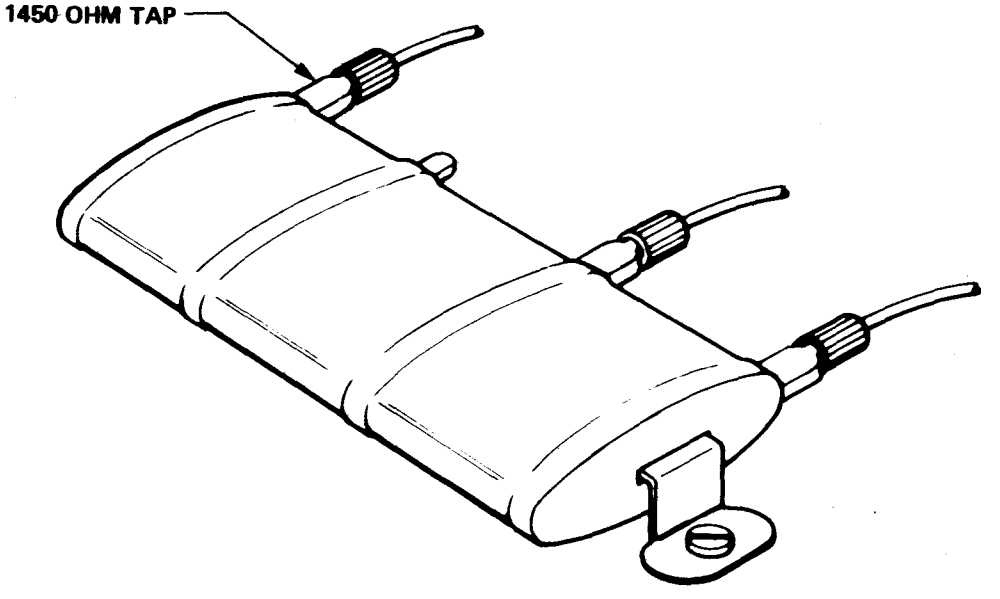


FIGURE A-1. TELETYPEWRITER TERMINAL STRIP CONNECTIONS





DETAIL A

FIGURE A-2. TELETYPEWRITER RESISTOR CONNECTION

APPENDIX B

TM 990/1481 CRU MAP

CRU SOFTWARE BASE ADDRESS (HEX)	BIT ADDRESS (HEX)	FUNCTION	INPUT	OUTPUT
0080_{16}	0040	SERIAL I/O PORT	RBR0	CTRL0
	0041	TMS 9902	RBR1	CTRL1
	0042		RBR2	CTRL2
	0043		RBR3	CTRL3
	0044		RBR4	CTRL4
	0045		RBR5	CTRL5
	0046		RBR6	CTRL6
	0047		RBR7	CTRL7
	0048		0	CTRL8
	0049		RCVERR	CTRL9
	004A		RPER	CTRL10
	004B		ROVER	LXDR
	004C		RFER	LRDR
	004D		RFBD	LDIR
	004E		RSBD	LDCTRL
	004F		RIN	TSTMD
	0050		RBINT	RTSON
	0051		XBINT	BRKON
	0052		0	RIENB
	0053		TIMINT	XBIENB
	0054		DSCINT	TIMENB
	0055		RBRL	DSCENB
	0056		XBRE	NOT USED
	0057		XSRE	NOT USED
	0058		TIMERR	NOT USED
	0059		TIMELP	NOT USED
	005A		RTS	NOT USED
	005B		DTR	NOT USED
	005C		CTS	NOT USED
	005D		DSCH	NOT USED
	005E		FLAG	NOT USED
	005F		INT	RESET
	0060	RESERVED		
	007F	RESERVED		
	0080	TMS 9901 PSI	CONTROL BIT	CONTROL BIT
0100_{16}	0081		INT1/CLK1	MASK1/CLK1
	0082		INT2/CLK2	MASK2/CLK2
	0083		INT3/CLK3	MASK3/CLK3
	0084		INT4/CLK4	MASK4/CLK4
	0085		INT5/CLK5	MASK5/CLK5
	0086		INT6/CLK6	MASK6/CLK6
	0087		INT7/CLK7	MASK7/CLK7
	0088		INT8/CLK8	MASK8/CLK8
	0089		INT9/CLK9	MASK9/CLK9
	008A		INT10/CLK10	MASK10/CLK10
	008B		INT11/CLK11	MASK11/CLK11

TM 990/1481 CRU MAP (CONCLUDED)

CRU SOFTWARE BASE ADDRESS (HEX)	BIT ADDRESS (HEX)	FUNCTION	INPUT	OUTPUT
	008C		INT12/CLK12	MASK12/CLK12
	008D		INT13/CLK13	MASK13/CLK13
	008E		INT14/CLK14	MASK14/CLK14
	008F		INT15/INTREQ	MASK15/RST2
	0090		P0 INPUT	P0 OUTPUT
	0091		P1 INPUT	P1 OUTPUT
	0092		P2 INPUT	P2 OUTPUT
	0093		P3 INPUT	P3 OUTPUT
	0094		P4 INPUT	P4 OUTPUT
	0095		P5 INPUT	P5 OUTPUT
	0096		P6 INPUT	P6 OUTPUT
	0097		P7 INPUT	P7 OUTPUT
	0098		P8 INPUT	P8 OUTPUT
	0099		P9 INPUT	P9 OUTPUT
	009A		P10 INPUT	P10 OUTPUT
	009B		P11 INPUT	P11 OUTPUT
	009C		P12 INPUT	P12 OUTPUT
	009D		P13 INPUT	P13 OUTPUT
	009E		P14 INPUT	P14 OUTPUT
	009F		P15 INPUT	P15 OUTPUT

APPENDIX C

ASCII CODE

TABLE C-1. *ASCII CONTROL CODES

CONTROL	BINARY CODE	HEXADECIMAL CODE
NUL – Null	000 0000	00
SOH – Start of heading	000 0001	01
STX – Start of text	000 0010	02
ETX – End of text	000 0011	03
EOT – End of transmission	000 0100	04
ENQ – Enquiry	000 0101	05
ACK – Acknowledge	000 0110	06
BEL – Bell	000 0111	07
BS – Backspace	000 1000	08
HT – Horizontal tabulation	000 1001	09
LF – Line feed	000 1010	0A
VT – Vertical tab	000 1011	0B
FF – Form feed	000 1100	0C
CR – Carriage return	000 1101	0D
SO – Shift out	000 1110	0E
SI – Shift in	000 1111	0F
DLE – Data link escape	001 0000	10
DC1 – Device control 1	001 0001	11
DC2 – Device control 2	001 0010	12
DC3 – Device control 3	001 0011	13
DC4 – Device control 4 (stop)	001 0100	14
NAK – Negative acknowledge	001 0101	15
SYN – Synchronous idle	001 0110	16
ETB – End of transmission block	001 0111	17
CAN – Cancel	001 1000	18
EM – End of medium	001 1001	19
SUB – Substitute	001 1010	1A
ESC – Escape	001 1011	1B
FS – File separator	001 1100	1C
GS – Group separator	001 1101	1D
RS – Record separator	001 1110	1E
US – Unit separator	001 1111	1F
DEL – Delete, rubout	111 1111	7F

*American Standards Institute Publication X3.4-1968

TABLE C-2. *ASCII CHARACTER CODE

CHARACTER	BINARY CODE	HEXADECIMAL CODE	CHARACTER	BINARY CODE	HEXADECIMAL CODE
Space	010 0000	20	P	101 0000	50
!	010 0001	21	Q	101 0001	51
" (dbl. quote)	010 0010	22	R	101 0010	52
#	010 0011	23	S	101 0011	53
$	010 0100	24	T	101 0100	54
%	010 0101	25	U	101 0101	55
&	010 0110	26	V	101 0110	56
' (sgl. quote)	010 0111	27	W	101 0111	57
(	010 1000	28	X	101 1000	58
)	010 1001	29	Y	101 1001	59
* (asterisk)	010 1010	2A	Z	101 1010	5A
+	010 1011	2B	[	101 1011	5B
, (comma)	010 1100	2C	\	101 1100	5C
– (minus)	010 1101	2D	]	101 1101	5D
. (period)	010 1110	2E	^	101 1110	5E
/	010 1111	2F	_ (underline)	101 1111	5F
0	011 0000	30	`	110 0000	60
1	011 0001	31	a	110 0001	61
2	011 0010	32	b	110 0010	62
3	011 0011	33	c	110 0011	63
4	011 0100	34	d	110 0100	64
5	011 0101	35	e	110 0101	65
6	011 0110	36	f	110 0110	66
7	011 0111	37	g	110 0111	67
8	011 1000	38	h	110 1000	68
9	011 1001	39	i	110 1001	69
:	011 1010	3A	j	110 1010	6A
;	011 1011	3B	k	110 1011	6B
<	011 1100	3C	l	110 1100	6C
=	011 1101	3D	m	110 1101	6D
>	011 1110	3E	n	110 1110	6E
?	011 1111	3F	o	110 1111	6F
@	100 0000	40	p	111 0000	70
A	100 0001	41	q	111 0001	71
B	100 0010	42	r	111 0010	72
C	100 0011	43	s	111 0011	73
D	100 0100	44	t	111 0100	74
E	100 0101	45	u	111 0101	75
F	100 0110	46	v	111 0110	76
G	100 0111	47	w	111 0111	77
H	100 1000	48	x	111 1000	78
I	100 1001	49	y	111 1001	79
J	100 1010	4A	z	111 1010	7A
K	100 1011	4B	{	111 1011	7B
L	100 1100	4C	|	111 1100	7C
M	100 1101	4D	}	111 1101	7D
N	100 1110	4E	~	111 1110	7E
O	100 1111	4F			

*American Standards Institute Publication X3.4-1968

APPENDIX D

TM 990/301 MICROTERMINAL

D.1 GENERAL

The Texas Instruments Microterminal offers all of the features of a minicomputer front panel at reduced cost. The Microterminal allows the user to do the following:

- Read from ROM or read/write to RAM
- Enter/display Program Counter
- Execute user program in free running mode or in single instruction mode
- Halt user program execution
- Enter/display Status Register
- Enter/display Workspace Pointer (this term is unique to the Texas Instruments 9900 microprocessor)
- Enter/display CRU data (this term is unique to the Texas Instruments 9900 microprocessor)
- Convert hexadecimal quantity to signed decimal quantity
- Convert signed decimal quantity to hexadecimal quantity

D.2 SPECIFICATIONS

- Power Requirements
  +12V (±3%), 50 mA
  −12V (±3%), 50 mA
  +5V (±3%), 150 mA
- Operating Temperature: 0°C to 50°C (+32° to +122°F)
- Operating Humidity: 0 to 95 percent, non-condensing
- Shock: Withstand 2 foot vertical drop

D.3 INSTALLATION AND STARTUP

To install the Microterminal onto a TM 990/1481 microcomputer, do the following:

- Attach the EIA cable from the Microterminal to connector P2 of controller. Signals between the Microterminal and the microcomputer are listed as in Table D-1.
- To initialize the system, actuate the microcomputer RESET switch, then press the microterminal CLR key.

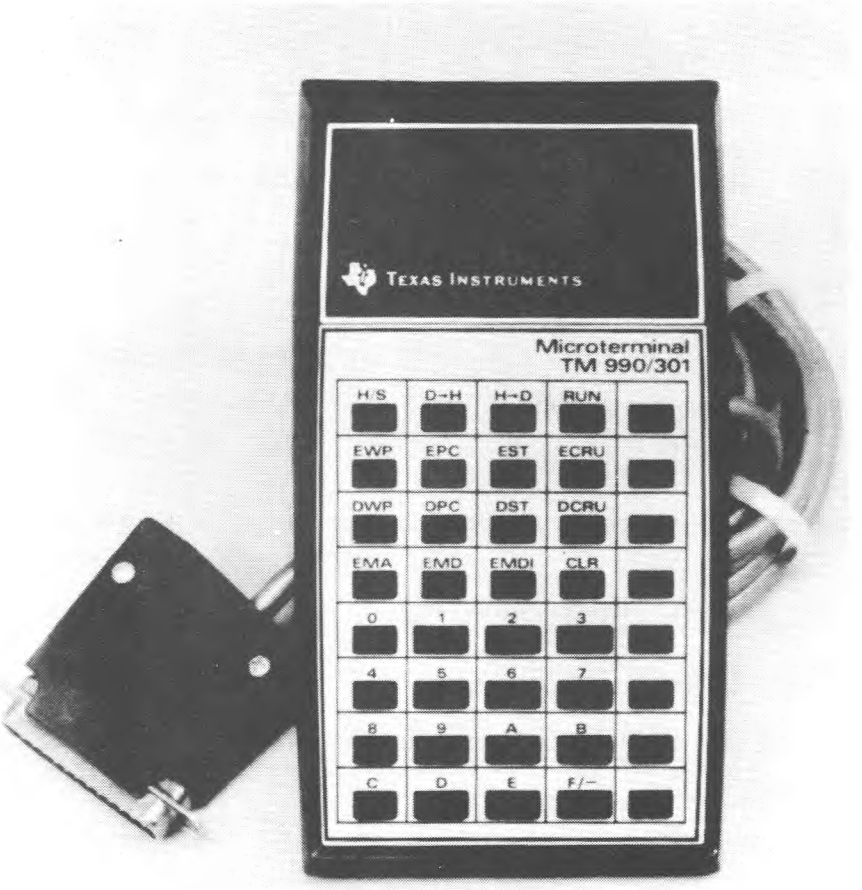


FIGURE D-1. TM 990/301 MICROTERMINAL

TABLE D-1. EIA CABLE SIGNALS

EIA Connector Pin	Interface Signal	At TM 990/1481	
		P2 Pin	Signal
2	$\overline{\text{TERMINAL DATA OUT}}$	−2	RS232 RCV
3	$\overline{\text{TERMINAL DATA IN}}$	−3	RS232 XMT
7	GND	−7	GND
12	+12V	−12	+12V
13	−12V	−13	−12V
14	+ 5V	−14	+ 5V
16	$\overline{\text{HALT}}$	−16	$\overline{\text{RESTART}}$

CAUTION

Before attaching the Microterminal to a power source, verify voltage levels between ground and EIA connector pins 12, 13, and 14 at connector P2 on the board. Voltage should not exceed values in Table I-1.

D.4 KEY DEFINITIONS

D.4.1 DATA KEYS

[CLR] Clear Key – Depressing this key blanks display, initializes and sends initialization message (ASCII code for A and ASCII code for Z) to host microcomputer.

[0]
[1]
.
.
.
[F/–]

Hexadecimal Data Keys – Depressing any one of these keys shifts that value into the right-hand display digit. All digits already in the data display are left shifted. For all operations other than decimal to hexadecimal conversion, the fourth digit from the right is shifted off the end of the right-hand display field when a data key is depressed. For a decimal to hexadecimal conversion, the fifth display digit from the right, rather than the fourth, is shifted off the end of the data field.

D.4.2 INSTRUCTION EXECUTION

[H/S] Pressing this key while a program is running (run displayed) will halt program execution. The address of the next instruction will be displayed in the four left-hand display digits, and the contents of that address will be displayed in the four right-hand digits. Pressing this key while the program is halted, will execute a single instruction using the values in the Workspace Pointer (WP), Program Counter (PC), and Status Register (ST), and the displays will be updated to the next memory address and contents at that address.

[RUN] Pressing this key initiates program execution at the current values in the WP, PC; run is displayed in the three right-hand display digits.

D.4.3 ARITHMETIC

[H→D] The signed hexadecimal data contained in the four right-hand display digits is converted to signed decimal data. Note that the fourth display digit from the right is the sign bit (1 = negative). The conversion limits are minus $32{,}768_{10}$ (8000_{16}) to plus 32,767 ($7FFF_{16}$). Two H→D key depressions are required. The sequence is:

1. Depress [H→D].
2. Enter data via hex data key depressions.
3. Depress [H→D]. The results of the conversion are displayed in the five right-hand display digits.

[D→H] The decimal data contained in the five right-hand display digits is converted to hexadecimal. The conversion limits are the same as for hexadecimal to decimal conversion. The sequence is:

1. Depress [D→H].
2. Enter data via hex data key depressions.
3. Depress [D→H]. The results of the conversion are displayed in the four right-hand display digits.

D.4.4 REGISTER ENTER/DISPLAY

[EWP] Pressing this key causes the value displayed in the four right-hand digits to be entered into the WP.

[DWP] Pressing this key causes the WP contents to be displayed in the four right-hand display digits.

[EPC] Pressing this key causes the value displayed in the four right-hand digits to be entered into the PC.

[DPC] Pressing this key causes the PC contents to be displayed in the four right-hand display digits.

[EST] Pressing this key causes the value displayed in the four right-hand digits to be entered into the ST.

[DST] Pressing this key causes the ST contents to be displayed in the four right-hand display digits.

D.4.5 CRU DISPLAY/ENTER

[DCRU] Pressing this key causes the data at the designated Communications Register Unit (CRU) addresses to be displayed. Designate from one to 16 CRU bits at a specified CRU address by using four hexadecimal digits. The first digit is the count of bits to be displayed. The next three digits are the CRU address (equal to bits 3 to 14 in register 12 for CRU addressing). When [DCRU] is depressed, the bit count and address are shifted to the left-hand display, and the right-hand display will contain the values at the selected CRU output addresses. The output value will be zero-filled on the left, depending upon bit count entered. If less than nine bits, the value will be contained in the left two hexadecimal digits. If nine or more, the value will be right justified in all four hexadecimal digits.

[ECRU] Pressing this key enters a new value at the CRU addresses and bit count shown in the left display after depressing [DCRU]. The new value is entered from the keyboard and displayed in the right-hand display. Pressing [ECRU] enters this value onto the CRU at the address shown in the left display.

CAUTION

Avoid setting new values at the TMS 9902 on the TM 990/1481 through the CRU (TMS 9902 is at CRU address 0040_{16}), as this device controls I/O functions.

D.4.6 MEMORY ENTER, DISPLAY, INCREMENT

[EMA] Pressing this key will cause (1) the memory address (MA) in the right-hand display to be shifted to the left-hand display and (2) the contents of that memory address to be displayed in the right-hand display.

[EMD] Pressing this key causes the value in the right-hand display to be entered into the memory address contained in the left-hand display. The contents of that location will then be displayed in the four right-hand display digits (entered then read back).

[EMDI] Pressing this key causes the same action as described for the [EMD] key; it also increments the memory address by two and displays the contents at that new address. The memory address is displayed on the left and the contents at that address is displayed on the right.

D.5 EXAMPLES

D.5.1 EXAMPLE 1, ENTER PROGRAM INTO MEMORY

Enter the following program starting at RAM location $FE00_{16}$. Set the workspace pointer to $FF00_{16}$ and the status register to 2000_{16}. Single step through the program and verify execution. Then execute the program in free run mode and verify execution. Then halt program execution.

NOTE

In the following examples, XXXX indicates memory contents at current value in Memory Address Register.

OPCODE	INSTRUCTIONS		
04C0	CLR	R0	CLEAR WORKSPACE REGISTER 0
0580	INC	R0	INCREMENT WORKSPACE REGISTER 0
0280	CI	R0, >00FF	CHECK FOR COUNT 255
00FF			
16FC	JNE	$–6	JUMP TO INC R0 IF NOT DONE
10FF	JMP	$–0	STAY HERE WHEN FINISHED

	KEY ENTRIES		DISPLAY
Clear Display	Depress	CLR	
Enter PC Value	Depress	F/– E 0 0	FE00
Enter into PC	Depress	EPC	FE00
Display PC	Depress	DPC	FE00
Enter ST Value	Depress	2 0 0 0	2000
Enter into ST	Depress	EST	2000
Display ST	Depress	DST	2000
Enter WP Value	Depress	F/– F/– 0 0	FF00
Enter Into WP	Depress	EWP	FF00
Display WP	Depress	DWP	FF00
Enter MA Value	Depress	F/– E 0 0	FE00
Enter Into MA	Depress	EMA	FE00 xxxx
Enter CLR 0 Opcode	Depress	0 4 C 0	FE00 04C0
Enter data, increment MA	Depress	EMDI	FE02 xxxx
Enter INC 0 Opcode	Depress	0 5 8 0	FE02 0580
Enter Data, Increment MA	Depress	EMDI	FE04 xxxx
Enter CI Opcode	Depress	0 2 8 0	FE04 0280
Enter Data, Increment MA	Depress	EMDI	FE06 xxxx

		KEY ENTRIES	DISPLAY
Enter CI Immediate Operand	Depress	0 0 F F	FE06 00FF
Enter Data, Increment MA	Depress	EMDI	FE08 xxxx
Enter JNE $-6 Opcode	Depress	1 6 F C	FE08 16FC
Enter Data, Increment MA	Depress	EMDI	FE0A xxxx
Enter JMP $-0 Opcode	Depress	1 0 F F	FE0A 10FF
Enter Data, Increment MA	Depress	EMDI	FE0C xxxx

The program has now been entered into RAM. Since the PC, ST and WP values have been previously set, the program can be executed in single step mode by depressing the H/S key.

		DISPLAY (AFTER)	EXECUTES INSTRUCTION
Depress	H/S	FE02 0580	CLR RO
Depress	H/S	FE04 0280	INC RO
Depress	H/S	FE08 16FC	CI RO,>00FF
Depress	H/S	FE02 0580	JNE $-6

This cycle will continue until R0 reaches the count of 255 at which point the program will continuously execute at location $FE0A_{16}$ because it is a jump to itself.

To verify this, depress: DISPLAY

RUN run

The program should now be "looping to self" at location $FE0A_{16}$. To verify this, depress:

H/S FE0A 10FF

Now examine the memory location corresponding to Register 0.

Depress F F 0 0 FE0A FF00

Depress EMA FF00 00FF

This illustrates that FF_{16} did become the final contents of WP0. Note that, when the program was being entered into RAM, EMDI was used rather than EMD because of the rather desirable feature of automatic address incrementing. The advantage of using EMD is that the actual contents of the addressed memory location are displayed after key depression (echoed back after being entered).

D.5.2 EXAMPLE 2, HEXADECIMAL TO DECIMAL CONVERSIONS

Convert 8000_{16} to a decimal number

Depress	CLR	
Depress	H→D	
Depress	8 0 0 0	8000
Depress	H→D	−3 2768

Convert 0020_{16} to a decimal number

Depress	CLR	
Depress	H→D	
Depress	2 0	20
Depress	H→D	32

D.5.3 EXAMPLE 3, DECIMAL TO HEXADECIMAL CONVERSIONS

Convert 45_{10} to hex

Depress	CLR	
Depress	D→H	
Depress	4 5	45
Depress	D→H	2D

Convert -1024_{10} to hex

Depress	CLR	
Depress	D→H	
Depress	F/− 1 0 2 4	− 1024
Depress	D→H	FC00

D.5.4 EXAMPLE 4, ENTER VALUE ON CRU

Send a bit pattern to the CRU at CRU address (bits 3 to 14 of R12) $0E0_{16}$ with a bit count of 9 containing a value of 5 (000000101_2).

Depress	CLR	
Depress	9 0 E 0	90E0
Depress	DCRU	90E0 YYYY
Depress	0 0 0 5	90E0 0005
Depress	ECRU	

YYYY indicates value at the current CRU address. Note that a DCRU operation is always required to specify bit count/CRU address.

D.5.5 EXAMPLE 5. ENTER, VERIFY VALUE AT MEMORY ADDRESS

Enter 0040_{16} into location FE20 and verify that it got there.

Depress	CLR	
Depress	F E 2 0	FE20
Depress	EMA	FE20 xxxx
Depress	0 0 4 0	FE20 0040
Depress	EMD	FE20 0040

The contents of address FE20 are verified by an echo of data from memory to display following the pressing of EMD. If it is desired to view and enter data at address FE22, depress EMD.

APPENDIX E

PARTS LIST

TABLE E-1. TM 990/1481 CONTROLLER PARTS LIST

Symbol	Description
C01-C05	Capacitor, tantalum, 47 uFd
C08-C47,C50	Capacitor, ceramic, 0.100 uFd
CR1	Diode, TI 1N914B
E01,E02,E06-E16	Jumper pins, 0.025 inch square
J2	Connector, 25-pin (AMP 206584-2) Jumper plug is Berg P/N 65474-005; order from Berg Electronics, Inc. Rt. 3 New Cumberland, Pennsylvania 17070
Q01	Transistor, 2N2905A, PNP
R01,R02	Resistor, 68 ohm, 5%, 0.25 W
R05	Resistor, 2.7 K ohm, 5%, 0.5 W
R06,R07,R10	Resistor, 3.3 K ohm, 5%, 0.25 W
R08	Resistor, 330 ohm, 5%, 0.5 W
R09	Resistor, 560 ohm, 5%, 0.5 W
R11	Resistor, 33 K ohm, 5%, 0.25 W
R12	Resistor, 620 ohm, 5%, 0.25 W
R13	Resistor, 910 ohm, 5%, 0.25 W
R14	Resistor, 1 K ohm, 0.25 W
S01,S02	Toggle switch
S03	SPDT switch
U01,U02	IC, SN74S373
U03	IC, 74S478, PROM, microcode #9
U04	IC, 74S478, PROM, microcode #10
U05	IC, 74S330, PLA entry point decode (PLA 1)
U06	IC, 74S478, PROM, microcode #7
U07	IC, 74S478, PROM, microcode #8
U08,U10,U15,U18, U24,U32,U37	IC, SN74S374N
U09	IC, 74S330, PLA entry point decode (PLA 2A)
U12	IC, 74S330, PLA entry point decode (PLA 2B)
U13	IC, 74S478, PROM, microcode #5
U14	IC, 74S478, PROM, microcode #6
U17	IC, 74S330, PLA entry point decode (PLAIR)
U20,U25,U64	IC, SN74LS244N
U21	IC, 74S478, PROM, microcode #3
U22	IC, 74S478, PROM, microcode #4
U23,U48,U53,U59	Network, SN74S174N
U26	IC, 74S478, PROM, microcode #1
U27	IC, 74S478, PROM, microcode #2

TABLE E-1. TM 990/1481 CONTROLLER BOARD PARTS LIST (Concluded)

Symbol	Description
U28,U72,U73	Network, SN74S32N
U29	Network, SN74LS00N
U30,U34,U35	IC, SN74LS257N
U31,U36	IC, SN74LS374N
U33	Network, SN74S260N
U38	Network, SN74S139N
U39,U55,U74,U75,U97	Network, SN 74S04N
U40,U46,U71,U86,U90	Network, SN74S74N
U41-U43,U93,U98	Network, SN74S251
U44,U45	IC, SN74S08
U47	IC, SN74LS259N
U49	ROM adapter plug
U50,U77	Network, SN74S00N
U51,U82,U87	Network, SN74S175N
U52	IC, 74S288, PROM, microcode register decode
U54	IC, UA 7905C
U56,U69,U70	IC, SN74S38N
U57	Network, SN74LS175N
U60,U61,U67	SIP, resistor, 1 K ohms, 2%
U62	Network, resistor, 4.700 K ohms
U63	IC, 74S287, PROM, CRU decode
U66	IC, SN74S241
U68	Network, SN74LS279N
U76,U83	Network, SN74LS08N
U78	IC, TMS 9902, communications controller
U79	Network, SN75189AN
U80	Network, resistor, 680 ohms, 1 W, 2%
U81	Network, SN74LS132N
U84	Network, SN74S112N
U85	IC, SN75188N
U88	Network, SN74LS74N
U89	Network, SN74LS164N
U91,U96	Network, SN74S163N
U92	Network, SN74S64N
U94	Memory adapter plug
U95	Oscillator, crystal, 30 MHz
U99	Memory adapter plug

TABLE E-2. CABLE ASSEMBLY CONNECTING P3/P3 AND P4/P4

Symbol	Description
P1,P2	40-pin edge connector 1½ inches flat 40-conductor cable, 2 inches wide

TABLE E-3. TM 990/1481 PROCESSOR PARTS LIST

Symbol	Description
C01,C02	Capacitor, tantalum, 47 uFd, 10%, 20 V
C03-C57	Capacitor, ceramic, 0.100 uFd, 10%, 100 V
E1,E2,E4-E6	Pin, 0.025 square Jumper plug is Berg P/N 65474-005; order from Berg Electronics, Inc. Rt. 3 New Cumberland, Pennsylvania 17070
R01,R02,R05	Resistor, 1.0 K ohms, 5%, 0.25 W
R03	Resistor, 220 ohms, 5%, 0.25 W
R04	Resistor, 330 ohms, 5%, 0.25 W
U01,U05,U26,U31,U56, U61,U82,U87	IC, SN74S241N
U02,U16,U47	Network, SN74S08N
U03,U04,U28,U29	Network, SN74LS253N
U06,U32,U62,U88	IC, SN74S481, 4-bit microprocessor
U07,U33,U63,U89	IC, SN74S189N, 4 X 16 static ram
U08,U34,U58,U84	Network, SN74S257N
U09,U35,U64,U90	IC, SN74LS379N
U10,U12,U13,U39,U40, U50,U66,U68 U91,U93,U95	IC, SN74LS244N
U11,U38,U67,U94	IC, SN74S240
U14	IC, 74S288, PROM, microinstruction decode
U15,U42	Network, SN74S74N
U17	IC, 74S288, PROM, status control
U18,U41,U49,U53,U55	Network, SN74S04N
U20	IC, 74S288, PROM, microinstruction decode
U21,U76	IC, SN74LS125N
U22	Network, SN74S20N
U23	IC, 74S288, PROM, status control
U24	Network, SN74LS280N
U27	Network, SN74S138N
U37	IC, SN74S374N
U43,U65	Network, SN74S10N
U44,U48	Network, SN74S00N
U45	IC, TMS 9901
U46,U52	Network, SN74S32N
U51	Resistor, fixed array, 4700 ohms
U54	Network, SN74S64N
U57	Network, SN74S182N
U59	IC, SN74S09N
U60	Network, resistor, 560 ohms, 2%, 1 W
U69	Network, SN74LS175N
U70,U77	Network, SN74LS251N
U71,U72	Network, SN74LS08N
U73	Network, resistor, 4700 ohms
U74,U78	Network, SN74LS04N
U79	Network, SN74LS32N
U83	Network, SN74LS163N
U85	Network, SN74LS85N

APPENDIX F

TM 990/1481 PROCESSOR AND CONTROLLER SCHEMATICS

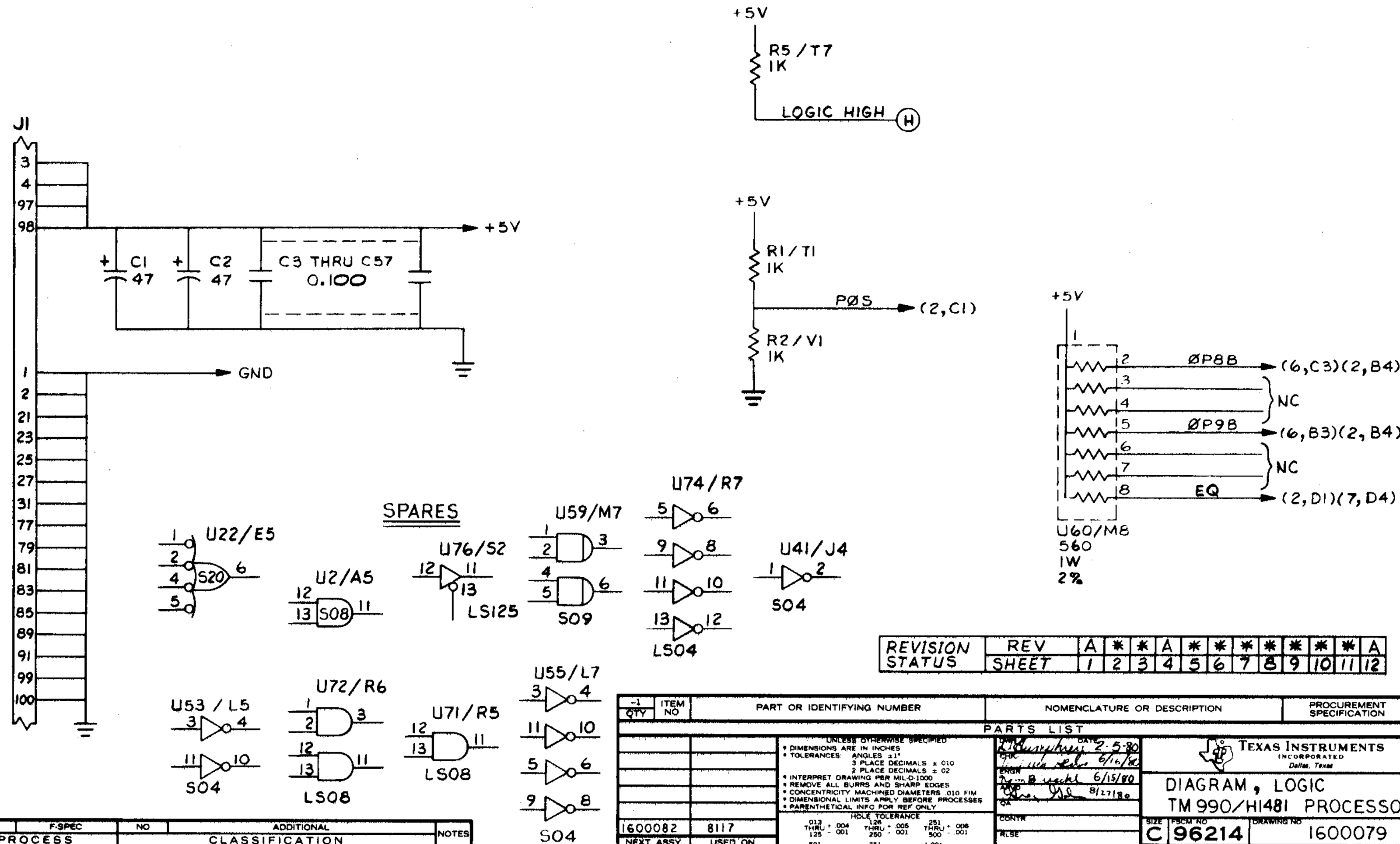
NOTES, UNLESS OTHERWISE SPECIFIED:

1. ALL RESISTANCE VALUES ARE IN OHMS
2. ALL CAPACITANCE VALUES ARE IN MICROFARADS
3. ALL RESISTORS ARE .25W, 5%
4. J1 IS MARKED P1 ON THE SILKSCREEN
5. ALL IC DEVICE TYPES ARE 74XXXXX
   EXAMPLE: LS125 IS DEVICE TYPE 74LS125

[6] USER'S OPTION

[7] NOT INSTALLED

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED
A	CN436255 INF	8-25-80	



REVISION STATUS	REV	A	*	*	A	*	*	*	*	*	*	*	A
	SHEET	1	2	3	4	5	6	7	8	9	10	11	12

PARTS LIST

QTY	ITEM NO	PART OR IDENTIFYING NUMBER	NOMENCLATURE OR DESCRIPTION	PROCUREMENT SPECIFICATION	NOTES

NEXT ASSY	USED ON
1600082	8117

APPLICATION

UNLESS OTHERWISE SPECIFIED
- DIMENSIONS ARE IN INCHES
- TOLERANCES: ANGLES ±1°, 3 PLACE DECIMALS ± .010, 2 PLACE DECIMALS ± .02
- INTERPRET DRAWING PER MIL-D-1000
- REMOVE ALL BURRS AND SHARP EDGES
- CONCENTRICITY MACHINED DIAMETERS .010 FIM
- DIMENSIONAL LIMITS APPLY BEFORE PROCESSES
- PARENTHETICAL INFO FOR REF ONLY

TEXAS INSTRUMENTS INCORPORATED, Dallas, Texas

DIAGRAM, LOGIC
TM 990/H1481 PROCESSOR

SIZE C | FSCM NO 96214 | DRAWING NO 1600079

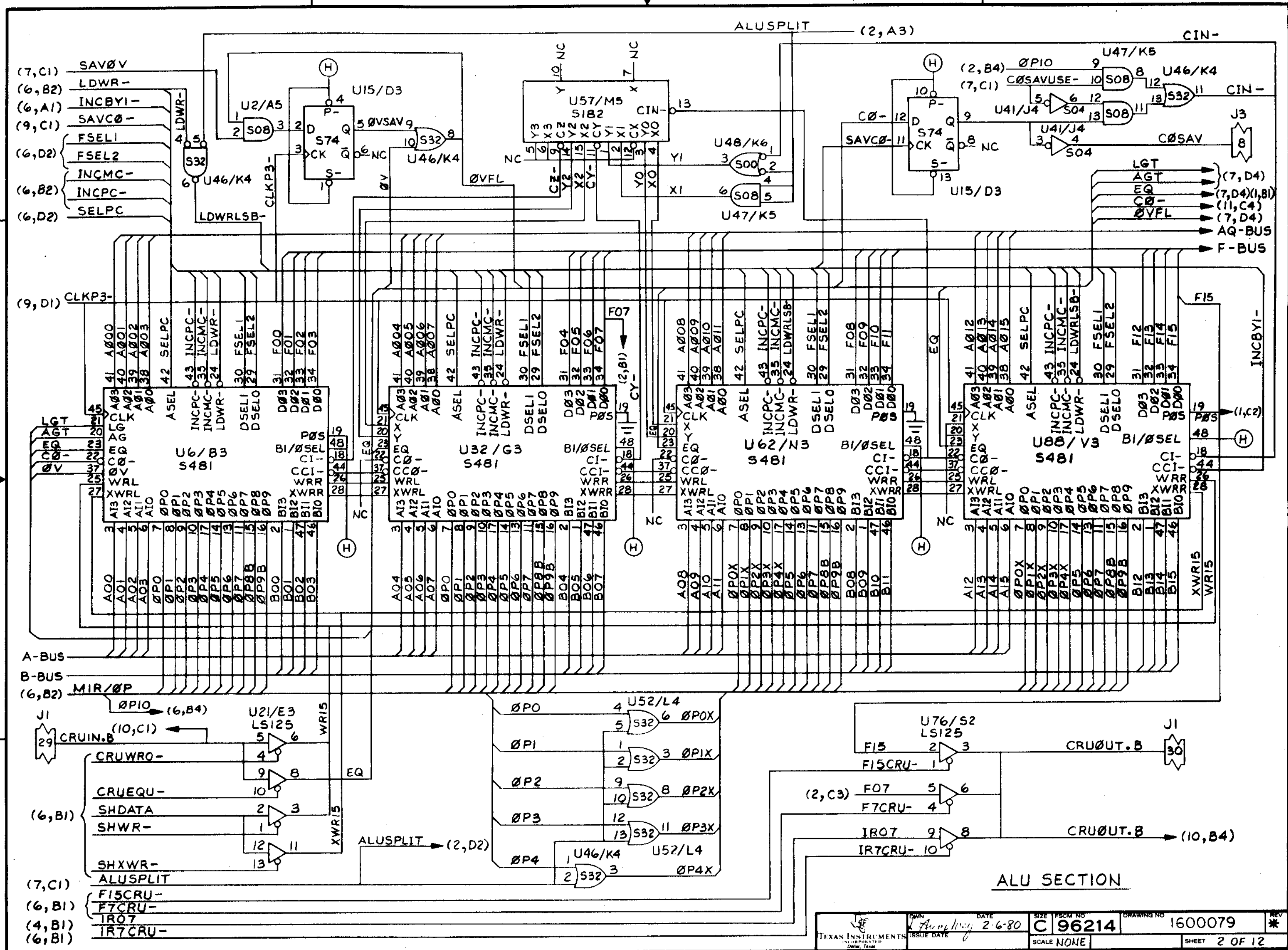
SCALE NONE | SHEET 1 OF 12

SEQ NO	IDENT	F-SPEC	NO	ADDITIONAL	NOTES
	PROCESS		CLASSIFICATION		

PROCESSES — FOR CORRELATION TO GOVT/IND SPECIFICATIONS, SEE TI DRAWING 729467

TM 990/1481 PROCESSOR SCHEMATIC

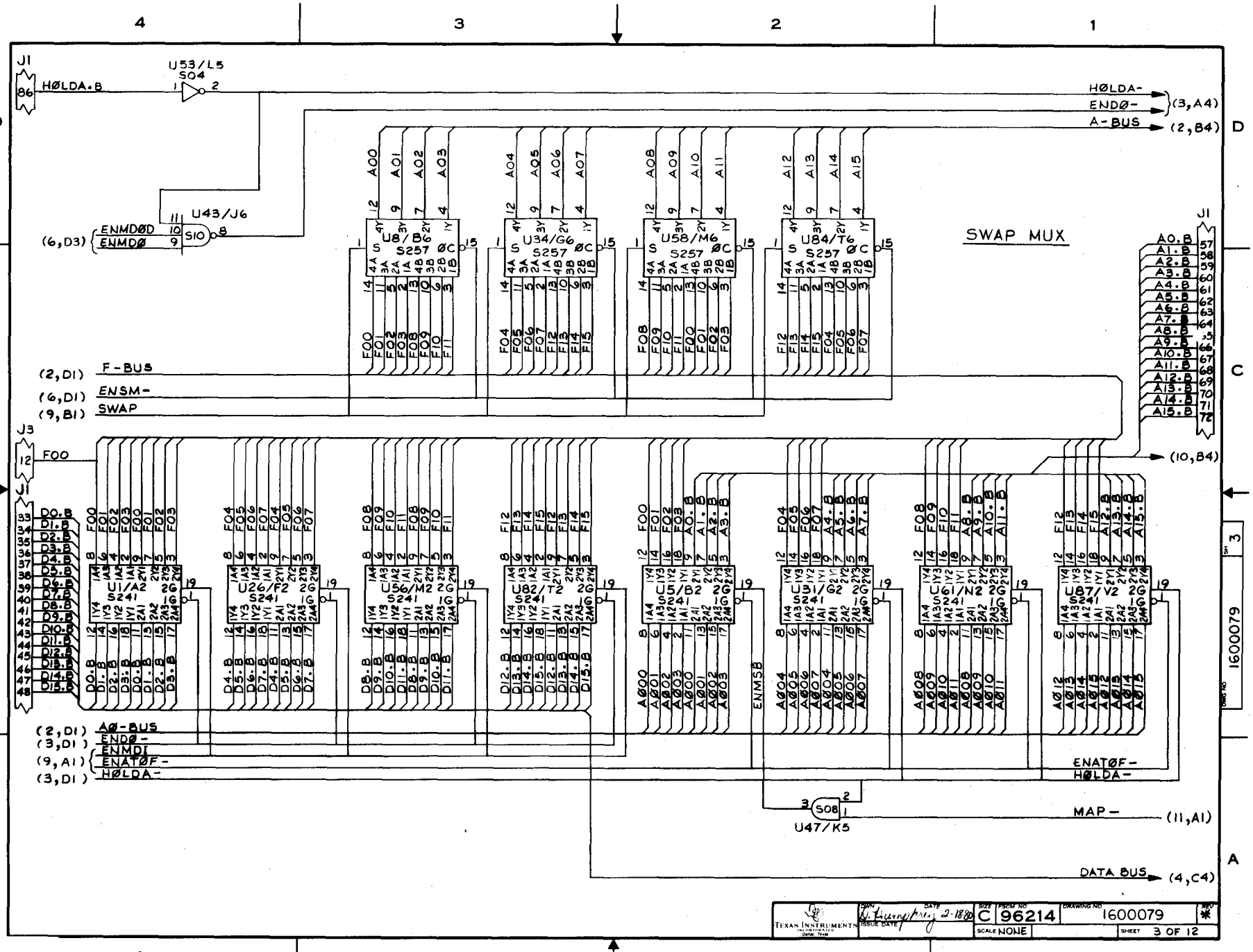
1600079



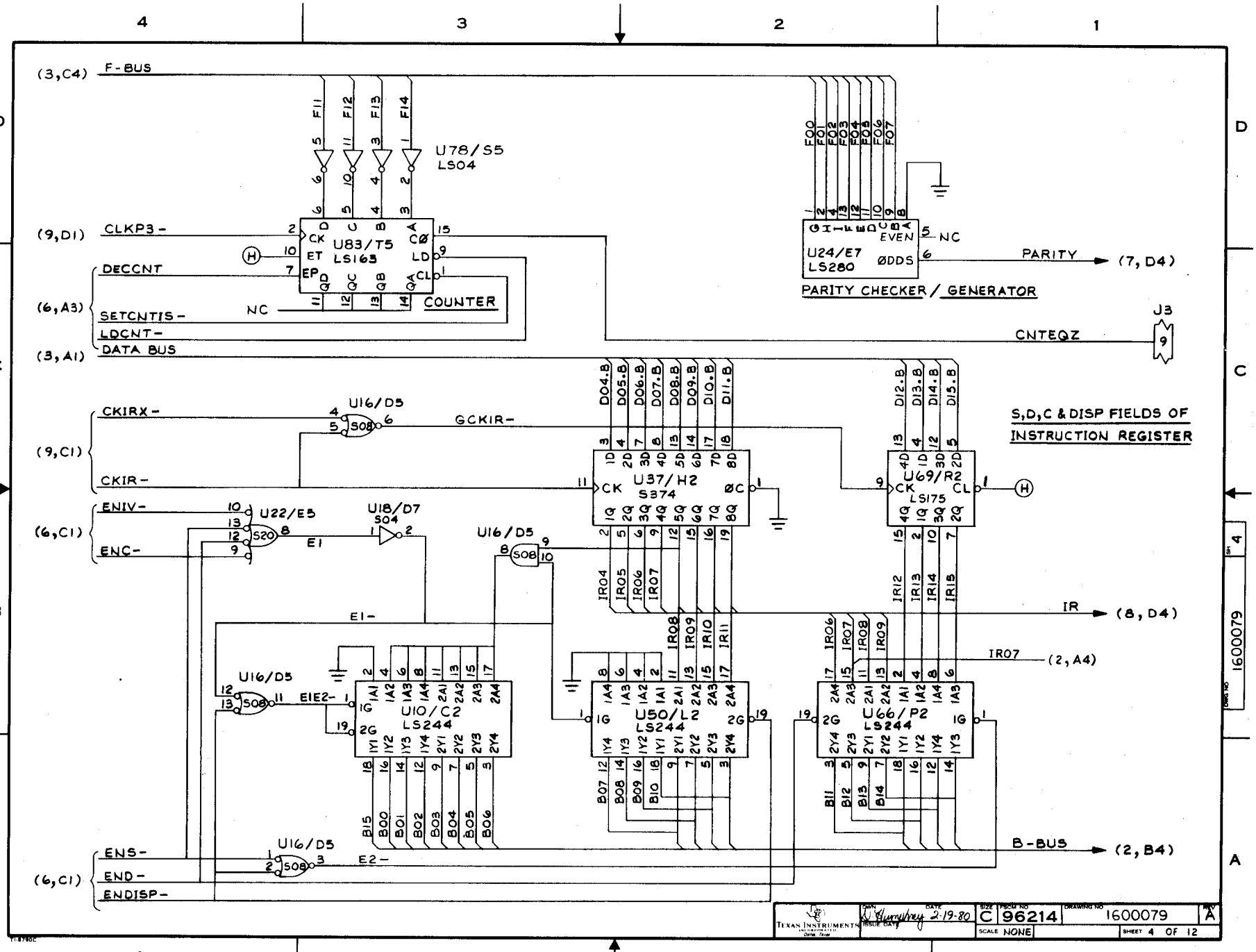
Texas Instruments
DWN: 2-6-80
SIZE C | FSCM NO 96214 | DRAWING NO 1600079 | REV *
SCALE NONE | SHEET 2 OF 12

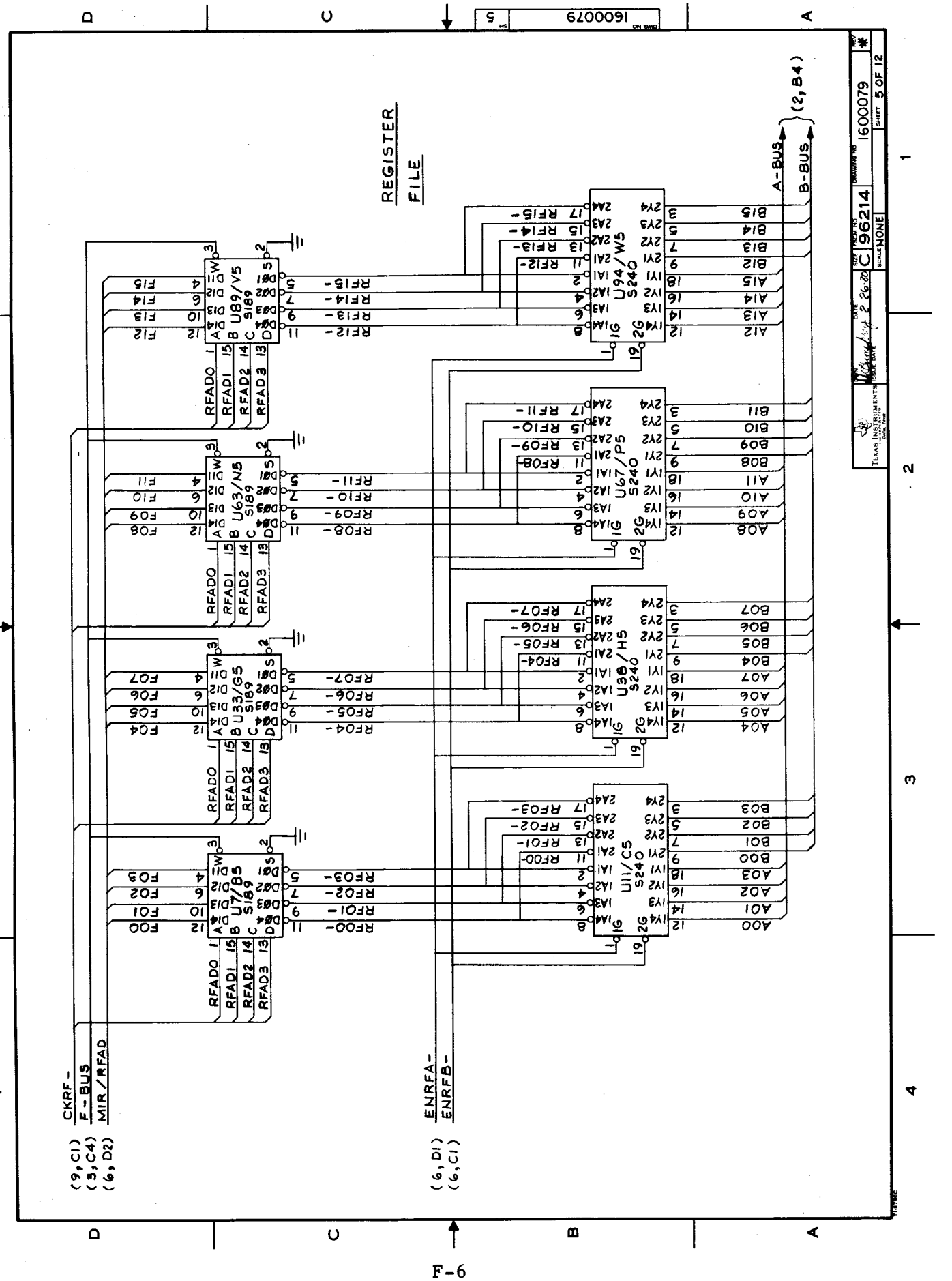
DWG NO 1600079 SH 2

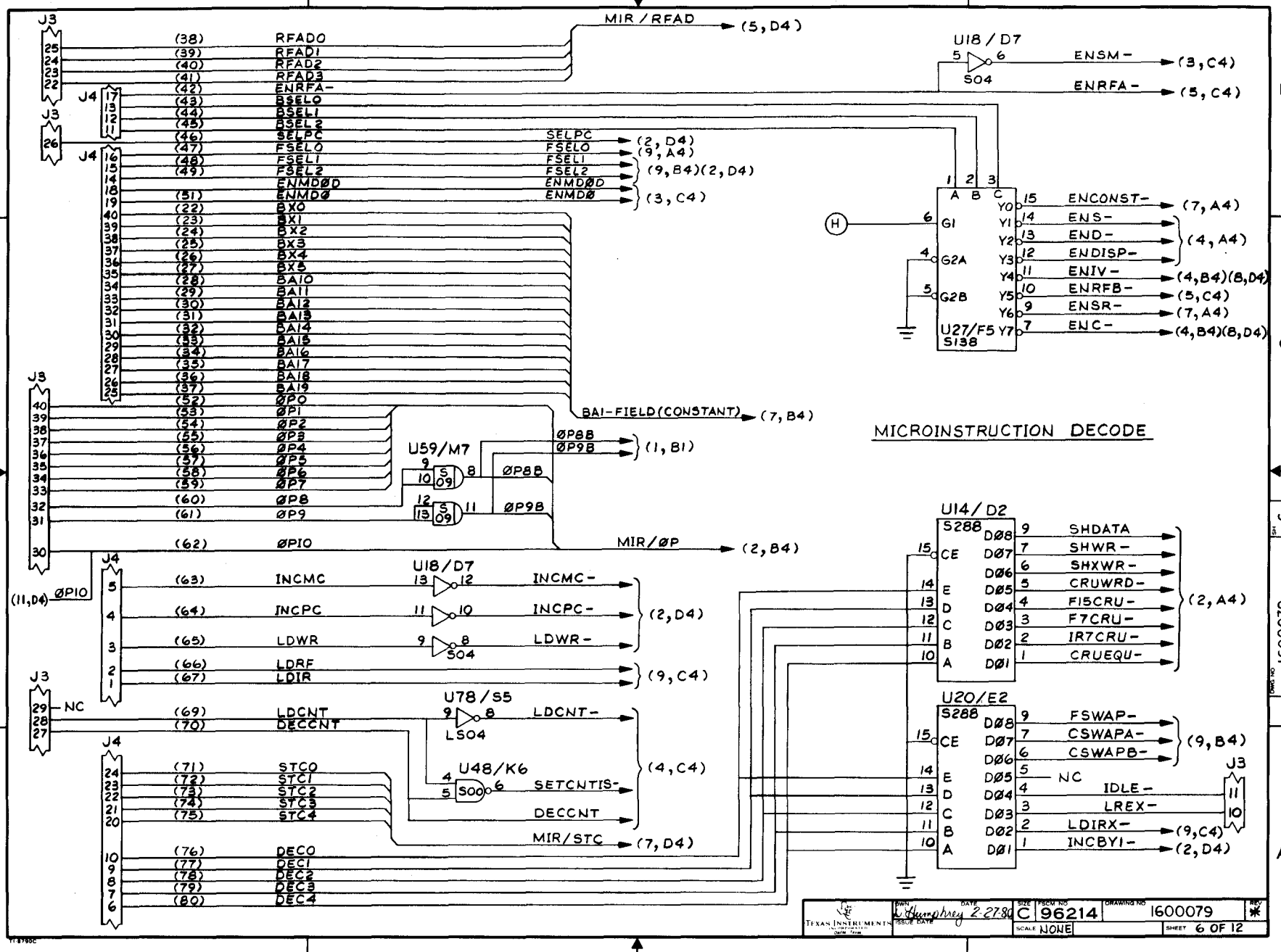
TM 990/1481 PROCESSOR SCHEMATIC

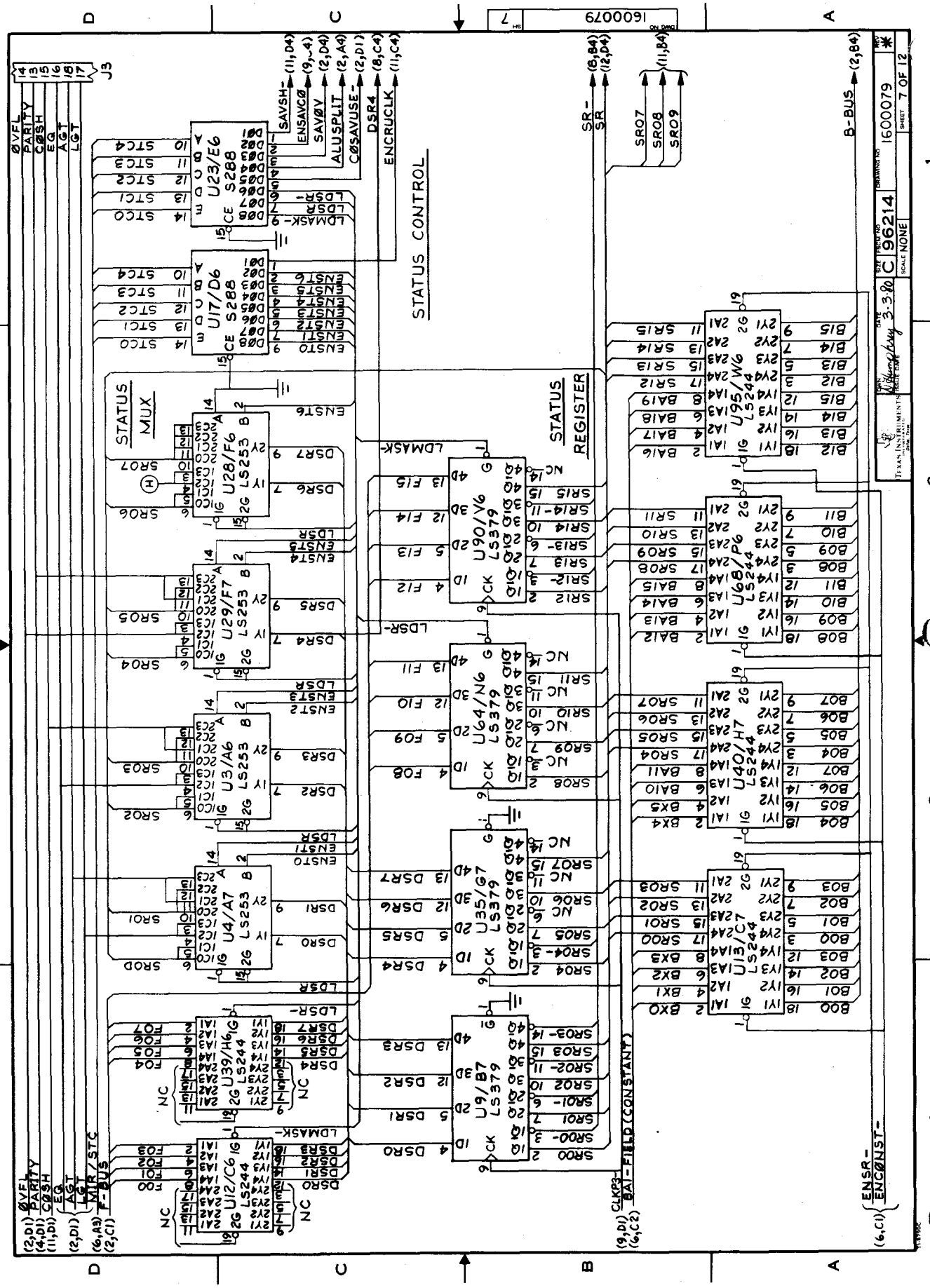


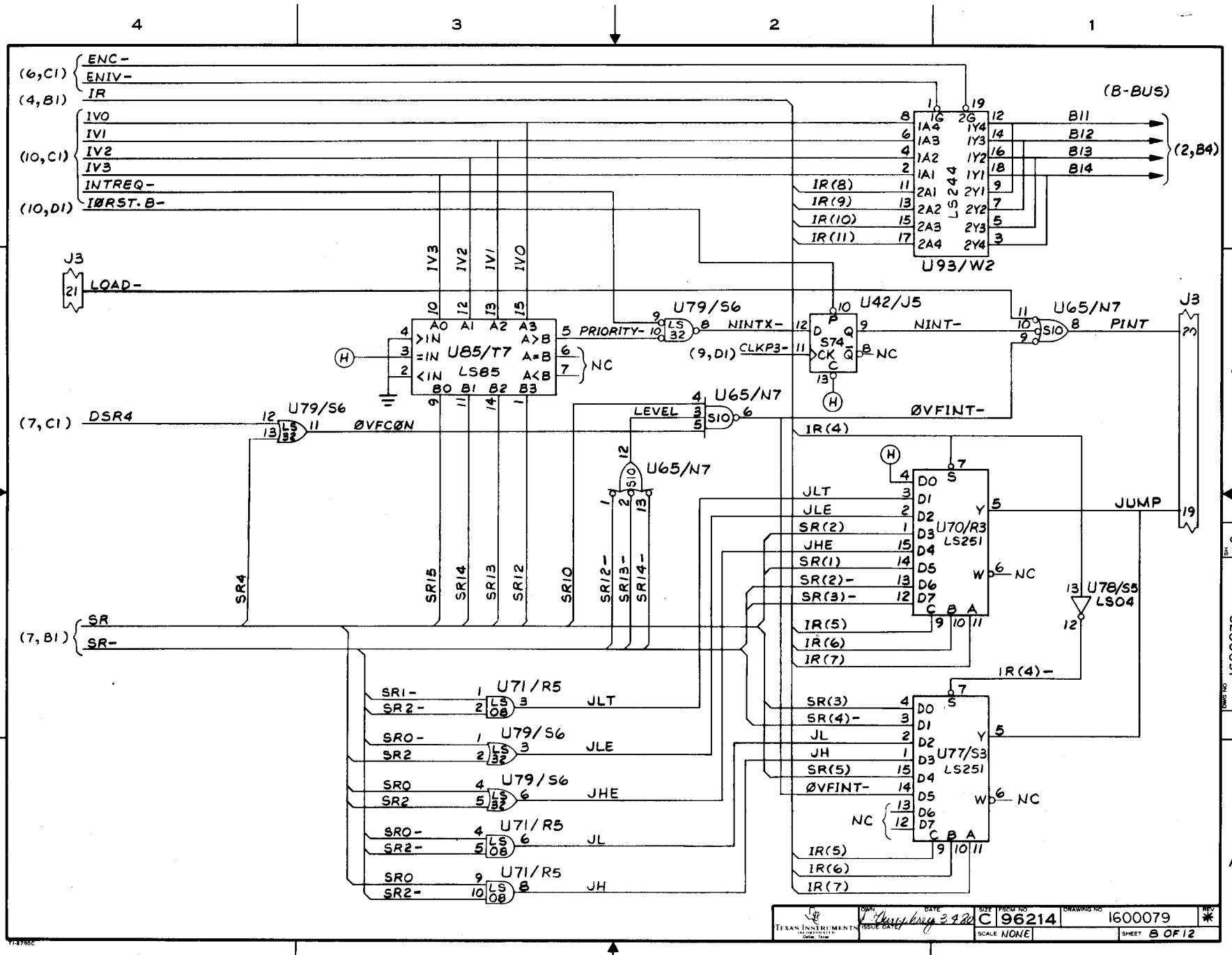
TM 990/1481 PROCESSOR SCHEMATIC

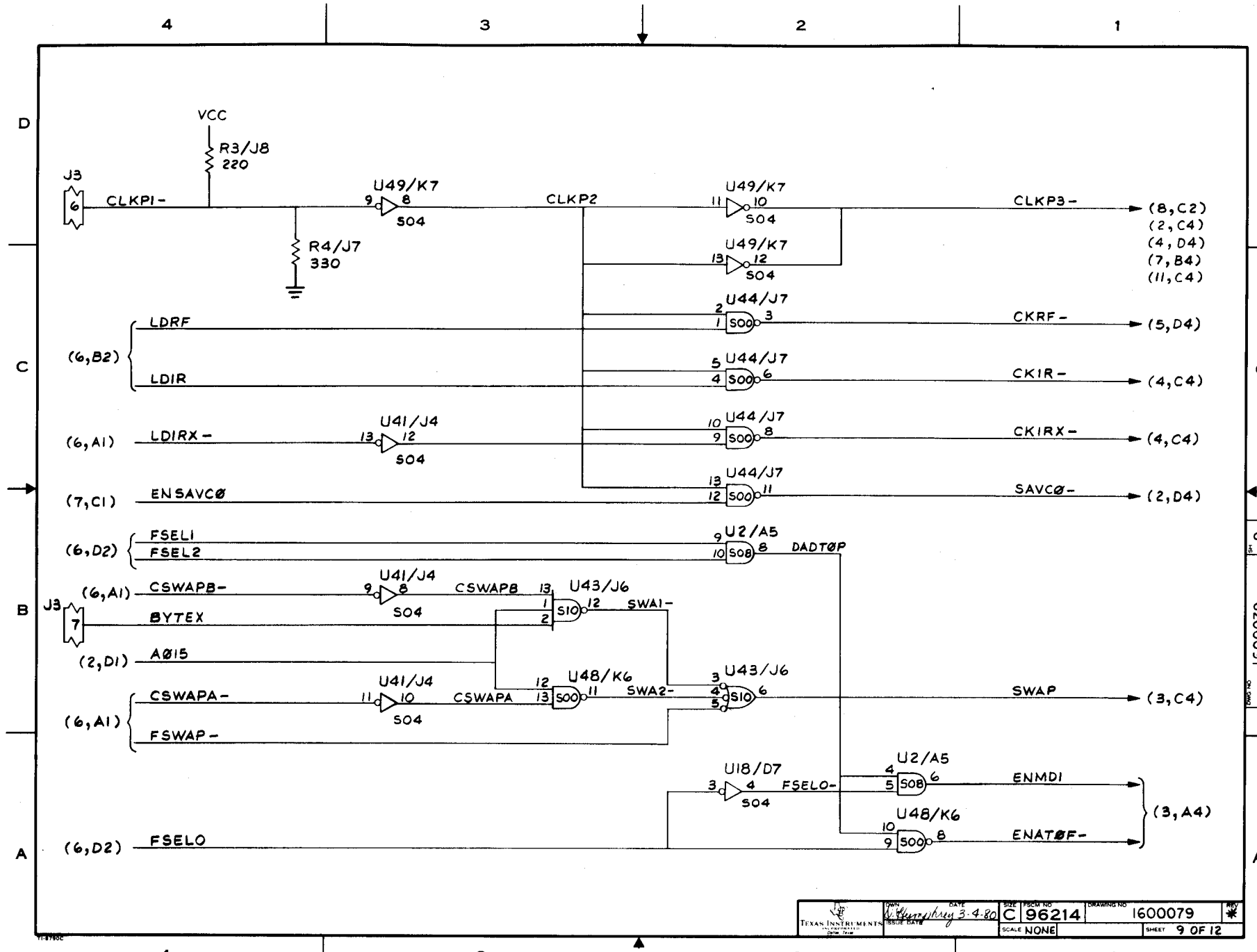

(3,C4) F-BUS
U78/S5 LS04
(9,D1) CLKP3-
DECCNT
(6,A3) SETCNTIS-
LDCNT-
U83/T5 LS163 COUNTER
NC
(3,A1) DATA BUS
U24/E7 LS280 PARITY
PARITY CHECKER / GENERATOR
(7, D4)
J3
CNTEQZ
S,D,C & DISP FIELDS OF INSTRUCTION REGISTER
CKIRX-
(9,C1) CKIR-
U16/D5 S08
GCKIR-
U37/H2 S374
U69/R2 LS175
ENIV-
(6,C1) ENC-
U22/E5 S20
U18/D7 S04
E1
E1-
U16/D5 S08
IR
(8, D4)
IR07
(2, A4)
U16/D5 S08
E1E2-
U10/C2 LS244
U50/L2 LS244
U66/P2 LS244
B-BUS
(2, B4)
ENS-
(6,C1) END-
ENDISP-
U16/D5 S08
E2-
TEXAS INSTRUMENTS
DRWN K. Humphrey 2-19-80
SIZE C
FSCM NO 96214
DRAWING NO 1600079
REV A
SCALE NONE
SHEET 4 OF 12
TM 990/1481 PROCESSOR SCHEMATIC
1600079
SH 4
F-5


REGISTER FILE
A-BUS
B-BUS
(2, B4)
U94/W5 S240
U67/P5 S240
U38/H5 S240
U11/C5 S240
B U89/V5 S189
B U63/N5 S189
B U33/G5 S189
B U7/B5 S189
RFADO
RFAD1
RFAD2
RFAD3
CKRF-
F-BUS
MIR/RFAD
(9, C1)
(3, C4)
(6, D2)
ENRFA-
ENRFB-
(6, D1)
(6, C1)
1600079
C 96214
SCALE NONE
SHEET 5 OF 12
TEXAS INSTRUMENTS


MICROINSTRUCTION DECODE
J3
J4
U18 / D7
S04
U59/M7
S09
U27/F5
S138
U14 / D2
S288
U20/E2
U78/S5
LS04
U48/K6
S00
MIR / RFAD
(5,D4)
ENSM-
(3,C4)
ENRFA-
(5,C4)
SELPC
FSEL0
FSEL1
FSEL2
ENMDØD
ENMDØ
(2,D4)
(9,A4)
(9,B4)(2,D4)
(3,C4)
ENCONST-
(7,A4)
ENS-
END-
ENDISP-
(4,A4)
ENIV-
(4,B4)(8,D4)
ENRFB-
(5,C4)
ENSR-
(7,A4)
ENC-
(4,B4)(8,D4)
BAI-FIELD(CONSTANT)
(7,B4)
ØP8B
ØP9B
(1,B1)
MIR/ØP
(2,B4)
INCMC-
INCPC-
LDWR-
(2,D4)
LDRF
LDIR
(9,C4)
LDCNT-
DECCNT
SETCNTIS-
(4,C4)
MIR/STC
(7,D4)
SHDATA
SHWR-
SHXWR-
CRUWRD-
F15CRU-
F7CRU-
IR7CRU-
CRUEQU-
(2,A4)
FSWAP-
CSWAPA-
CSWAPB-
(9,B4)
IDLE-
LREX-
LDIRX-
(9,C4)
INCBYI-
(2,D4)
(11,D4) ØPIO
NC
RFAD0
RFAD1
RFAD2
RFAD3
ENRFA-
BSEL0
BSEL1
BSEL2
SELPC
FSEL0
FSEL1
FSEL2
ENMDØD
ENMDØ
BX0
BX1
BX2
BX3
BX4
BX5
BAI0
BAI1
BAI2
BAI3
BAI4
BAI5
BAI6
BAI7
BAI8
BAI9
ØP0
ØP1
ØP2
ØP3
ØP4
ØP5
ØP6
ØP7
ØP8
ØP9
ØPIO
INCMC
INCPC
LDWR
LDRF
LDIR
LDCNT
DECCNT
STC0
STC1
STC2
STC3
STC4
DEC0
DEC1
DEC2
DEC3
DEC4
Texas Instruments
DRWN L Humphrey 2-27-80
ISSUE DATE
SIZE C
FSCM NO 96214
DRAWING NO 1600079
REV *
SCALE NONE
SHEET 6 OF 12


STATUS CONTROL
STATUS MUX
STATUS REGISTER
U23/E6 S288
U17/D6 S288
U28/F6 LS253
U29/F7 LS253
U3/A6 LS253
U4/A7 LS253
U90/V6 LS379
U64/N6 LS379
U35/G7 LS379
U9/B7 LS379
U39/H6 LS244
U12/C6 LS244
U95/W6 LS244
U68/P6 LS244
U40/H7 LS244
U13/C7 LS244
B-BUS
SR-
SR
SR07
SR08
SR09
ENCRUCLK
DSR4
ALUSPLIT
CØSAVUSE-
SAVØV
ENSAVCØ
SAVSH-
LDMASK-
LDSR-
ENSTØ
ENST1
ENST2
ENST3
ENST4
ENST5
ENST6
ØVFL
PARITY
CØSH
EQ
AGT
LGT
MAR/STC
F-BUS
CLKP3
BAI-FIELD (CONSTANT)
ENSR-
ENCØNST-
DRAWING NO 1600079
SHEET 7 OF 12
SIZE C
CAGE CODE 96214
SCALE NONE
DATE 3-3-80
TEXAS INSTRUMENTS

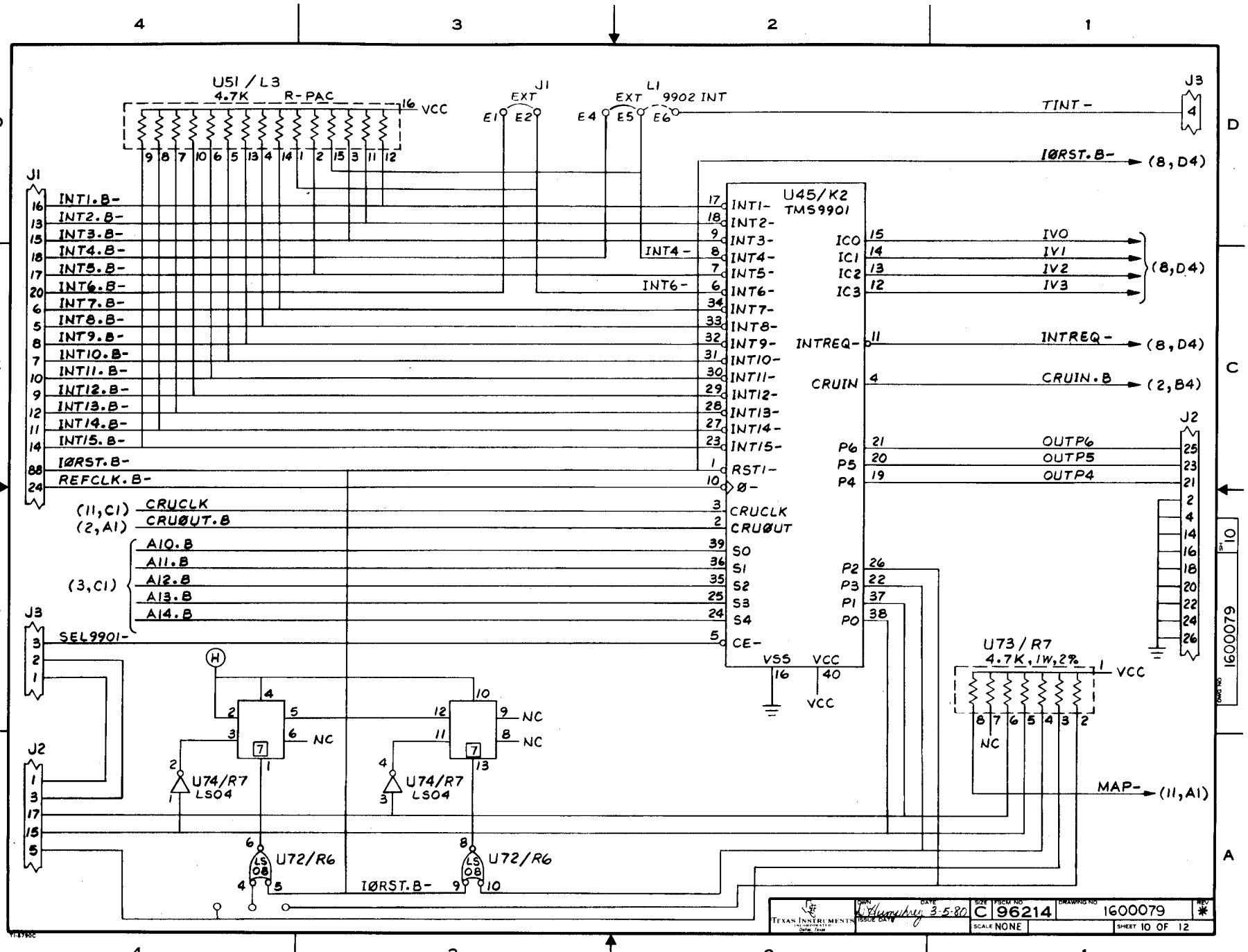

ENC-
ENIV-
IR
IV0
IV1
IV2
IV3
INTREQ-
IØRST.B-
(6,C1)
(4,B1)
(10,C1)
(10,D1)
(B-BUS)
B11
B12
B13
B14
(2,B4)
U93/W2
LS244
IR(8)
IR(9)
IR(10)
IR(11)
J3
LOAD-
U85/T7
LS85
U79/S6
LS32
PRIORITY-
NINTX-
U42/J5
S74
CLKP3-
(9,D1)
NC
U65/N7
S10
NINT-
PINT
ØVFINT-
LEVEL
DSR4
(7,C1)
ØVFCØN
IR(4)
U70/R3
LS251
JLT
JLE
SR(2)
JHE
SR(1)
SR(2)-
SR(3)-
JUMP
U78/S5
LS04
IR(5)
IR(6)
IR(7)
IR(4)-
U77/S3
SR(3)
SR(4)-
JL
JH
SR(5)
ØVFINT-
SR4
SR15
SR14
SR13
SR12
SR10
SR12-
SR13-
SR14-
SR
SR-
(7,B1)
U71/R5
LS08
SR1-
SR2-
SR0-
SR2
SR0
JL
JH
TEXAS INSTRUMENTS
C
96214
1600079
SCALE NONE
SHEET 8 OF 12
8
1600079

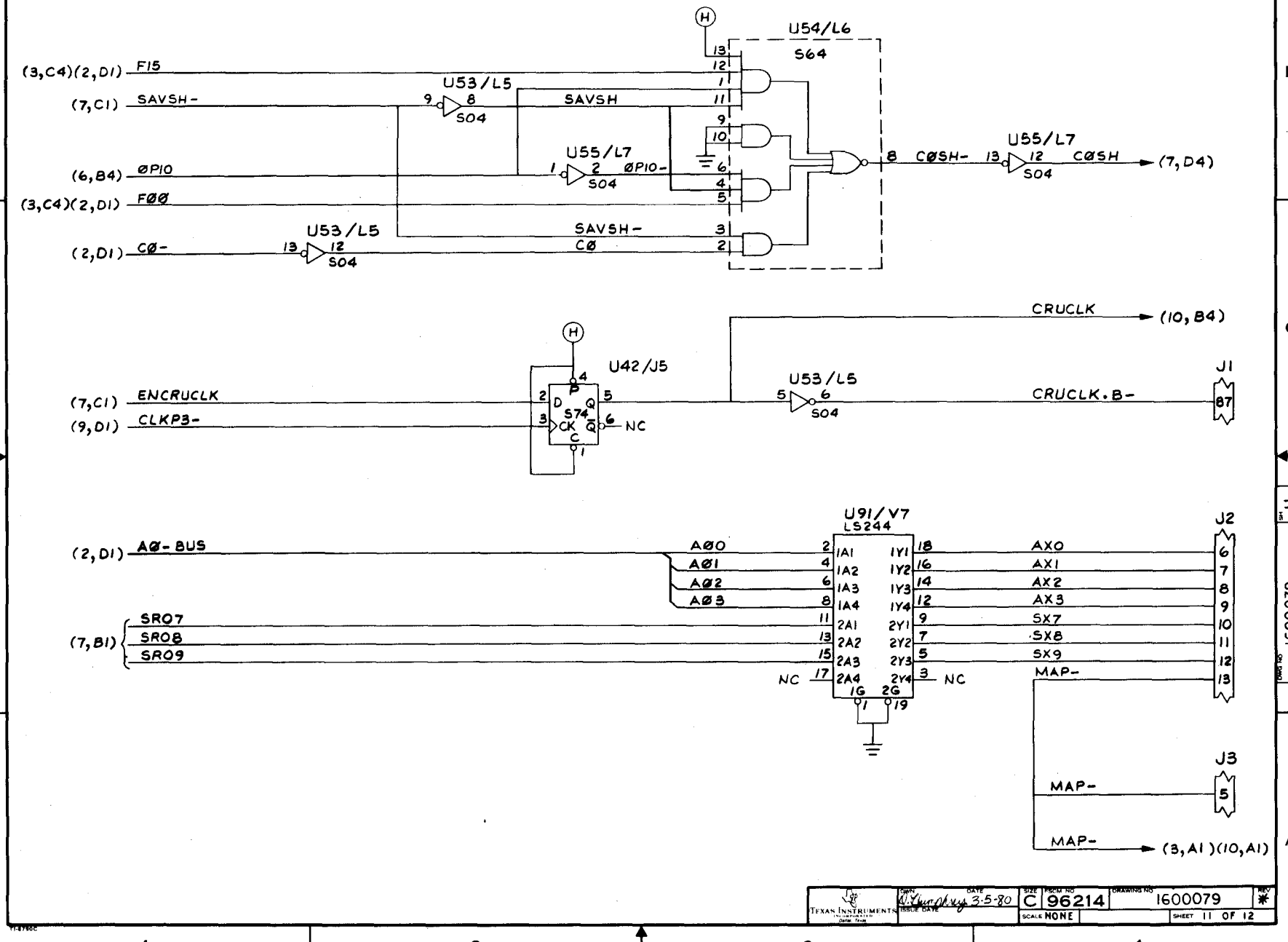


TM 990/1481 PROCESSOR SCHEMATIC

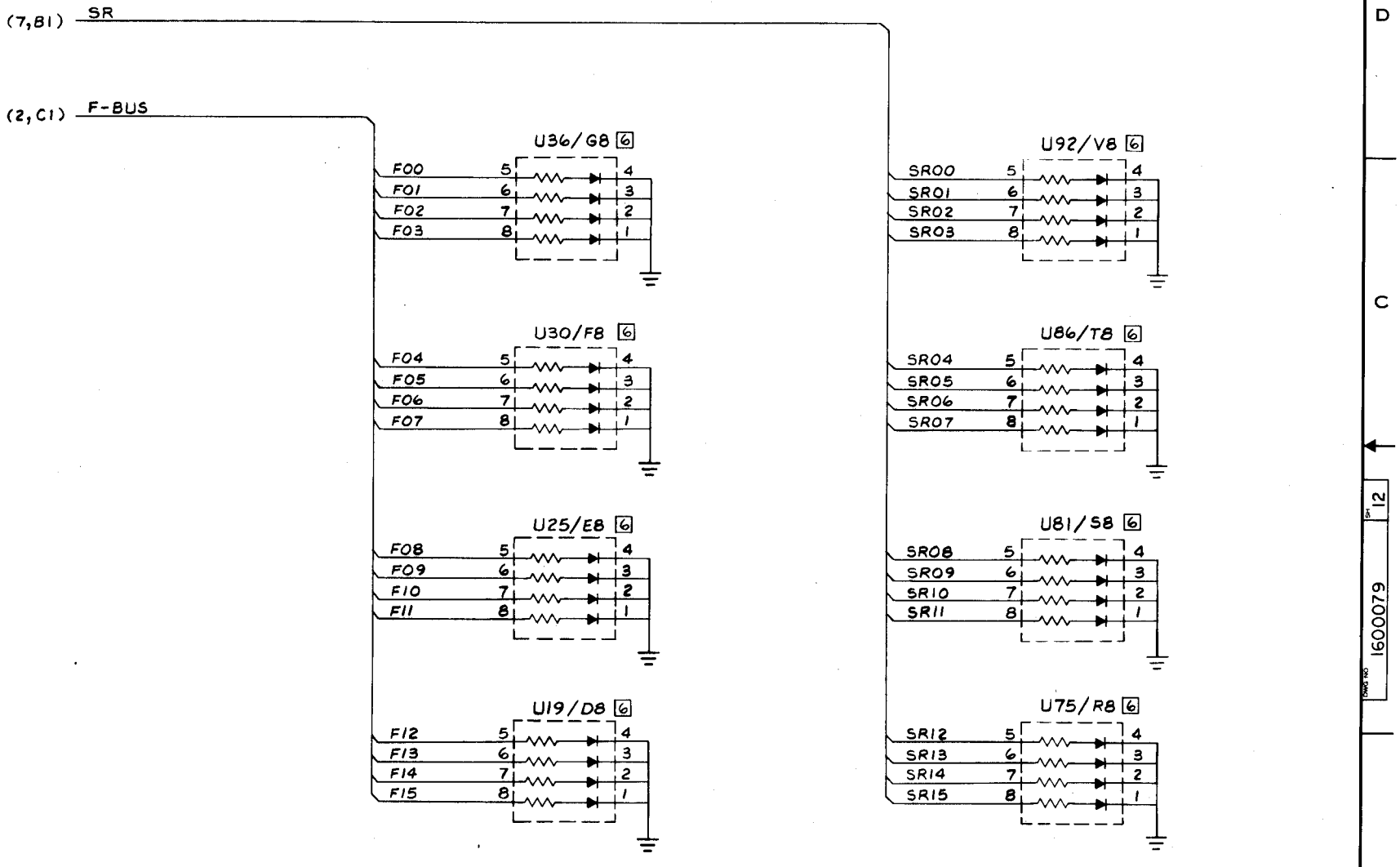
F-10

TEXAS INSTRUMENTS	DWN	DATE	SIZE	FSCM NO	DRAWING NO
	ISSUE DATE	3-4-80	C	96214	1600079
			SCALE NONE		SHEET 9 OF 12


U51 / L3
4.7K R-PAC
16 VCC
9 8 7 10 6 5 13 4 14 1 2 15 3 11 12
J1
EXT
E1 E2
L1
EXT 9902 INT
E4 E5 E6
J3
TINT -
4
IØRST.B- (8,D4)
U45/K2
TMS9901
INT1- 17
INT2- 18
INT3- 9
INT4- 8
INT5- 7
INT6- 6
INT7- 34
INT8- 33
INT9- 32
INT10- 31
INT11- 30
INT12- 29
INT13- 28
INT14- 27
INT15- 23
RST1- 1
Ø- 10
CRUCLK 3
CRUØUT 2
S0 39
S1 36
S2 35
S3 25
S4 24
CE- 5
ICO 15
IC1 14
IC2 13
IC3 12
IVO
IV1
IV2
IV3
(8,D4)
INTREQ- 11
INTREQ- (8,D4)
CRUIN 4
CRUIN.B (2,B4)
P6 21
P5 20
P4 19
OUTP6
OUTP5
OUTP4
P2 26
P3 22
P1 37
P0 38
VSS 16
VCC 40
VCC
INT1.B-
INT2.B-
INT3.B-
INT4.B-
INT5.B-
INT6.B-
INT7.B-
INT8.B-
INT9.B-
INT10.B-
INT11.B-
INT12.B-
INT13.B-
INT14.B-
INT15.B-
IØRST.B-
REFCLK.B-
(11,C1) CRUCLK
(2,A1) CRUØUT.B
(3,C1)
A10.B
A11.B
A12.B
A13.B
A14.B
SEL9901-
J2
J3
H
U74/R7
LS04
NC
U72/R6
LS 08
IØRST.B-
U73 / R7
4.7K, 1W, 2%
VCC
8 7 6 5 4 3 2
NC
MAP- (11,A1)
TEXAS INSTRUMENTS
DWN
DATE 3-5-80
ISSUE DATE
SIZE C
FSCM NO 96214
DRAWING NO 1600079
REV *
SCALE NONE
SHEET 10 OF 12
1600079
10
TM 990/1481 PROCESSOR SCHEMATIC
F-11



TEXAS INSTRUMENTS	DWN	DATE	SIZE	FSCM NO	DRAWING NO	REV
	W. Murphy	3-5-80	C	96214	1600079	*
	ISSUE DATE		SCALE NONE		SHEET 11 OF 12	



F-BUS LED DISPLAY

STATUS REGISTER LED DISPLAY

SUGGEST DIALIGHT #547-2007 LED'S IF LED'S ARE DESIRED

TEXAS INSTRUMENTS	DWN	DATE	SIZE	FSCM NO	DRAWING NO	REV
	ISSUE DATE 3-6-80		C	96214	1600079	A
			SCALE NONE		SHEET 12 OF 12	

NOTES, UNLESS OTHERWISE SPECIFIED:

1. ALL RESISTANCE VALUES ARE IN OHMS
2. ALL RESISTORS ARE .25W, 5%
3. ALL CAPACITANCE VALUES ARE IN MICROFARADS

[4] USER'S OPTION

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED
A	CN451539(C) INF	8-25-80	

U80/T1A
680

8	HOLD.B- (6,B4)
7	MES-
6	MES (6,C2)
5	READY-
4	ENUM- (6-D2)
3	NC
2	NC
1	VCC

U62/K1
4.7K

8	NC
7	NC
6	NC
5	PRES-
4	RESTART- (7,C4)
3	NC
2	NC
1	VCC

U60/P7
1K

8	NC
7	CMAX
6	CMAP (4,A1)
5	SW4 NC
4	SW4 NO
3	SW3 NC (9,C4)
2	SW3 NO
1	VCC

U61/P7A
1K

8	MAPP- (6,D3)
7	NC
6	NC
5	SW2 NC
4	SW2 NO
3	SW1 NC (9,D4)
2	SW1 NO
1	VCC

U67/R5
1K

8	NC
7	NC
6	NC
5	NC
4	NC
3	(H2) (1,B3)
2	(H3) (6,C4)(7,B2)(7,C2)(8,D4)(6,C2)(6,A3)(7,D3)(8,C2)(8,C3)
1	(H1) (3,D2)(8,C2)(9,A4)(9,D2)(6,C4)(6,A2)(8,A2)(8,C3)
	VCC

J1

3	VCC
4	
97	
98	
75	+12V
76	
73	-12V
74	
1	GND
2	
21	
23	
25	
27	
31	
77	
79	
81	
83	
85	
89	
91	
99	
100	

C1 47 20V
C2 47 20V
C8 - C47 .100
C3 47 20V
U54/P1
C4 47 20V
UA7905CKC
C5 47 20V
C50 .100

U49/N2A

(H2)	
1	
2	
3	+12V
4	
5	CMAP
6	
7	-5V
VCC	14
	13
	12
	11
	10
	9
	8

CMAX
ROM P18
ROM P19
ROM P20
ROM P21
TO CONTROL MEMORY ROMS (2,C4)

ROM ADAPTER PLUG

REVISION STATUS											
REV	A	*	A	A	*	A	A	A	A	A	
SHEET	1	2	3	4	5	6	7	8	9	10	

SEQ NO	IDENT	F-SPEC	NO	ADDITIONAL	NOTES
	PROCESS			CLASSIFICATION	

PROCESSES — FOR CORRELATION TO GOVT/IND SPECIFICATIONS, SEE TI DRAWING 729467

-1 QTY	ITEM NO	PART OR IDENTIFYING NUMBER	NOMENCLATURE OR DESCRIPTION	PROCUREMENT SPECIFICATION	NOTES
			PARTS LIST		

UNLESS OTHERWISE SPECIFIED
* DIMENSIONS ARE IN INCHES
* TOLERANCES ANGLES ±1°
3 PLACE DECIMALS ± .010
2 PLACE DECIMALS ± .02
* INTERPRET DRAWING PER MIL-D-1000
* REMOVE ALL BURRS AND SHARP EDGES
* CONCENTRICITY MACHINED DIAMETERS .010 FIM
* DIMENSIONAL LIMITS APPLY BEFORE PROCESSES
* PARENTHETICAL INFO FOR REF ONLY

HOLE TOLERANCE

.013 THRU .004 / .125 .001	.126 THRU .005 / .250 .001	.251 THRU .008 / .500 .001	
.501 THRU .008 / .750 .001	.751 THRU .010 / 1.000 .001	1.001 THRU .012 / 2.000 .001	

1600091	8117
NEXT ASSY	USED ON
APPLICATION	

DRAWN 6/16/80
CHK'D 6/16/80
ENGR 6/16/80
QA 6/17/80
CONTR
RLSE

TEXAS INSTRUMENTS INCORPORATED Dallas, Texas

DIAGRAM, LOGIC
TM 990/H1481 CONTROLLER

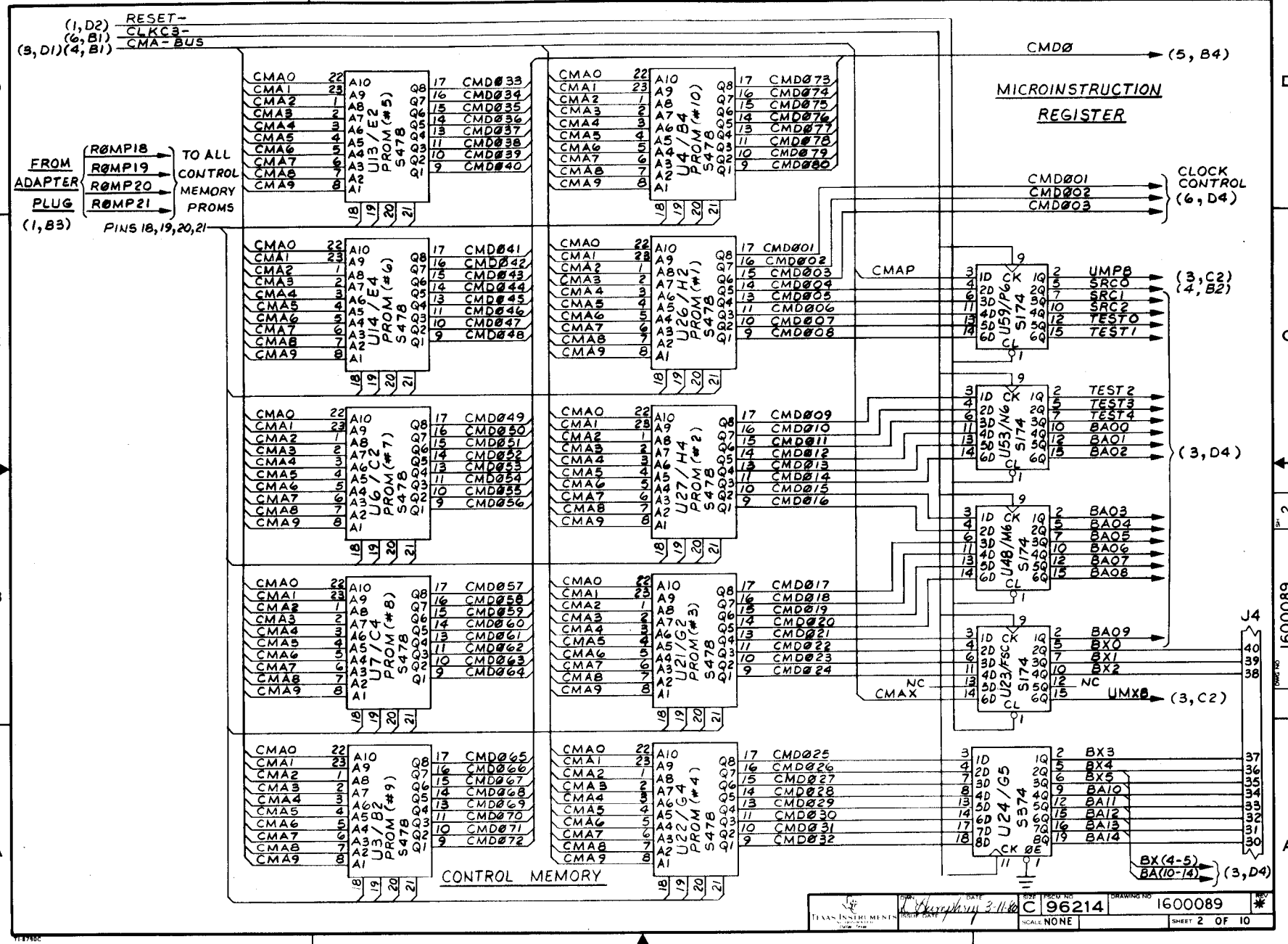
SIZE	FSCM NO	DRAWING NO
C	96214	1600089

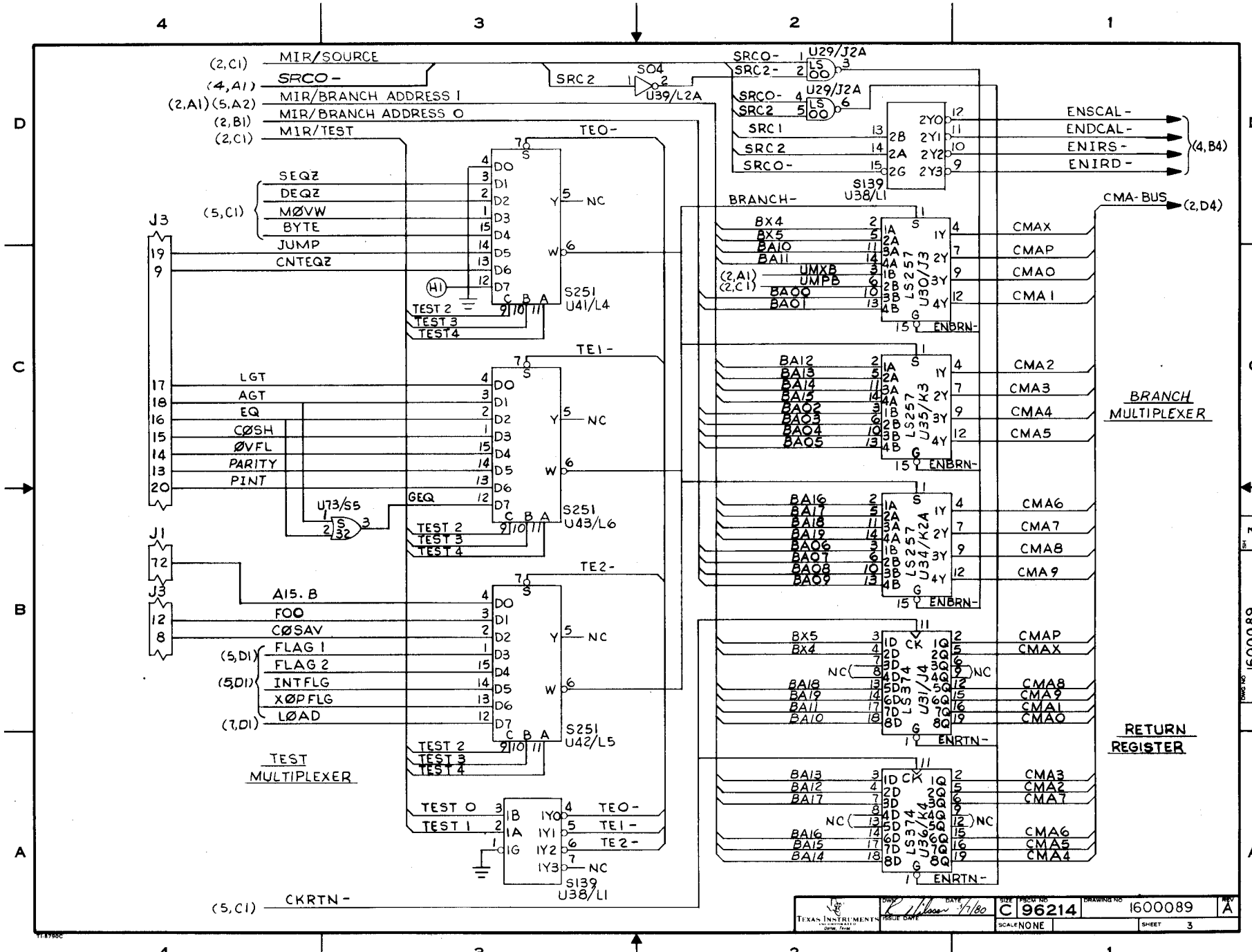
SCALE NONE SHEET 1 OF 10

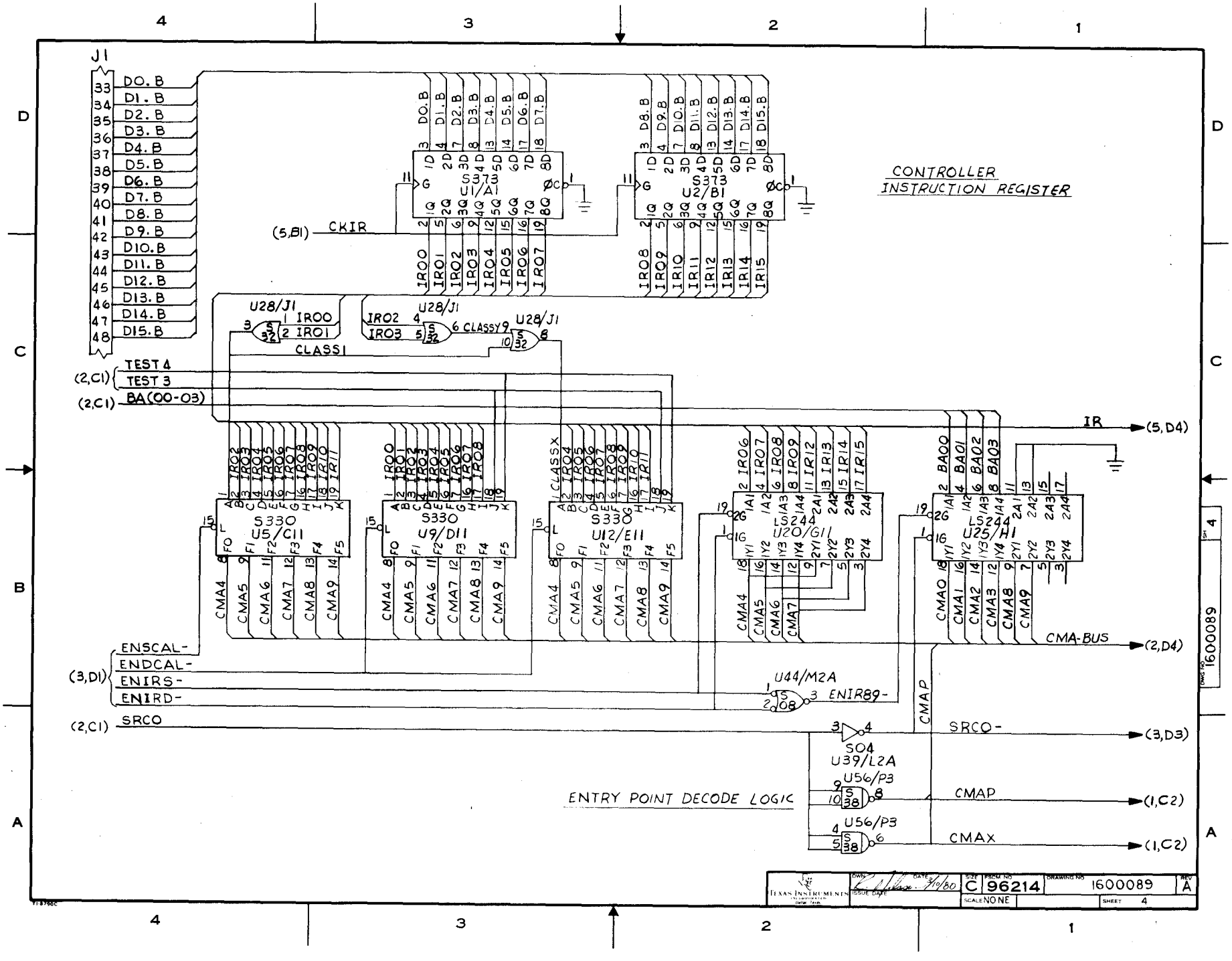
1600089 SH 1

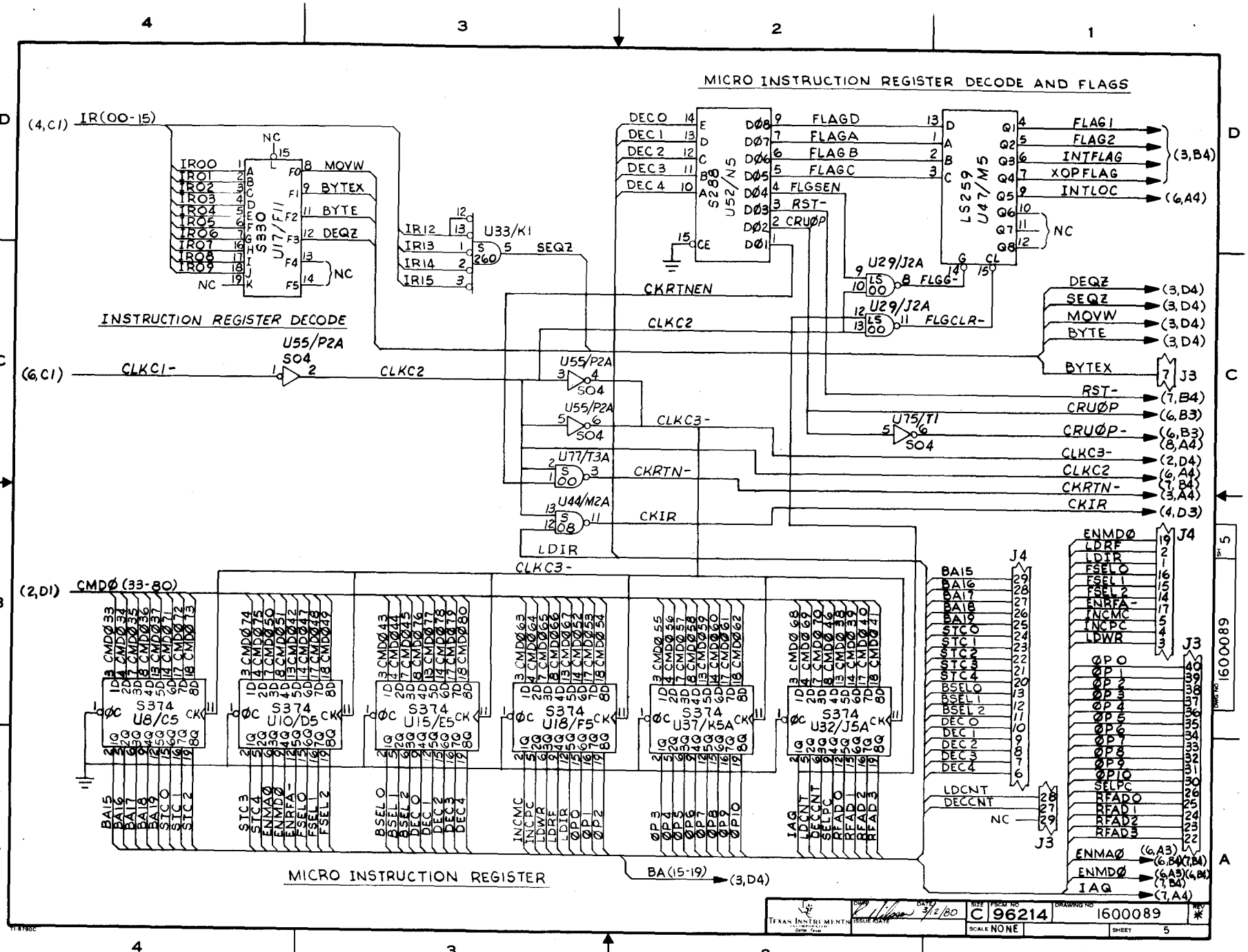
F-14

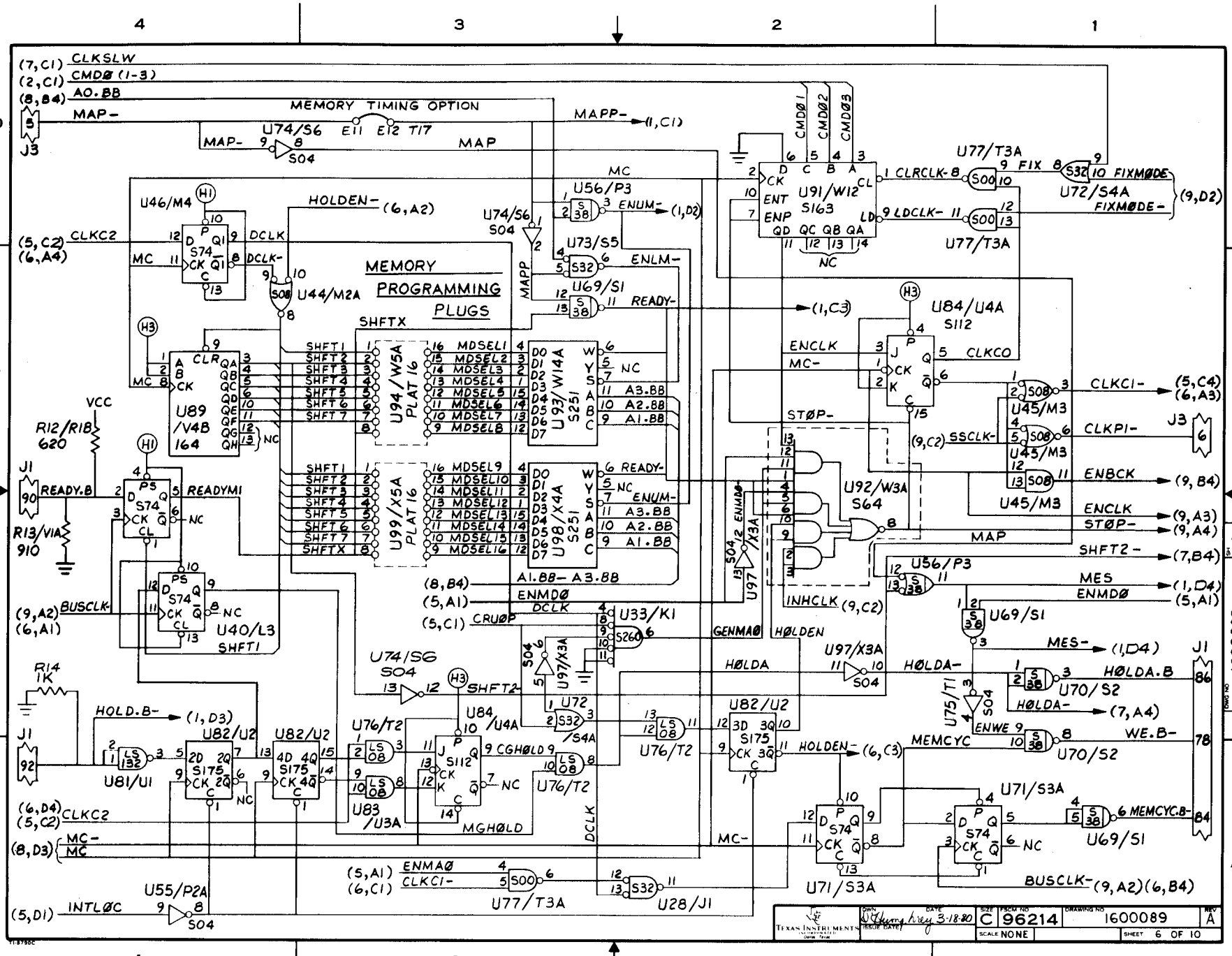
43

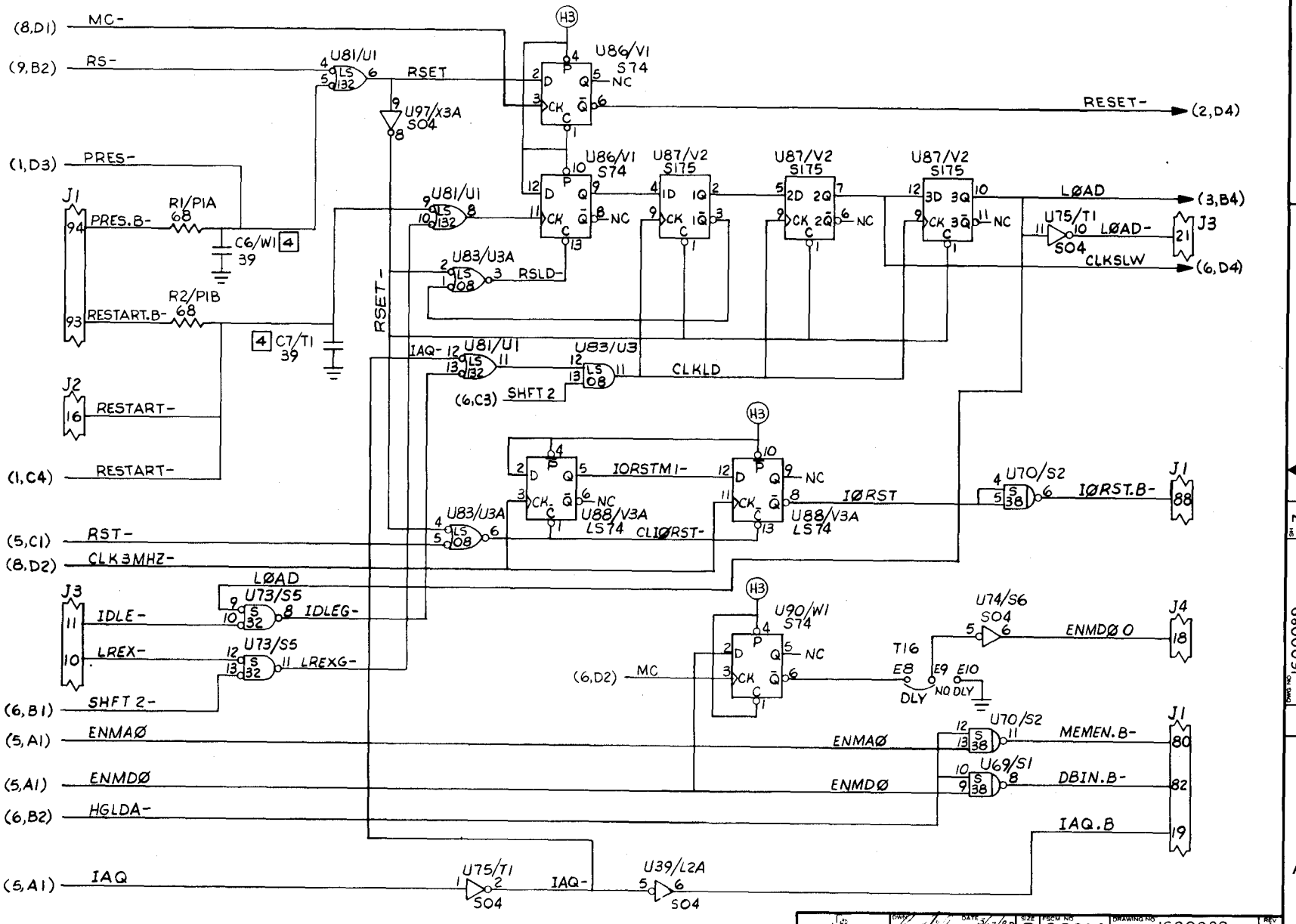

(1, D2) RESET-
(6, B1) CLKC3-
(3, D1)(4, B1) CMA-BUS
FROM ADAPTER PLUG (1, B3)
RØMP18
RØMP19
RØMP20
RØMP21
TO ALL CONTROL MEMORY PROMS
PINS 18, 19, 20, 21
U13/E2 PROM(#5) S478
U4/B4 PROM(#10) S478
U14/E4 PROM(#6) S478
U26/H2 PROM(#1) S478
U6/C2 PROM(#7) S478
U27/H4 PROM(#2) S478
U7/C4 PROM(#8) S478
U21/G2 PROM(#3) S478
U3/B2 PROM(#9) S478
U22/G4 PROM(#4) S478
CONTROL MEMORY
CMDØ → (5, B4)
MICROINSTRUCTION REGISTER
CMDØ01
CMDØ02
CMDØ03
CLOCK CONTROL (6, D4)
CMAP
U59/P6 S174
UMPB
SRCØ
SRC1
SRC2
TESTØ
TEST1
(3, C2)
(4, B2)
U53/N6 S174
TEST2
TEST3
TEST4
BAØØ
BAØ1
BAØ2
(3, D4)
U48/M6 S174
BAØ3
BAØ4
BAØ5
BAØ6
BAØ7
BAØ8
U23/FSC S174
BAØ9
BXØ
BX1
BX2
NC
CMAX
UMXB
(3, C2)
J4
U24/G5 S374
BX3
BX4
BX5
BAI0
BAI1
BAI2
BAI3
BAI4
BX(4-5)
BA(10-14)
(3, D4)


MIR/SOURCE
SRCO-
MIR/BRANCH ADDRESS 1
MIR/BRANCH ADDRESS 0
MIR/TEST
SEQZ
DEQZ
MOVW
BYTE
JUMP
CNTEQZ
LGT
AGT
EQ
COSH
OVFL
PARITY
PINT
GEQ
A15. B
FOO
COSAV
FLAG 1
FLAG 2
INTFLG
XOPFLG
LOAD
TEST 2
TEST 3
TEST4
TEO-
TE1-
TE2-
TEST O
TEST 1
TEST MULTIPLEXER
BRANCH-
BX4
BX5
BA10
BA11
UMXB
UMPB
BA00
BA01
BA12
BA13
BA14
BA15
BA02
BA03
BA04
BA05
BA16
BA17
BA18
BA19
BA06
BA07
BA08
BA09
ENBRN-
BRANCH MULTIPLEXER
CMA-BUS
CMAX
CMAP
CMAO
CMA1
CMA2
CMA3
CMA4
CMA5
CMA6
CMA7
CMA8
CMA9
ENRTN-
RETURN REGISTER
SRC1
SRC2
ENSCAL-
ENDCAL-
ENIRS-
ENIRD-
CKRTN-
S251 U41/L4
S251 U43/L6
S251 U42/L5
S139 U38/L1
LS257 U30/J3
LS257 U35/K3
LS257 U34/K2A
LS374 U31/J4
LS374 U36/K4
U29/J2A
U39/L2A
U73/S5
TEXAS INSTRUMENTS
C 96214
1600089
A
SCALE NONE
SHEET 3

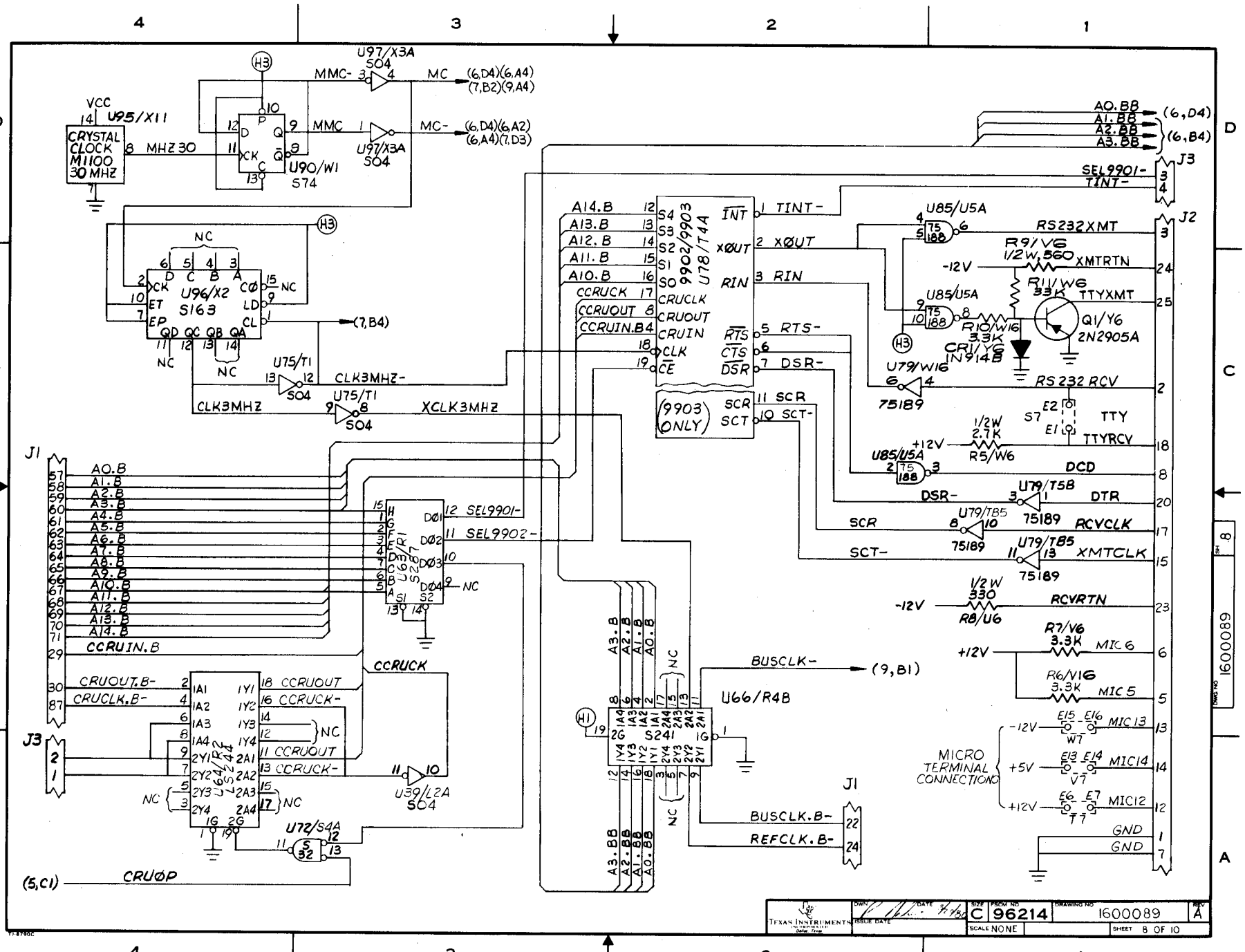

J1
DO.B
D1.B
D2.B
D3.B
D4.B
D5.B
D6.B
D7.B
D8.B
D9.B
D10.B
D11.B
D12.B
D13.B
D14.B
D15.B
CONTROLLER
INSTRUCTION REGISTER
(5,B1) CKIR
S373
U1/A1
S373
U2/B1
IR00
IR01
IR02
IR03
IR04
IR05
IR06
IR07
IR08
IR09
IR10
IR11
IR12
IR13
IR14
IR15
U28/J1
CLASS1
CLASSY9
CLASSX
TEST 4
TEST 3
(2,C1)
BA(00-03)
IR
(5,D4)
S330
U5/C11
S330
U9/D11
S330
U12/E11
LS244
U20/G11
LS244
U25/H1
CMA-BUS
(2,D4)
ENSCAL-
ENDCAL-
ENIRS-
ENIRD-
(3,D1)
U44/M2A
ENIR89-
SRCO
(2,C1)
S04
U39/L2A
SRCO-
(3,D3)
CMAP
U56/P3
(1,C2)
CMAX
U56/P3
(1,C2)
ENTRY POINT DECODE LOGIC
TEXAS INSTRUMENTS
DATE 7/10/80
SIZE C
PROJ NO 96214
DRAWING NO 1600089
REV A
SCALE NONE
SHEET 4
DWG NO 1600089
SH 4
F-17

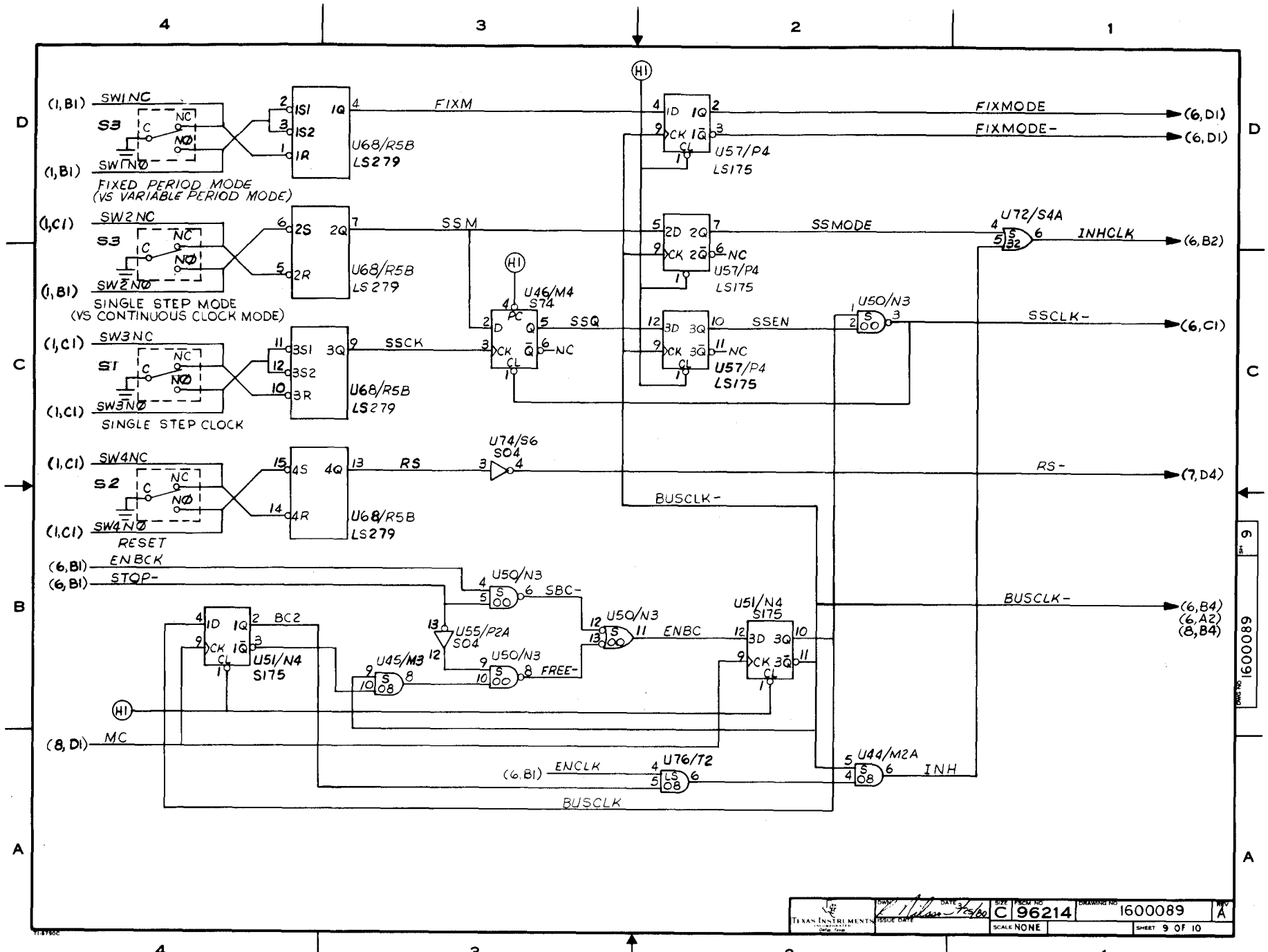

MICRO INSTRUCTION REGISTER DECODE AND FLAGS
INSTRUCTION REGISTER DECODE
MICRO INSTRUCTION REGISTER
IR(00-15)
CLKC1-
CLKC2
CLKC3-
CKRTNEN
CKRTN-
CKIR
LDIR
CMDØ(33-80)
BA(15-19)
DEQZ
SEQZ
MOVW
BYTE
BYTEX
RST-
CRUØP
CRUØP-
FLAG1
FLAG2
INTFLAG
XOPFLAG
INTLOC
FLAGD
FLAGA
FLAGB
FLAGC
FLGSEN
FLGG-
FLGCLR-
ENMDØ
LDRF
FSEL0
FSEL1
FSEL2
ENRFA-
INCMC
INCPC
LDWR
ØP0
ØP1
ØP2
ØP3
ØP4
ØP5
ØP6
ØP7
ØP8
ØP9
ØP10
SELPC
RFAD0
RFAD1
RFAD2
RFAD3
LDCNT
DECCNT
IAQ
ENMAØ
ENMDØ
U8/C5
U10/D5
U15/E5
U18/F5
U37/K5A
U32/J5A
U17/F11
U52/N5
U47/M5
U33/K1
U55/P2A
U29/J2A
U75/T1
U77/T3A
U44/M2A
J3
J4
NC
TEXAS INSTRUMENTS
C 96214
1600089
SCALE NONE
SHEET 5
3/2/80

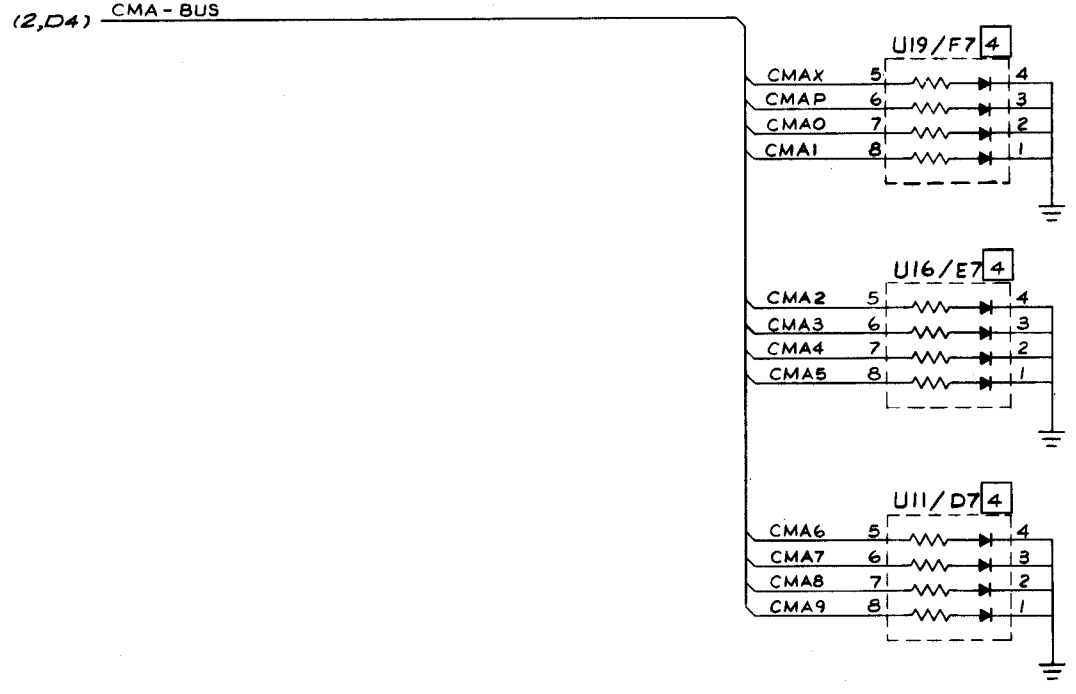

(7,C1) CLKSLW
(2,C1) CMDØ (1-3)
(8,B4) AO.BB
MAP-
J3
MEMORY TIMING OPTION
U74/S6 E11 E12 T17
MAP- MAP
S04
MAPP- (1,C1)
MC
U56/P3
HOLDEN- (6,A2)
U74/S6 S04
MAPP
U73/S5
ENUM- (1,D2)
ENLM-
U69/S1
READY- (1,C3)
CMDØ1 CMDØ2 CMDØ3
U91/W12 S163
CLRCLK-B
LDCLK-
NC
U77/T3A
FIX
FIXMØDE
U72/S4A
FIXMØDE- (9,D2)
U46/M4
DCLK
DCLK-
U44/M2A
(5,C2) CLKC2
(6,A4)
MC
MEMORY PROGRAMMING PLUGS
SHFTX
U84/U4A S112
ENCLK
MC-
CLKCO
STØP-
CLKCI- (5,C4) (6,A3)
U45/M3
CLKPI-
J3
SSCLK- (9,C2)
ENBCK (9,B4)
ENCLK (9,A3)
STØP- (9,A4)
U92/W3A S64
MAP
SHFT2- (7,B4)
U56/P3
MES
ENMDØ (5,A1)
INHCLK (9,C2)
U69/S1
MES- (1,D4)
U97/X3A
HØLDA
HØLDA-
HØLDA.B
U70/S2
HØLDA- (7,A4)
U75/T1
ENWE
WE.B-
MEMCYC
U70/S2
U71/S3A
6 MEMCYC.B-
U69/S1
BUSCLK- (9,A2)(6,B4)
U71/S3A
U82/U2 S175
HOLDEN- (6,C3)
U76/T2
MGHØLD
DCLK
U83/U3A
U81/U1
HOLD.B- (1,D3)
U84/U4A
SHFT2
U72/S4A
U33/K1
CRUØP
ENMDØ
A1.BB- A3.BB
U98/X4A S251
U93/W14A S251
U94/W5A PLAT 16
U99/X5A PLAT 16
MDSEL1 MDSEL2 MDSEL3 MDSEL4 MDSEL5 MDSEL6 MDSEL7 MDSEL8
MDSEL9 MDSEL10 MDSEL11 MDSEL12 MDSEL13 MDSEL14 MDSEL15 MDSEL16
SHFT1 SHFT2 SHFT3 SHFT4 SHFT5 SHFT6 SHFT7 SHFTX
A3.BB A2.BB A1.BB
READY-
ENUM-
U89/V4B 164
U40/L3 SHFT1
VCC
R12/R18 620
R13/V1A 910
R14 1K
READY.B
READYMI
BUSCLK-
(9,A2) (6,A1)
(5,C1)
(8,B4)
(5,A1)
(6,D4) (5,C2) CLKC2
(8,D3) MC- MC
(5,A1) ENMAØ
(6,C1) CLKCI-
U77/T3A
U28/J1
(5,D1) INTLØC
U55/P2A
S04
J1
92
86
78
84
90
6
TEXAS INSTRUMENTS
DRAWN
DATE 3-18-80
SIZE C
FSCM NO 96214
DRAWING NO 1600089
REV A
SCALE NONE
SHEET 6 OF 10



TEXAS INSTRUMENTS	DWN	DATE 3/13/80	SIZE C	FSCM NO 96214	DRAWING NO 1600089	REV A
		ISSUE DATE	SCALE NONE		SHEET 7	


TM 990/1481 CONTROLLER SCHEMATIC
F-21
U97/X3A S04
MMC-
MC
(6,D4)(6,A4) (7,B2)(9,A4)
MMC
MC-
(6,D4)(6,A2) (6,A4)(7,D3)
VCC
U95/X11
CRYSTAL CLOCK M1100 30 MHZ
MHZ 30
U90/W1 S74
U96/X2 S163
NC
(7,B4)
U75/T1 S04
CLK3MHZ-
CLK3MHZ
XCLK3MHZ
U78/T44 9902/9903
A14.B
A13.B
A12.B
A11.B
A10.B
CCRUCK
CCRUOUT
CCRUIN.B4
INT
XOUT
RIN
RTS
CTS
DSR
CLK
CE
TINT-
XOUT
RIN
RTS-
DSR-
(9903 ONLY)
SCR
SCT
SCR
SCT-
J1
J2
J3
AO.BB
A1.BB
A2.BB
A3.BB
(6,D4)
(6,B4)
SEL9901-
TINT-
U85/U5A
RS232XMT
R9/V6 1/2W, 560
XMTRTN
-12V
R11/W6 33K
TTYXMT
Q1/Y6 2N2905A
R10/W16 3.3K
CR1/Y6 IN914B
U79/W16 75189
RS232RCV
S7
E2
E1
TTY
TTYRCV
1/2W 2.7K
R5/W6
+12V
DCD
U79/T5B
DTR
DSR-
U79/TB5
75189
RCVCLK
SCR
U79/T8S
XMTCLK
SCT-
1/2W 330
R8/U6
RCVRTN
R7/V6 3.3K
MIC 6
R6/V16 3.3K
MIC 5
MICRO TERMINAL CONNECTIONS
E15 E16 W7
MIC 13
E13 E14 V7
MIC 14
E6 E7 T7
MIC 12
GND
GND
U66/R4B
S241
BUSCLK-
(9,B1)
BUSCLK.B-
REFCLK.B-
U72/S4A
U39/L2A S04
CRUOP
(5,C1)
CCRUIN.B
CRUOUT.B-
CRUCLK.B-
CCRUOUT
CCRUCK-
SEL9901-
SEL9902-
AO.B
A1.B
A2.B
A3.B
A4.B
A5.B
A6.B
A7.B
A8.B
A9.B
A10.B
A11.B
A12.B
A13.B
A14.B
TEXAS INSTRUMENTS
SIZE C
FSCM NO 96214
DRAWING NO 1600089
REV A
SCALE NONE
SHEET 8 OF 10
8
1600089


SW1NC
S3
SW1NO
(1,B1)
FIXED PERIOD MODE
(VS VARIABLE PERIOD MODE)
U68/R5B
LS279
FIXM
U57/P4
LS175
FIXMODE
FIXMODE-
(6,D1)
SW2NC
SINGLE STEP MODE
(VS CONTINUOUS CLOCK MODE)
SSM
U46/M4
S74
SSQ
SSMODE
U72/S4A
INHCLK
(6,B2)
U50/N3
SSEN
SSCLK-
(6,C1)
SW3NC
S1
SW3NO
SINGLE STEP CLOCK
SSCK
SW4NC
S2
SW4NO
RESET
U74/S6
S04
RS
RS-
(7,D4)
BUSCLK-
(6,B4)
(6,A2)
(8,B4)
ENBCK
STOP-
SBC-
U55/P2A
ENBC
U51/N4
S175
BC2
U45/M3
FREE-
H1
MC
(8,D1)
ENCLK
U76/T2
U44/M2A
INH
BUSCLK
TM 990/1481 CONTROLLER SCHEMATIC
F-22
6800091
TEXAS INSTRUMENTS
SIZE C
FSCM NO 96214
DRAWING NO 1600089
REV A
SCALE NONE
SHEET 9 OF 10



CONTROL MEMORY ADDRESS LED DISPLAY
SUGGEST DIALIGHT #547-2007 LED'S,
IF LED'S ARE DESIRED

TEXAS INSTRUMENTS INCORPORATED Dallas, Texas	DWN D. Humphrey 3 2 82	SIZE C	PSCM NO 96214	DRAWING NO 1600089	REV A
	ISSUE DATE	SCALE NONE		SHEET 10 OF 10	

APPENDIX G

990 OBJECT CODE FORMAT

G.1 GENERAL

In order to correctly load a program into memory using a loader, the program in hexadecimal machine code must be in a particular format called object format. Such a format is required by the *TIBUG* loader (paragraph 3.2.7 explains loader execution). This object format has a tag character for each 16-bit word of coding which flags the loader to perform one of several operations. These operations include:

- Load the code at a user-specified absolute address and resolve relative addresses. (Most assemblers assemble a program as if it was loaded at memory address 0000_{16}; thus, relative addresses have to be resolved.)
- Load entire program at a specific address.
- Set the program counter to the entry address after loading.
- Check for checksum errors that would indicate a data error in an object record.

G.2 STANDARD 990 OBJECT CODE

Standard 990 object code consists of a string of hexadecimal digits, each representing four bits, as shown in Figure G-1.

\0001462

FIGURE G-1. OBJECT CODE EXAMPLE

The object record consists of a number of tag characters, each followed by one or two fields as defined in Table G-1. The first character of a record is the first tag character, which tells the loader which field or pair of fields follows the tag. The next tag character follows the end of the field or pair of fields associated with the preceding tag character. When the assembler has no more data for the record, the assembler writes the tag character 7 followed by the checksum field, and the tag character F, which requires no fields. The assembler then fills the rest of the record with blanks, and begins a new record with the appropriate tag character.

Tag character 0 is followed by two fields. The first field contains the number of bytes of relocatable code, and the second field contains the program identifier assigned to the program by an IDT assembler directive. When no IDT directive is entered, the field contains blanks. The loader uses the program identifier to identify the program, and the number of bytes of relocatable code to determine the load bias for the next module or program. The PX9ASM assembler is unable to determine the value for the first field until the entire module has been assembled, so PX9ASM places a tag character 0 followed by a zero field and the program identifier at the beginning of the object code file. At the end of the file, PX9ASM places another tag character zero followed by the number of bytes of relocatable code and eight blanks.

Tag characters 1 and 2 are used with entry addresses. Tag character 1 is used when the entry address is absolute. Tag character 2 is used when the entry address is relocatable. The hexadecimal field contains the entry address. One of these tags may appear at the end of the object code file. The associated field is used by the loader to determine the entry point at which execution starts when the loading is complete.

Tag characters 3 and 4 are used for external references. Tag character 3 is used when the last appearance of the symbol in the second field is in relocatable code. Tag character 4 is used when the last appearance of the symbol is absolute code. The hexadecimal field contains the location of the last appearance. The symbol in the second field is the external reference. Both fields are used by the linking loader to provide the desired linking to the external reference.

For each external reference in a program, there is a tag character in the object code, with a location, or an absolute zero, and the symbol that is referenced. When the object code field contains absolute zero, no location in the program requires the address that corresponds to the reference (an IDT character string, for example). Otherwise, the address corresponding to the reference will be placed in the location specified in the object code by the linking loader. The location specified in the object code similarly contains absolute zero or another location. When it contains absolute zero, no further linking is required. When it contains a location, the address corresponding to the reference will be placed in that address by the linking loader. The location of each appearance of a reference in a program contains either an absolute zero or another location into which the linking loader will place the referenced address.

TABLE G-1. OBJECT OUTPUT TAGS SUPPLIED BY ASSEMBLERS

TAG CHARACTER	HEXADECIMAL FIELD (FOUR CHARACTERS)	SECOND FIELD	MEANING
0	Length of all relocatable code	8-character program identifier	Program start
1	Entry address	None	Absolute entry address
2	Entry address	None	Relocatable entry address
3	Location of last appearance of symbol	6-character symbol	External reference last used in relocatable code
4	Location of last appearance of symbol	6-character symbol	External reference last used in absolute code
5	Location	6-character symbol	Relocatable external definition
6	Location	6-character symbol	Absolute external definition
7	Checksum for current record	None	Checksum
8	Ignore checksum	None	Do not checksum for error
9	Load address	None	Absolute load address
A	Load address	None	Relocatable load address
B	Data	None	Absolute data
C	Data	None	Relocatable data
D	Load bias value*	None	Load point specifier
F	None	None	End-of-record
G	Location	6-character symbol	Relocatable symbol definition
H	Location	6-character symbol	Absolute symbol definition

*Not supplied by assembler

Tag characters 5 and 6 are used for external definitions. Tag character 5 is used when the location is relocatable. Tag character 6 is used when the location is absolute. Both fields are used by the linking loader to provide the desired linking to the external definition. The second field contains the symbol of the external definition.

Tag character 7 precedes the checksum, which is an error detection word. The checksum is formed as the record is being written. It is the 2's complement of the sum of the 8-bit ASCII values of each character in the object record from the first tag of the record through (and including) the checksum tag 7. If the tag character 7 is replaced by an 8, the checksum will be ignored. The 8 tag can be used when object code is changed in editing and it desired to ignore checksum.

Tag characters 9 and A are used with load addresses for data that follows. Tag character 9 is used when the load address is absolute. Tag character A is used when the load address is relocatable. The hexadecimal field contains the address at which the following data word is to be loaded. A load address is required for a data word that is to be placed in memory at some address other than the next address. The load address is used by the loader.

Tag characters B and C are used with data words. Tag character B is used when the data is absolute; an instruction word or a word that contains text characters or absolute constants, for example. Tag character C is used for a word that contains a relocatable address. The hexadecimal field contains the data word. The loader places the word in the memory location specified in the preceding load address field, or in the memory location that follows the preceding data word.

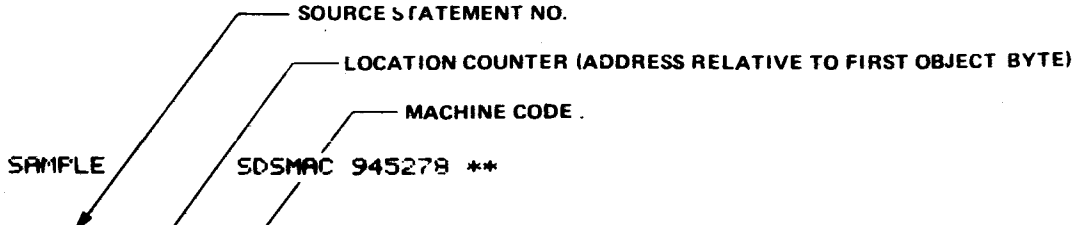
To have object code loaded at a specific memory address, precede the object program with the D tag followed by the desired memory address (e.g., DFD00).

Tag character F indicates the end of record. It may be followed by blanks.

Tag characters G and H are used when the symbol table option is specified with other 990 assemblers. Tag character G is used when the location or value of the symbol is relocatable, and tag character H is used when the location or value of the symbol is absolute. The first field contains the location or value of the symbol, and the second field contains the symbol to which the location is assigned.

The last record of an object code file has a colon (:) in the first character position of the record, followed by blanks. This record is referred to as an end-of-module separator record.

Figure G-2 is an example of an assembler source listing and corresponding object code. A comparison of the object tag characters and fields with the machine code in the source listing will show how object code is constructed for use by the loader.




```
SAMPLE        SDSMAC 945278 **                                  PAGE 0001

0001                          IDT  'SAMPLE'
 0002 0000 0006'              DATA WSPACE
 0003 0002 008A'              DATA START
0004 0004 0000                DATA 0
0005 0006           WSPACE    BSS  32
0006 0026           TABLE     BSS  100
0007 008A           START
0008 008A 04CC                CLR  12
0009 008C 04C0                CLR  0
0010 008E 0202                LI   2,TABLE
     0090 0026'
0011 0092 C800                MOV  0,@TABLE+2
     0094 0028'
0012 0096 1001                JMP  $+4
0013 0098           LOOP
0014 0098 0204                LI   4,>1234
     009A 1234
0015 009C 0244                ANDI 4,>FEED
     009E FEED
0016 00A0 DC84                MOVB 4,*2+
0017 00A2 0205                LI   5,>5555
     00A4 5555
0018 00A6 C805                MOV  5,@TABLE
     00A8 0026'
0019                          END
NO ERRORS
```

```
000AASAMPLE  A0000C0006C008AB0000A008AB04CCB04C0B0202C0026BC8007F200F          000
C0028B1001B0204B1234B0244BFEEDBDC84B0205B5555BC805C00267F3C1F                  000
:       SAMPLE     00/00/00   08:14:23                SDSMAC 945278 **
```

FIGURE G-2. SOURCE CODE AND CORRESPONDING OBJECT CODE

APPENDIX H
SPECIAL INSTRUCTIONS IMPLEMENTED BY THE TM 990/1481

H.1 GENERAL

These 26 special instructions that are implemented by the TM 990/1481 are described in this appendix. Twenty-two floating point instructions, signed multiply and divide, and load status and workspace pointer register instructions are the instructions that are covered.

H.2 INSTRUCTION DESCRIPTIONS

The instruction descriptions include the following information:

- The instruction's opcode.
- The instruction's assembled format.
- The syntax definition.
- An example.
- A definition of the instruction's operation.
- The status bits affected.
- The execution results.
- The application notes.

H.2.1 OPCODE. The opcode is the four-digit hexadecimal number which defines the instruction to be executed.

H.2.2 ADDRESSING MODE. The addressing mode lists the format (I-XXI) in which the instruction will be assembled.

H.2.3 INSTRUCTION FORMAT. The instruction format gives a block diagram of the machine language format of the instruction after it is assembled. An 'X' in the instruction format indicates an unused bit. The assembler generates zeros for bits shown as 'X'.

H.2.4 SYNTAX DEFINITION. The syntax definition for each instruction is shown, using the conventions described in Section II. The generic names used in these definitions are:

- ga_s — General address of source operand.
- ga_d — General address of destination operand.
- wa — Workspace register address.
- iop — Immediate operand.
- wa_d — Destination workspace register address.
- disp — Displacement of CRU lines from the CRU base register or signed word displacement from the current location counter.
- exp — Expression that represents an instruction location.

- cnt — Bit or byte count of another operand (specific type of nibble).
- m — Memory map file.
- scnt — Shift count (nibble value).
- op — Number (zero through 15) of extended operation (nibble value).
- ckpt — Checkpoint register (specific type of wa).
- pos — Bit position (nibble value).
- wid — Bit width (nibble value).
- cond — Matching condition to search for (nibble value).

Source statements that contain machine instructions use the label field, the operation field, the operand field, and the comment field. Use of the label field is optional for machine instructions. When the label field is used, the label is assigned the address of the machine instruction. The assembler advances the location counter to a word boundary (even address) before assembling a machine instruction. The operation (opcode) field contains the mnemonic operation code of the instruction. The use of the comment field is optional. When the comment field is used, it may contain any ASCII character, including blank, and has no effect on the assembly process other than to be printed in the listing.

H.2.5 **INSTRUCTION EXAMPLE.** An executable example is given for each instruction. Across from the example is a brief description of the operation taking place in the example.

H.2.6 **OPERATION DEFINITION.** The operation definition describes the function of each of the operands in the operand field.

H.2.7 **STATUS BITS AFFECTED.** The status bits affected by the execution of the instruction are listed.

H.2.8 **EXECUTION RESULTS.** The execution results uses a relational expression to describe the execution results. The following conventions are used in the expression:

- ( ) — indicates "the contents of".
- → — indicates "replaces".
- | | — indicates the absolute value.

H.2.9 **APPLICATION NOTES.** The application notes expand on the operation definition to give the user a more complete explanation of the use of the instruction from an application point of view.

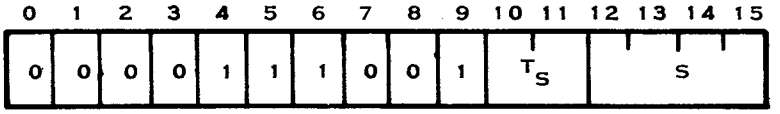
H.3 INSTRUCTIONS

H.3.1 ADD DOUBLE PRECISION REAL – AD

Opcode: 0E40

Addressing mode: Format VI

Format:



Syntax definition:

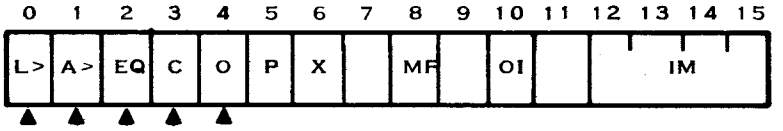
[<label>]b. . . ADb. . . <ga_s>b. . . [<comment>]

Example:

LABEL AD R6	Add the contents of workspace registers six through nine to the FPA (R0-R3)

Definition: The contents of the source address are added to the FPA (R0-R3).

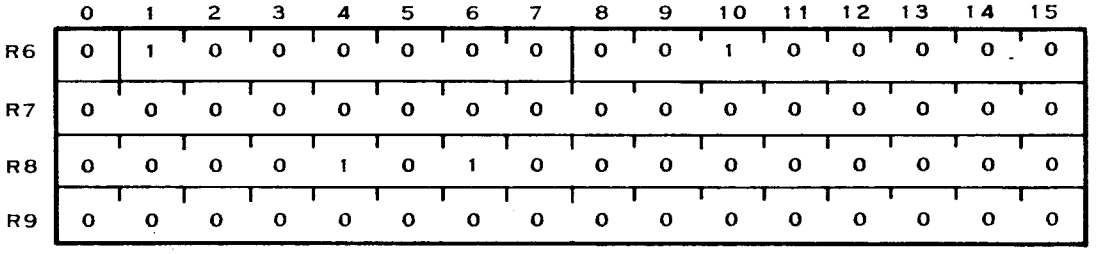
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, overflow.



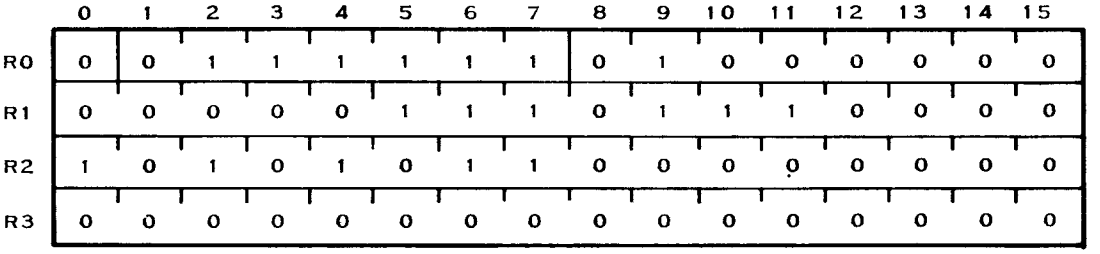
Execution results: (ga_s) + FPA → FPA

Application notes: The results of the AD instruction are compared to zero and status register bits zero, one, and two reflect the comparison. If status regiter bits three and four are set to zero and one, respectively, underflow has occurred. If both are set to one, overflow has occurred. If T_s is equal to three, the indicated register is incremented by eight.

An example of the add double precision real instruction is: If workspace registers six, seven, eight, and nine contain, after normalization, the value .2000000A_{16}, as shown figuratively below,



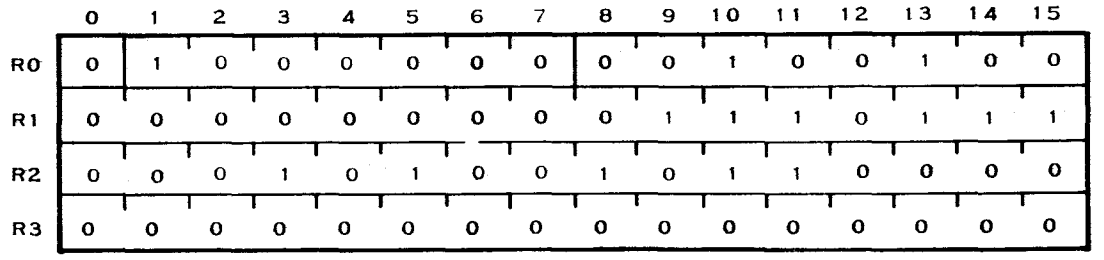
and the double precision FPA (R0-R3) contains, after normalization, the value .0400770AB$_{16}$, as shown figuratively below,



then the instruction

LABEL AD R6

will add the contents R6-R9 to the FPA and place the result, $.24007714B_{16}$, in the FPA, figuratively shown below.



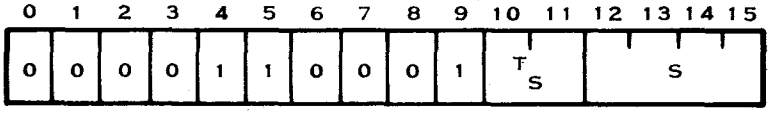
The logical greater than and arithmetic greater than bits of the status register are set; and the equal, carry, and overflow bits of the status register are reset.

H.3.2 ADD REAL – AR

Opcode: 0C40

Addressing mode: Format VI

Format:



Syntax definition:

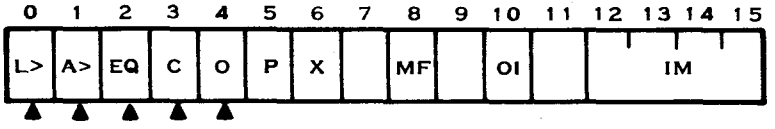
[<label>]b. . . ARb. . . <ga_s>b. . . [<comment>]

Example:

LABEL AR R5	Add workspace register five and six to the FPA (R0,R1)

Definition: Add the real number specified by the source address to the FPA and store the result in the FPA (R0,R1).

Status bits affected: Logical greater than, arithmetic greater than, equal, carry, overflow.



Execution results: FPA + (ga_s) → FPA

Application notes: If T_s is equal to three, the indicated register is incremented by four. The results of the AR instruction are compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If both are set to ones, overflow has occurred.

An example of an add real instruction is: If workspace register five and workspace register six, after normalization, contain the value $.3_{16}$, as shown figuratively below,

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R5	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0
R6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

and the FPA (R0,R1) contains the value $.00A_{16}$, after normalization, as shown figuratively below,

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R0	0	0	1	1	1	1	1	0	1	0	1	0	0	0	0	0
R1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

then the instruction

LABEL AR R5

will add the contents of R5 and R6 to the FPA and place the result, $.30A_{16}$, in the FPA, shown figuratively below.

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

The logical greater than and arithmetic greater than bits of the status register are set; and the equal, carry, and overflow status bits of the status register are reset.

Refer to Section II for a detailed description of normalization and single precision floating point instructions.

H.3.3 CONVERT DOUBLE PRECISION REAL TO EXTENDED INTEGER – CDE

Opcode: 0C05

Addressing mode: Format VII

Format:

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

Syntax definition:

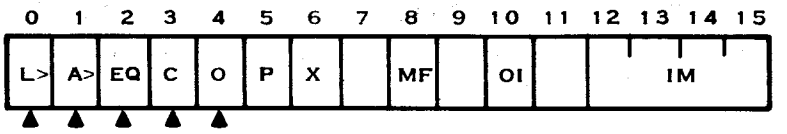
[<label>]b. . . CDEb. . . [<comment>]

Example:

LABEL CDE	Convert the double precision number in the FPA to an extended integer and place it in the FPA.

Definition: Convert the double precision number in the FPA (R0, R1, R2, R3) to an extended integer in the FPA (R0,R1).

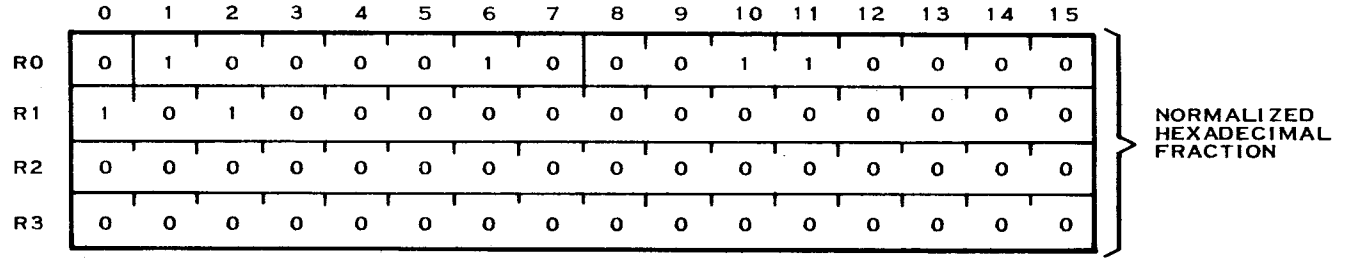
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA→FPA

Application notes: The result of the CDE instruction is compared to zero and status register bits zero, one, and two reflect the comparison. Status bit three is set to one. If status register bit four is set to one, overflow has occurred. Fractions are converted to zero without underflow.

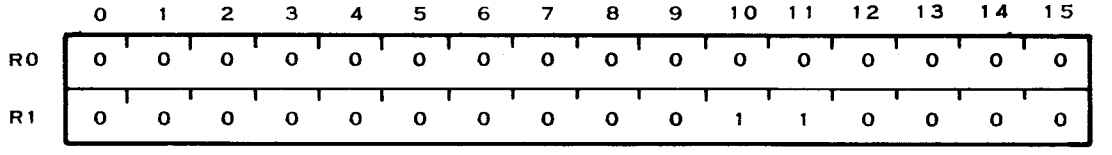
An example of the convert double precision real to extended integer instruction is: If the double precision real number in the FPA (R0-R3) is the normalized repesentation of $30.A_{16}$ as shown figuratively below:



then the instruction

LABEL CDE

will convert the double precision real number in the FPA to an extended integer, and place the result, 30_{16}, in the FPA (R0-R1), as shown figuratively below:



The logical greater than, arithmetic greater than, and carry bits of the status register are set; the equal and overflow bits are reset.

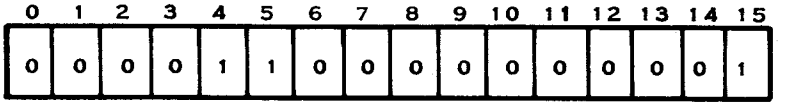
Refer to Section II for information concerning normalization and double precision real numbers.

H.3.4 CONVERT DOUBLE PRECISION REAL TO INTEGER – CDI

Opcode: 0C01

Addressing mode: VII

Format:



Syntax definition:

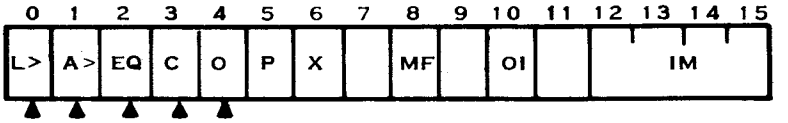
[<label>]b. . . CDIb. . . [<comment>]

Example:

LABEL CDI	Convert the double precision number in the FPA to an integer and place it in the FPA.

Definition: The double precision number in the FPA (R0, R1, R2, R3) is converted to an integer and the result is placed in the FPA (R0).

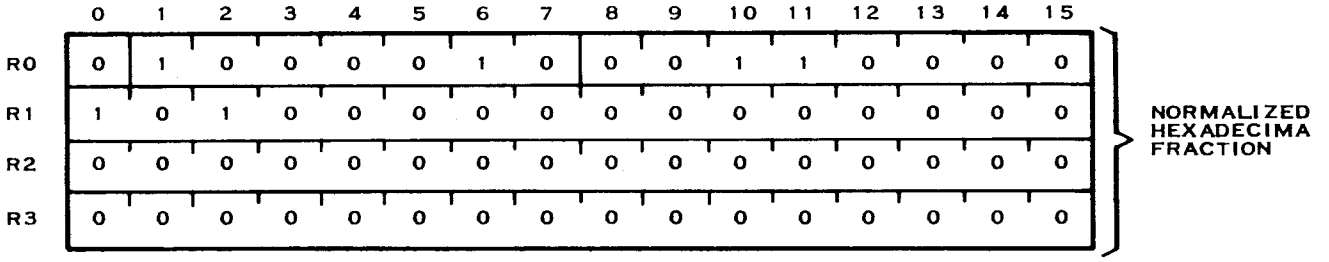
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA→FPA

Application notes: The results of the CDI instruction are compared to zero and status register bits zero, one, and two reflect the results. Bit three is set to one. If overflow occurs, bit four is set to one. Fractions are converted to zero without underflow.

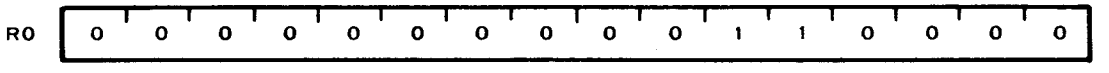
An example of the convert double precision real to integer instruction is: If the double precision real number in the FPA (R0-R3) is the normalized representation of $30.A_{16}$, as shown figuratively below:



then the instruction

LABEL CDI

will convert the double precision real number in the FPA to an integer, and place the result, 30_{16}, in the FPA (R0), as shown figuratively below:



The logical greater than, arithmetic greater than, and carry bits of the status register are set; and the equal and overflow bits are reset.

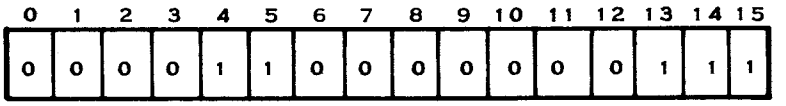
Refer to Section II for information concerning normalization and double precision real numbers.

H.3.5 CONVERT EXTENDED INTEGER TO DOUBLE PRECISION REAL – CED

Opcode: 0C07

Addressing mode: Format VII

Format:



Syntax definition:

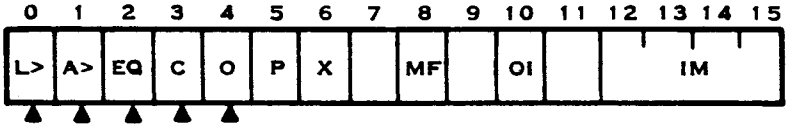
[<label>]♭. . . CED♭. . . [<comment>]

Example:

LABEL CED	Convert the extended integer in the FPA to a double precision real number and place it in the FPA.

Definition: The extended integer in the FPA (R0,R1) is converted to a double precision number and placed in the FPA (R0,R1,R2,R3).

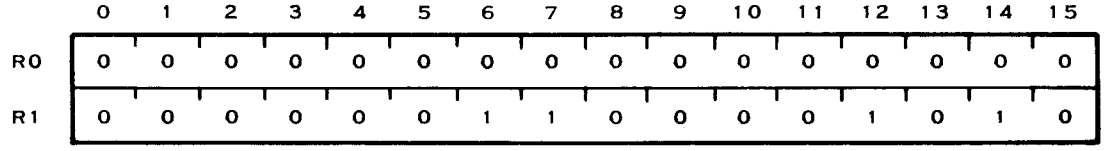
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA→FPA

Application notes: The result of the operation is compared to zero and status register bits zero, one, and two reflect the comparison. Status register bits three and four are set to zero.

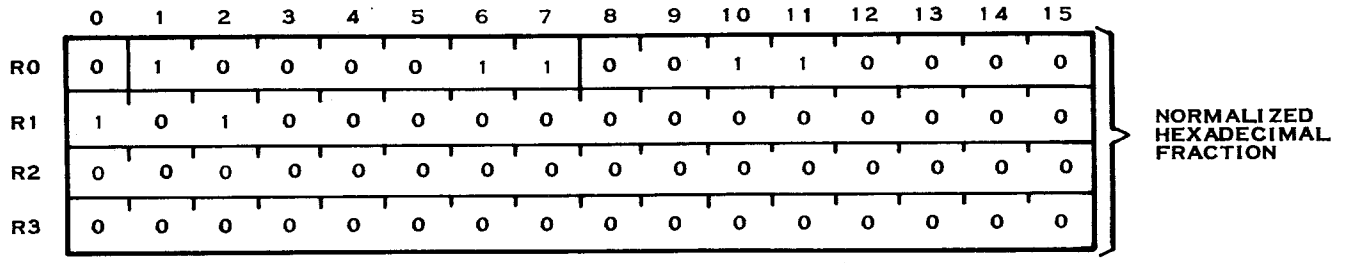
An example of the convert extended integer to double precision real instruction is: If the value in the FPA (R0-R1) is $030A_{16}$, as shown figuratively below:



then the instruction

LABEL CED

will convert the extended integer in the FPA to a normalized double precision real number and place it in the double precision FPA, as shown figuratively below:



The logical greater than and arithmetic greater than bits of the status register are set; the equal, carry, and overflow bits are reset.

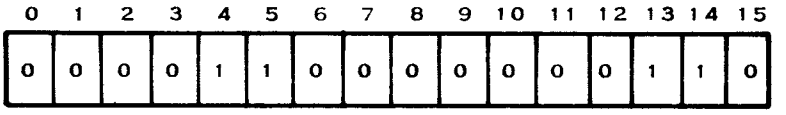
Refer to Section II for information concerning normalization and double precision real numbers.

H.3.6 CONVERT EXTENDED INTEGER TO REAL – CER

Opcode: 0C06

Addressing mode: Format VII

Format:



Syntax definition:

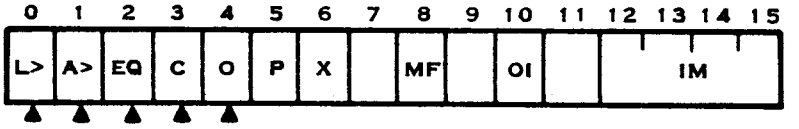
[<label>]␣. . . CER␣. . . [<comment>]

Example:

LABEL CER	Convert the extended integer in the FPA to a real number and place it in the FPA.

Definition: The extended integer in the FPA is converted to a real number and placed in the FPA (R0,R1).

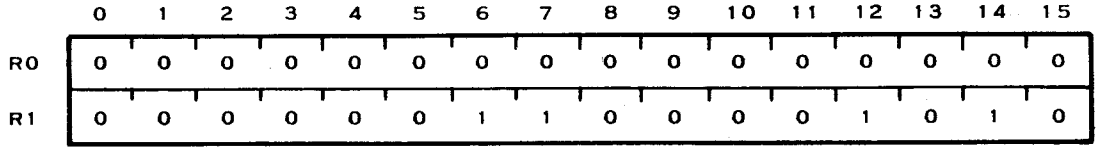
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA→FPA

Application notes: The result of the CER instruction is compared to zero and status register bits zero, one, and two reflect the comparison. Status register bits three and four are reset to zero.

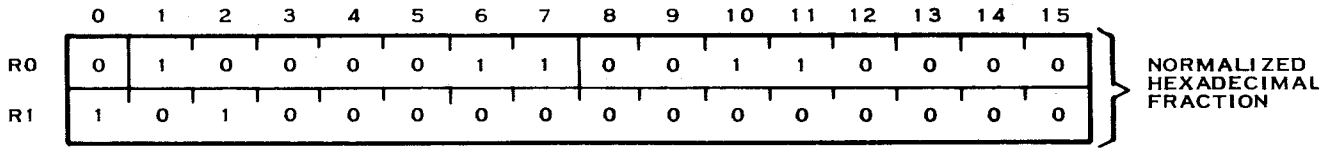
An example of the convert extended integer to real instruction is: If the value in the FPA (R0,R1) is $030A_{16}$, as shown figuratively below:



then the instruction

LABEL CER

will convert the extended integer in the FPA to a normalized single precision real number and place it in the single precision FPA, as shown figuratively below:



The logical greater than and arithmetic greater than bits of the status register are set; the equal, carry, and overflow bits are reset.

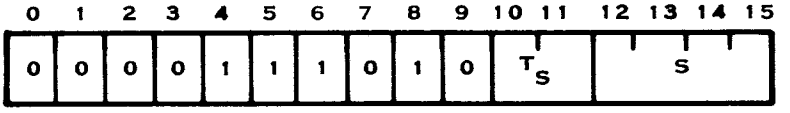
Refer to Section II for information concerning normalization and single precision real numbers.

H.3.7 CONVERT INTEGER TO DOUBLE PRECISION REAL – CID

Opcode: 0E80

Addressing mode: Format VI

Format:



Syntax definition:

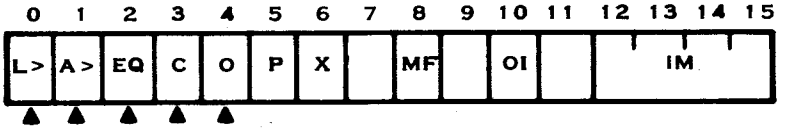
[<label>]␣. . . CID␣. . . <ga_s>␣. . .[<comment>]

Example:

LABEL CID @WORD	Convert the integer at location WORD to a double precision real number and place it in the FPA.

Definition: The integer at the source address (1 word) is converted to double precision and is stored in the floating point accumulator (R0-R3).

Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



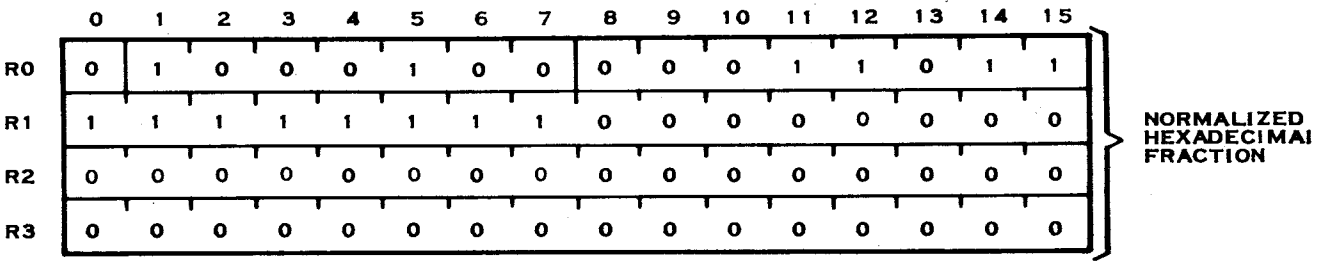
Execution results: (ga_s)→FPA

Application notes: If T_s is equal to three, the indicated register is incremented by two. The results of the CID instruction are compared to zero and status register bits zero, one, and two reflect the results. Status register bits three and four are reset.

An example of the convert integer to double precision real instruction is: If the value in WORD is $1BFF_{16}$, then the instruction

LABEL CID @WORD

will convert the integer in WORD to a normalized double precision real number and place the value in the double precision FPA, as shown figuratively below:



The logical greater than and arithmetic greater than bits of the status register are set; the equal, carry, and overflow bits are reset.

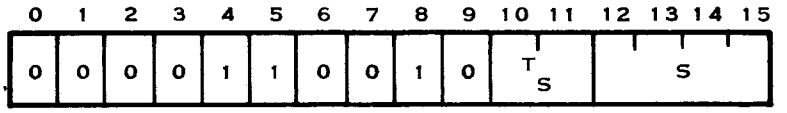
Refer to Section II for information on normalization and double precision real numbers.

H.3.8 CONVERT INTEGER TO REAL – CIR

Opcode: 0C80

Addressing mode: Format VI

Format:



Syntax definition:

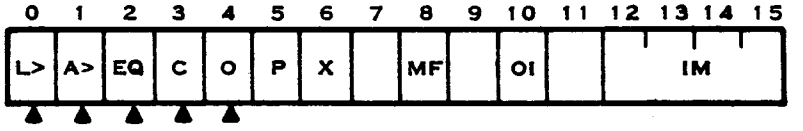
[<label>]♭. . . CIR♭. . . <ga_s>♭. . .[<comment>]

Example:

LABEL CIR @WORD	Convert the integer at location WORD to a real number and store it in the FPA.

Definition: The integer specified by the source address is converted to a real number and stored in the FPA (R0-R1).

Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



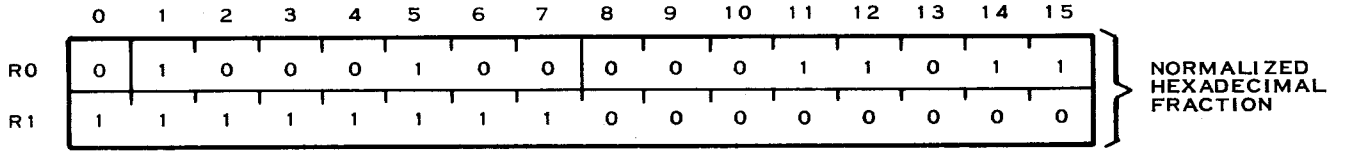
Execution results: $(ga_s) \rightarrow FPA$

Application notes: The results of the CIR instruction are compared to zero and status register bits zero, one, and two reflect the comparison. Status register bits three and four are set to zero. If T_s is equal to three, the indicated register is incremented by two.

An example of the convert integer to real instruction is: If location WORD contains the value $1BFF_{16}$, then the instruction

```
LABEL CIR @WORD
```

will convert the integer at location WORD to a normalized single precision real number and place the value in the single precision FPA, as shown figuratively below:



The logical greater than and arithmetic greater than bits of the status register are set; the equal, carry, and overflow bits are reset.

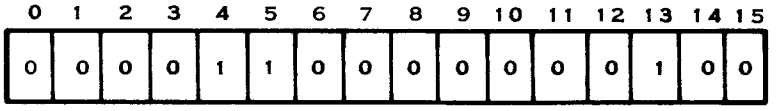
Refer to Section II for information concerning normalization and single precision real numbers.

H.3.9 CONVERT REAL TO EXTENDED INTEGER – CRE

Opcode: 0C04

Addressing mode: Format VII

Format:



Syntax definition:

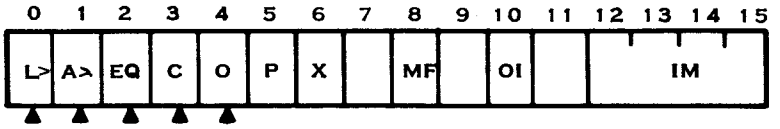
[<label>]b. . . CREb. . . [<comment>]

Example:

LABEL CRE	Convert the real number in the FPA to an extended integer and place it in the FPA.

Definition: The real number in the FPA (R0,R1) is converted to an extended integer in the FPA (R0,R1).

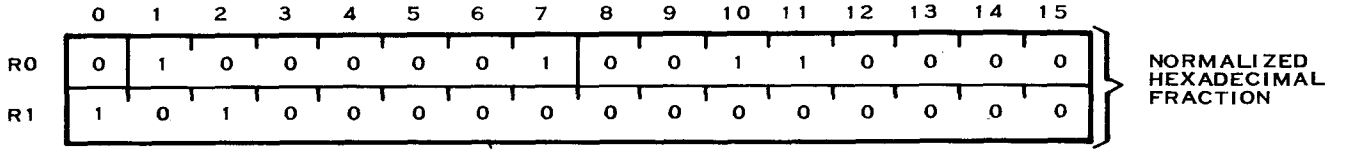
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA→FPA

Application notes: The result of the CRE instruction is compared to zero and status register bits zero, one, and two reflect the comparison. Status bit three is set to one. Status bit four is set to one if overflow occurs; otherwise it is set to zero. Fractions convert to zero and underflow does not occur.

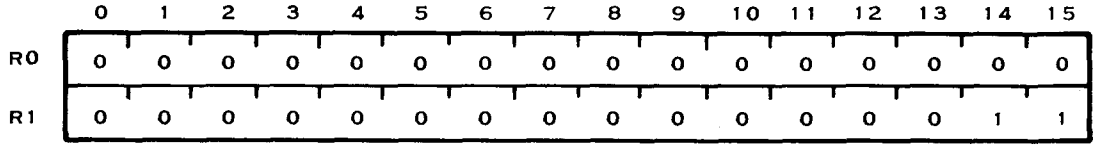
An example of the convert real to extended integer instruction is: If the real number in the FPA (R0-R1) is the normalized representation of $3.0A_{16}$, as shown figuratively below:



then the instruction

LABEL CRE

will convert the real number in the FPA to an extended integer, and place the result, 3_{16}, in the FPA (R0-R1), as shown figuratively below:



The logical greater than, arithmetic greater than, and carry bits of the status register are set; the equal and overflow bits are reset.

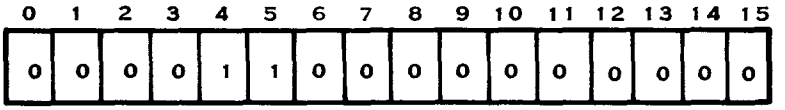
Refer to Section II for information concerning normalization and single precision real numbers.

H.3.10 CONVERT REAL TO INTEGER – CRI

Opcode: 0C00

Addressing mode: Format VII

Format:



Syntax definition:

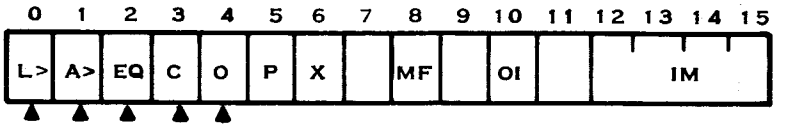
[<label>]b. . . CRIb. . .[<comment>]

Example:

LABEL CRI	Convert the real number in the FPA to an integer and place it in the FPA.

Definition: The real number in the FPA (R0,R1) is converted to an integer in the FPA (R0). Fractions convert to zero.

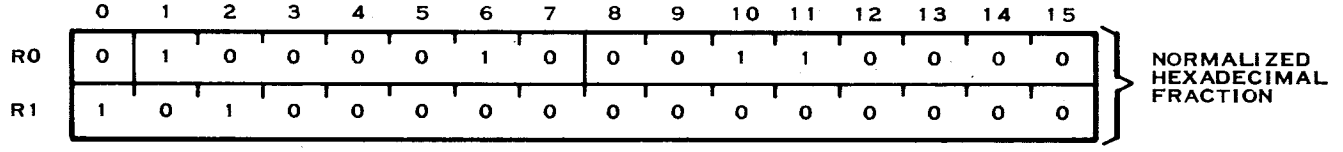
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA→FPA

Application notes: The result of the CRI instruction is compared to zero and status register bits zero, one, and two reflect the comparison. Status bit three is set to one. Status bit four is set to one if overflow occurs, otherwise it is set to zero.

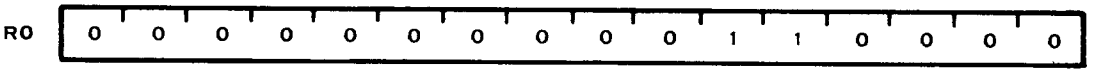
An example of the convert real to integer instruction is: If the real number in the FPA (R0-R1) is the normalized representation of $30.A_{16}$ as shown figuratively below:



then the instruction

LABEL CRI

will convert the real number in the FPA to an integer, and place the result, 30_{16}, in the FPA (R0), as shown figuratively below:



The logical greater than, arithmetic greater than, and carry bits of the status register are set; the equal and overflow bits are reset.

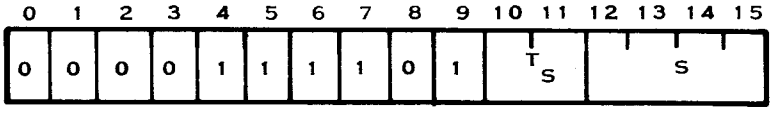
Refer to Section II for information concerning normalization and single precision real numbers.

H.3.11 DIVIDE DOUBLE PRECISION REAL – DD

Opcode: 0F40

Addressing mode: Format VI

Format:



Syntax definition:

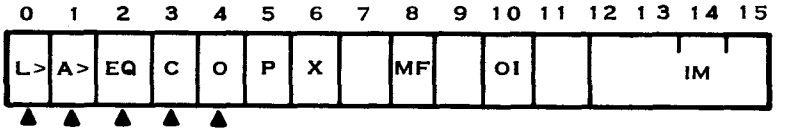
[<label>]␣. . . DD␣. . . <ga_s>␣. . . [<comment>]

Example:

LABEL DD @WORD	Divide the contents of the FPA by the contents of the word at location WORD and place the result in the FPA.

Definition: Divide the FPA by the word at the source address and place the result in the FPA (R0-R3).

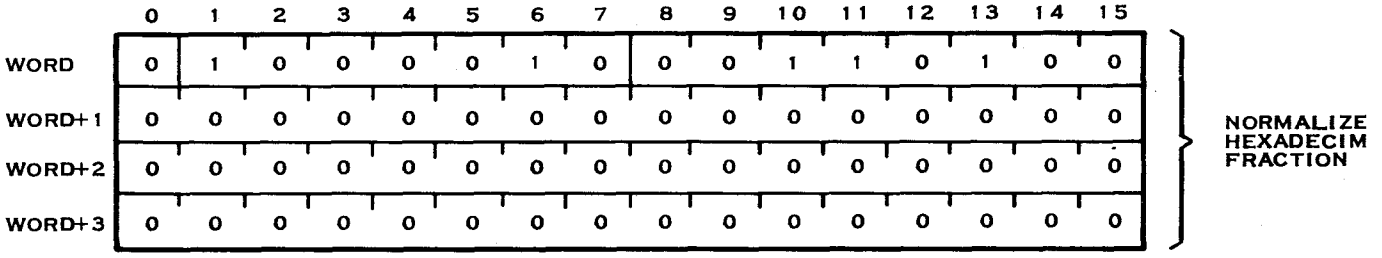
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



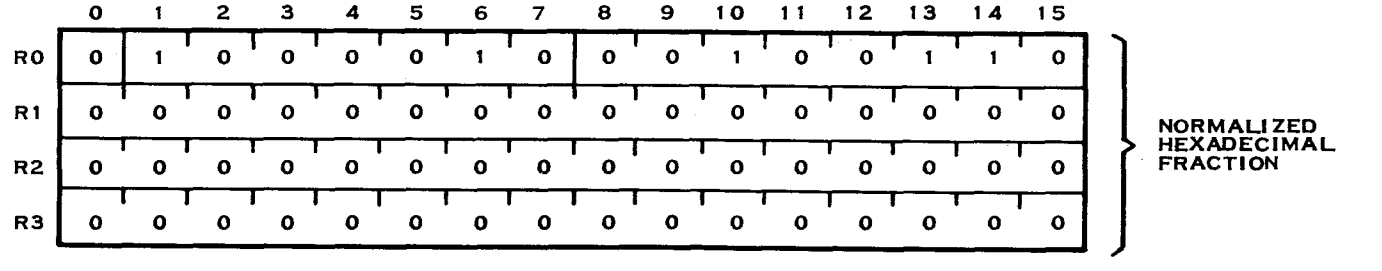
Execution results: FPA ÷ (ga_s)→FPA

Application notes: The results of the DD instruction are compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to one, overflow has occurred. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If T_s is equal to three, the indicated register is incremented by eight.

An example of the divide double precision real instruction is: If the value starting at location WORD, after normalization, is 34_{16}, shown figuratively below,



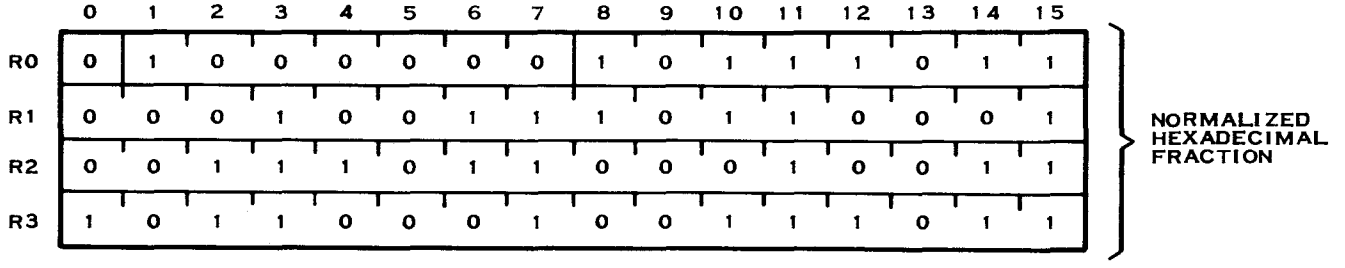
and the value in the double precision FPA (R0-R3), after normalization, is 26_{16}, shown figuratively below,



then the instruction

LABEL DD WORD

will divide the value in the FPA by the value starting at location WORD, and place the result, .BB13B13B13B13B$_{16}$, in the FPA; shown figuratively below.



The logical greater than and arithmetic greater than bits of the status register are set; and the equal, carry, and overflow bits of the status register are reset.

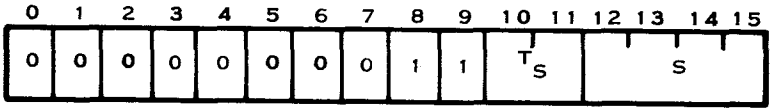
Refer to Section II for a detailed description of normalization and double precision floating point instructions.

H.3.12 DIVIDE SIGNED – DIVS

Opcode: 0180

Addressing mode: Format VI

Format:



Syntax definition:

[<label>]b. . . DIVSb. . . <ga_s>b. . . [<comment>]

Example:

LABEL DIVS R4 — Divide the two's complement of the value in workspace register zero and one by the two's complement value in workspace register four and place the result in workspace register zero and the remainder in workspace register one.

Definition: The signed, double-precision two's complement integer in workspace registers zero and one is divided by the signed two's complement integer at the source address. Algebraic two's complement integer division is performed. The quotient is deposited in workspace register zero and the remainder is deposited in workspace register one. The quotient and remainder are derived so that the following conditions are met:

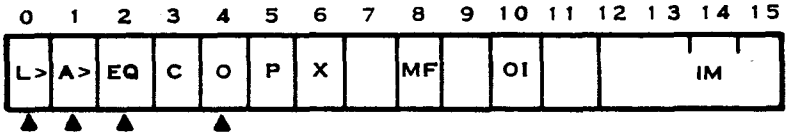
DIVISOR × QUOTIENT + REMAINDER = DIVIDEND,

where the absolute value of the remainder is less than the absolute value of the divisor.

The sign of the remainder is the same as the sign of the dividend. The sign of the quotient is derived by algebraic rules, as shown below.

DIVISOR	DIVIDEND: POS.	DIVIDEND: NEG.
POS.	POS.	NEG.
NEG.	NEG.	POS.

Status bits affected: Logical greater than, arithmetic greater than, equal, and overflow.



Execution results: R0, R1 ÷ (ga_s) = the remainder in R1
= the quotient in R0

Application notes: The DIVS instruction allows for division of signed numbers. The quotient is compared to zero and status register bits zero, one, and two reflect the comparison. Status register bit four is set if the quotient cannot be expressed in 16 bits or if the divisor equals zero. If status bit four is set, workspace registers zero and one remain unmodified.

If T_s is equal to three, the indicated register is incremented by two.

An example of a divide signed instruction is: If the double precision value contained in workspace register zero and workspace register one is $FFFFFF9F_{16}$, and the value contained in workspace register four is $FFD0_{16}$, then the instruction

LABEL DIVS R4

will divide the two's complement of the value in R0 and R1, 61_{16}, by the two's complement of the value in R4, 30_{16}, and place the quotient result, 0002_{16}, in R0 and the remainder, $FFFF_{16}$, in R1.

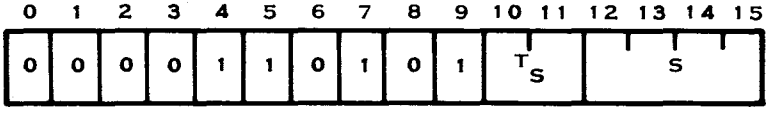
The logical greater than and the arithmetic greater than bits of the status register are set; the equal and overflow bits of the status register are reset.

H.3.13 DIVIDE REAL – DR

Opcode: 0D40

Addressing mode: Format VI

Format:



Syntax definition:

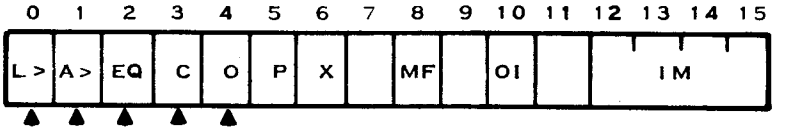
[<label>]b. . . DRb. . . <ga_s>b. . . [<comment>]

Example:

LABEL DR R7	Divide the contents of the FPA (two words) by the contents of workspace registers seven and eight and place the result in the FPA.

Definition: The real number specified by the source address is divided into the FPA (R0,R1) and the result is stored in the FPA (R0,R1).

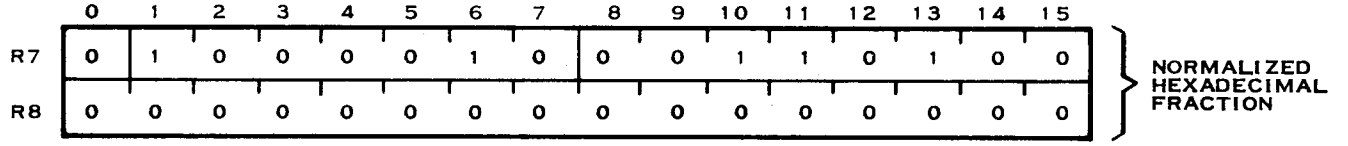
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



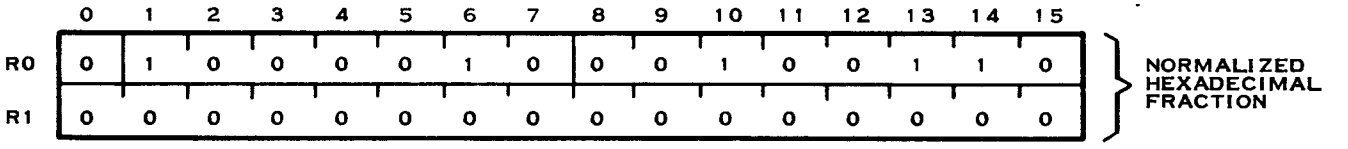
Execution results: FPA ÷ (ga_s)→FPA

Application notes: The result of the DR instruction is compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If they are set to one, overflow has occurred. If T_s is equal to three, the indicated register is incremented by four.

An example of the divide real instruction is: If the value contained in workspace registers seven and eight, after normalization, is 34_{16}, shown figuratively below,



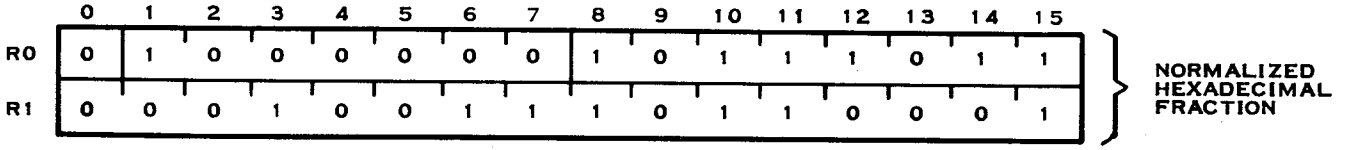
and the value contained in the single precision FPA (R0,R1), after normalization, is 26_{16}, shown figuratively below,



then the instruction

LABEL DR R7

will divide the value in the FPA by the value contained in R7 and R8, and place the result, $.BB13B1_{16}$, in the FPA, shown figuratively below.



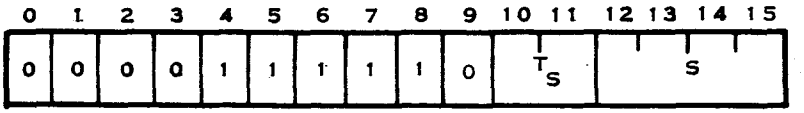
The logical greater than and the arithmetic greater than bits of the status register are set; and the equal, carry, and overflow bits of the status register are reset.

H.3.14 LOAD DOUBLE PRECISION REAL – LD

Opcode: 0F80

Addressing mode: Format VI

Format:



Syntax definition:

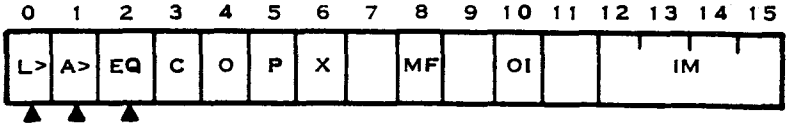
[<label>]♭. . . LD♭. . . <ga_s>♭. . . [<comment>]

Example:

LABEL LD @WORD	Load the contents of the memory location pointed to by address WORD into the FPA.

Definition: The value specified by the source address is loaded into the FPA (R0-R3).

Status bits affected: Logical greater than, arithmetic greater than, equal.



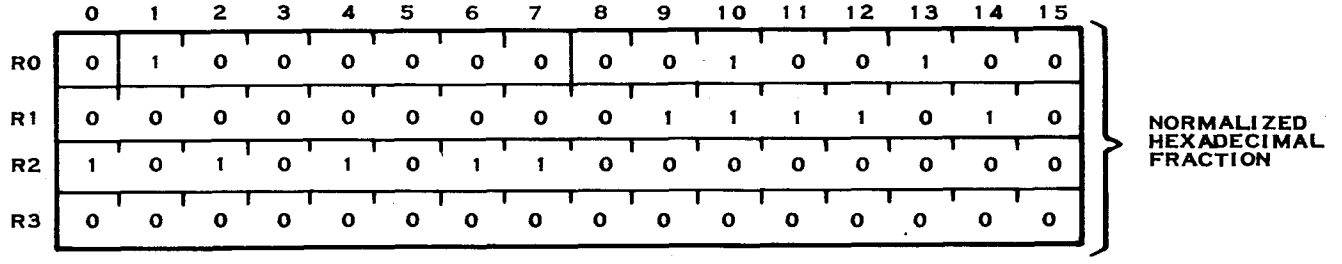
Execution results: (ga_s)→FPA

Application notes: The results of the LD instruction are compared to zero and status register bits zero, one, and two reflect the comparison. If T_s is equal to three, the indicated register is incremented by eight.

An example of the load double precision real instruction is: If the value contained in the eight bytes pointed to by WORD, after normalization, is .24007AAB$_{16}$, then the instruction

LABEL LD @WORD

will store the normalized fraction in the FPA (R0-R3), as shown figuratively below.



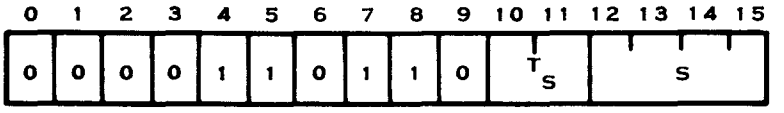
The logical greater than and arithmetic greater than bits of the status register are set; and the equal bit is reset.

H.3.15 LOAD REAL – LR

Opcode: 0D80

Addressing mode: Format VI

Format:



Syntax definition:

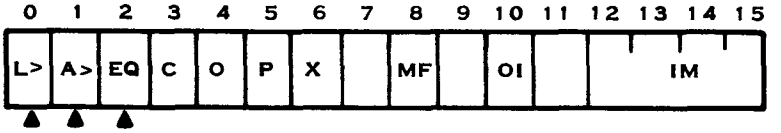
[<label>]b. . . LRb. . . <ga_s>b. . . [<comment>]

Example:

LABEL LR R6	Load workspace registers six and seven into the FPA (R0-R1)

Definition: The real number in the source address is stored in the FPA (R0-R1).

Status bits affected: Logical greater than, arithmetic greater than, and equal.



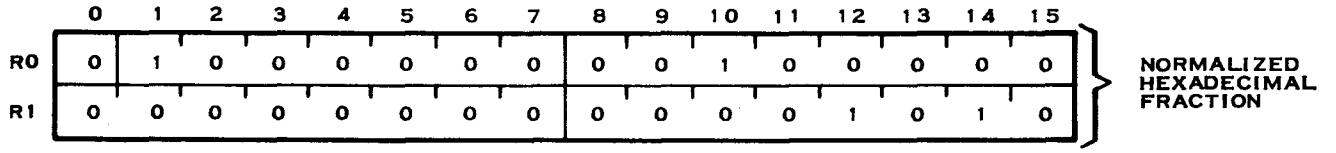
Execution results: (ga_s)→FPA

Application notes: The result of the LR instruction is compared to zero and status register bits zero, one, and two reflect the comparison. If T_s is equal to three, the indicated register is incremented by four.

An example of the load real instruction is: If the value contained in workspace register six and workspace register seven, after normalization, is .20000A_{16} then the instruction

LABEL LR R6

will store the normalized fraction in the FPA (R0-R1), shown figuratively below.



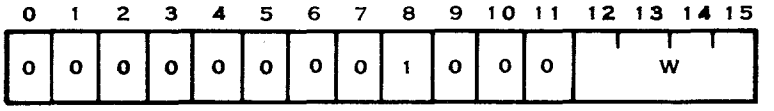
The logical greater than and arithmetic greater than bits of the status regsiter are set; and the equal bit is reset.

H.3.16 LOAD STATUS REGISTER – LST

Opcode: 0080

Addressing mode: Format XVIII

Format:



Syntax definition:

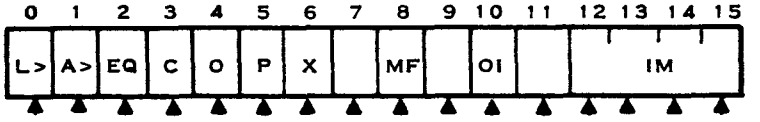
[<label>]b. . . LSTb. . . <wa>b. . . [<comment>]

Example:

LABEL LST R3	The contents of workspace register three is loaded into the status register.

Definition: The contents of $<wa_d>$ are placed in the status register.

Status bits affected: All bits.



If the privileged bit is set to one in the current status register, then only bits zero through five and ten will be loaded into the status register.

Execution results: $(wa_d) \rightarrow (ST)$

Application notes: The LST instruction loads the status register.

An example of the load status register instruction is: If workspace register three contains the value 8700_{16}, and the value of the status register is 8000_{16}, then the instruction

LABEL LST R3

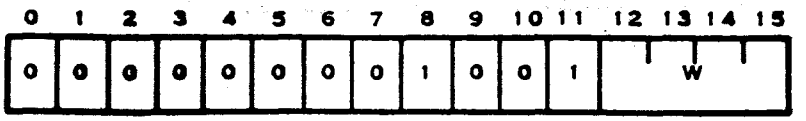
will place the value of workspace register three into the status register, in this case 8700_{16}. If the privileged bit of the status register had been set, the value of the status register, after execution of this instruction, would be 8400_{16}.

H.3.17 LOAD WORKSPACE POINTER REGISTER – LWP

Opcode: 0090

Addressing mode: Format XVIII

Format:



Syntax definition:

[<label>]b. . . LWPb. . . <wa_d>b. . . [<comment>]

Example:

LABEL LWP R5	Place the contents of workspace register five in the workspace pointer register.

Definition: The contents of the workspace register <wa_d> are placed in the workspace pointer register.

Status bits affected: None.

Execution results: $(wa_d) \rightarrow WP$

Application notes: An example of the load workspace pointer register instruction is: If workspace register five contains the value of 220_{16}, the instruction

LABEL LWP R5

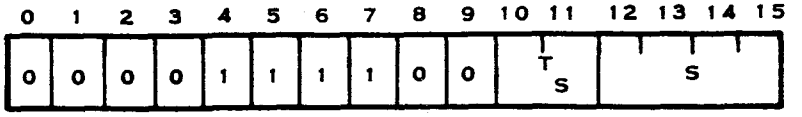
will place the contents of workspace register five, 220_{16}, into the workspace pointer register.

H.3.18 MULTIPLY DOUBLE PRECISION REAL – MD

Opcode: 0F00

Addressing mode: Format VI

Format:



Syntax Definition:

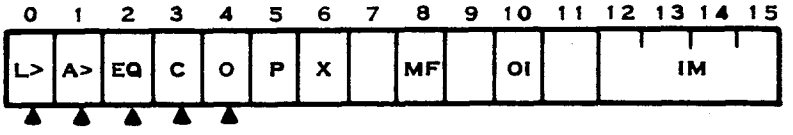
[<label>]b. . . MDb. . . <ga_s>b. . . [<comment>]

Example:

LABEL MD @WORD	Multiply the contents of the FPA by the contents of the word at location WORD and place the result in the FPA.

Definition: The contents of the FPA are multiplied by the word at the source address. The result is placed in the FPA (R0-R3).

Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



Execution results: FPA × $(ga_s) \rightarrow$FPA

Application notes: The results of the MD instruction are compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to one, overflow has occurred. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If T_s is equal to three, the indicated register is incremented by eight.

An example of multiply double precision real is: If the value starting at location WORD contains, after normalization, the value 34_{16}, as shown figuratively below,

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

NORMALIZED HEXADECIMAL FRACTION

and the double precision FPA (R0-R3), after normalization, contains the value 26_{16}, shown figuratively below,

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	1	0
R1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NORMALIZED HEXADECIMAL FRACTION

then the instruction

```
LABEL MD @WORD
```

will multiply the contents at location WORD by the contents of the FPA, and place the result, $7B8_{16}$, in the FPA, shown figuratively below.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R0	0	1	0	0	0	0	1	1	0	1	1	1	1	0	1	1
R1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NORMALIZED HEXADECIMAL FRACTION

The logical greater than and the arithmetic greater than bits of the status register are set; and the equal, carry, and overflow bits of the status register are reset.

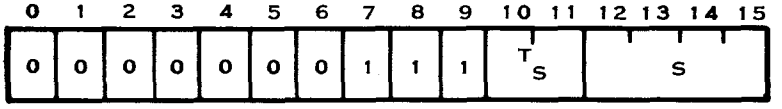
Refer to Section II for a detailed description of normalization and double precision floating point instructions.

H.3.19 MULTIPLY SIGNED – MPYS

Opcode: 01C0

Addressing mode: Format VI

Format:



Syntax definition:

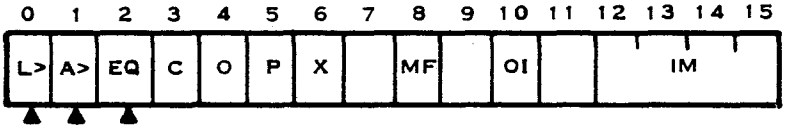
[<label>]b. . . MPYSb. . . <ga_s>b. . . [<comment>]

Example:

LABEL MPYS R3	Multiply the contents of workspace register zero by the contents of workspace register three and place the results in workspace register zero and one.

Definition: The signed, two's complement integer in workspace register zero is multiplied by the signed, two's complement integer at the source address. The product, a signed, two's complement double-length integer, is deposited in workspace registers zero (most-significant half) and one (least significant half).

Status bits affected: Logical greater than, arithmetic greater than, and equal.



Execution results: (R0) × (ga_s)→(R0 and R1)

Application notes: The MPYS instruction allows for multiplication of signed numbers.

The result of the MPYS instruction is compared to zero and status register bits zero, one, and two reflect the comparison. If T_s is equal to three, the indicated register is incremented by two.

An example of the multiply signed instruction is: If workspace register zero contains the value $FFCC_{16}$, and workspace register three contains the value $FFDA_{16}$, then the instruction

LABEL MPYS R3

will multiply the signed, two's complement value of workspace register zero by the signed, two's complement value of workspace register three, and will place the double-length result in workspace register zero and workspace register one.The product is, in this example, R0=0 and R1=$7B8_{16}$.

The sign of the result follows normal algebraic rules as follows:

positive × positive = positive

positive × negative = negative

negative × negative = positive

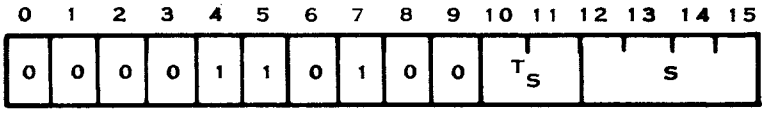
The logical greater than and the arithmetic greater than bits of the status register are set; and the equal bit of the status register is reset.

H.3.20 MULTIPLY REAL – MR

Opcode: 0D00

Addressing mode: Format VI

Format:



Syntax definition:

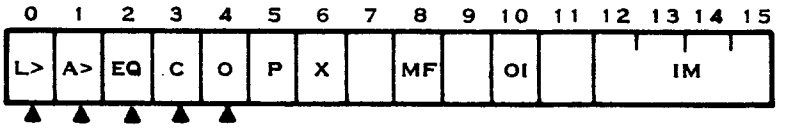
[<label>]b. . . MRb. . . <ga_s>b. . . [<comment>]

Example:

LABEL MR R5	The contents of the FPA are multiplied by the contents of workspace registers five and six. The result is placed in the FPA.

Definition: The FPA (R0,R1) is multiplied by the contents of the two words at the source address and the result is placed in the FPA (R0,R1).

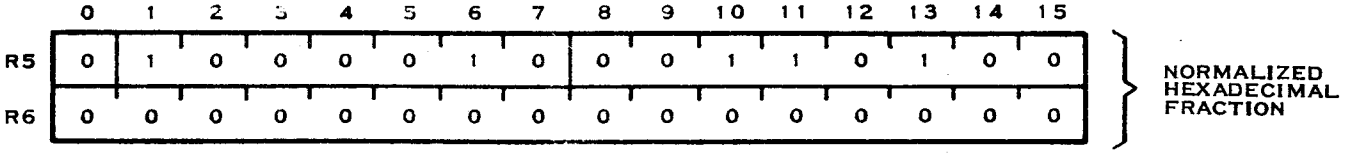
Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.



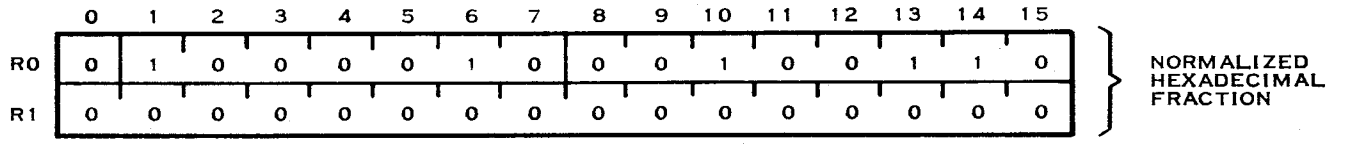
Execution results: FPA × (ga_s)→FPA

Application notes: The results of the MR instructions are compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If they are set to ones, overflow has occurred. If T_s is equal to three, the indicated register is incremented by four.

An example of a multiply real instruction is: If workspace registers five and six, after normalization, contain the value 34_{16}, shown figuratively below,



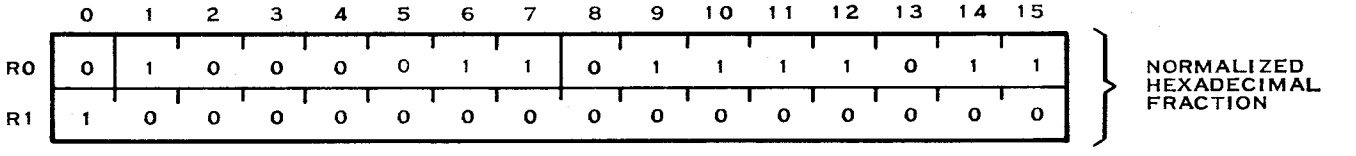
and the single precision FPA (R0-R1), after normalization, contains the value 26_{16}, as shown figuratively below,



then the instruction

LABEL MR R5

will multiply the contents of the FPA by the contents of R5 and R6, and place the results, $7B8_{16}$, in the FPA, as shown figuratively below.



The logical greater than and arithmetic greater than bits of the status register are set; and the equal, carry, and overflow bits of the status register are reset.

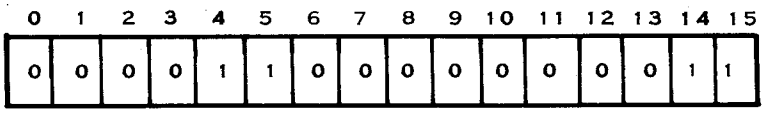
Refer to Section II for a detailed description of normalization and single precision instructions.

H.3.21 NEGATE DOUBLE PRECISION REAL – NEGD

Opcode: 0C03

Addressing mode: Format VII

Format:



Syntax definition:

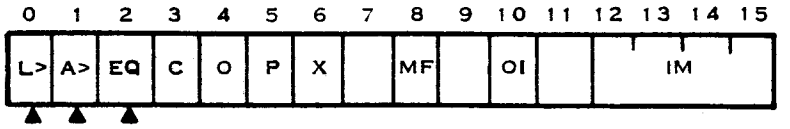
[<label>]b. . . NEGDb. . . [<comment>]

Example:

LABEL NEGD	Negate the double precision number in the FPA and place the results in the FPA.

Definition: Negate the double precision number in the FPA (R0-R3) and place the results in the FPA (R0-R3).

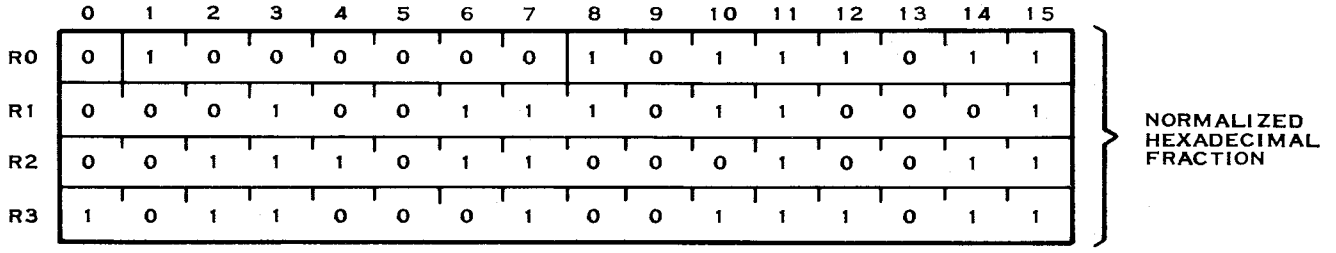
Status bits affected: Logical greater than, arithmetic greater than, and equal.



Execution results: -FPA→FPA

Application notes: The results of the NEGD instruction are compared to zero and status register bits zero, one, and two reflect the comparison.

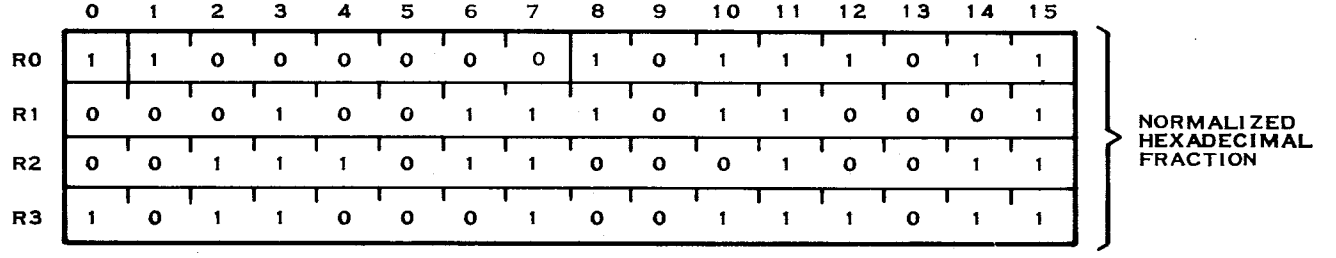
An example of the negate double precision real instruction is: If the value in the double precision FPA (R0-R3), after normalization, is .BB13B13B13B13B_{16}, shown figuratively below



then the instruction

LABEL NEGD

will negate (additive inverse) double precision value in the FPA, and place the result, -.BB13B13B13B13B_{16}, in the FPA as shown figuratively below.



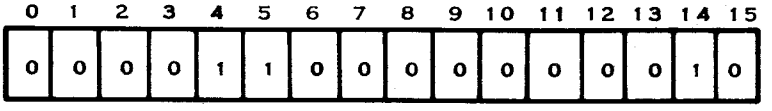
The logical greater than bit of the status register is set; and the arithmetic greater than and equal bits of the status register are reset.

H.3.22 NEGATE REAL – NEGR

Opcode: 0C02

Addressing mode: Format VII

Format:



Syntax definition:

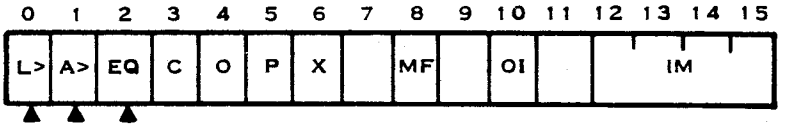
[<label>]b. . . NEGRb. . . [<comment>]

Example:

LABEL NEGR	Negate the real number in the FPA and place the result in the FPA.

Definition: The real number in the FPA (R0, R1) is negated and the result is placed in the FPA (R0, R1).

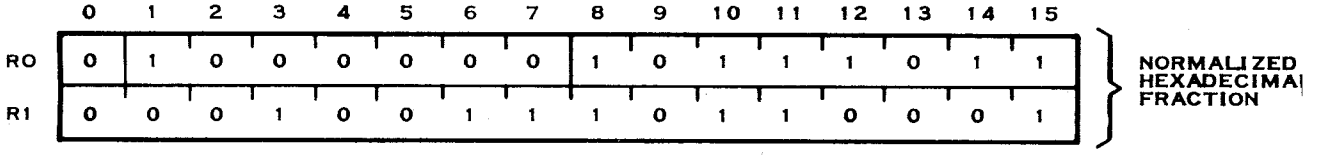
Status bits affected: Logical greater than, arithmetic greater than, and equal.



Execution results: -(FPA)→(FPA)

Application notes: The results of the NEGR instruction are compared to zero and status register bits zero, one, and two reflect the results.

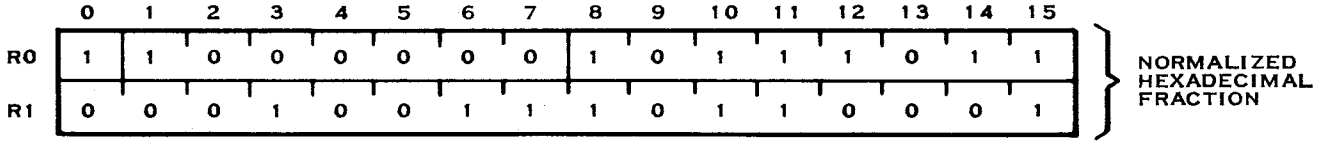
An example of the negate real instruction is: If the value in the single precision FPA (R0-R1), after normalization, is .BB13B1$_{16}$, shown figuratively below,



then the instruction

LABEL NEGR

will negate (additive inverse) the single precision value in the FPA, and place the result, -.BB13B1$_{16}$, in the FPA as shown figuratively below.



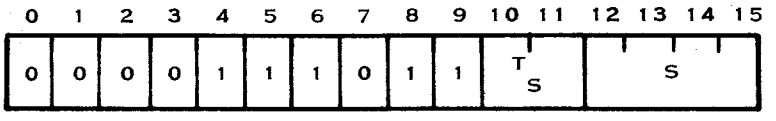
The logical greater than bit of the status register is set, and the arithmetic greater than and equal bits of the status register are reset.

H.3.23 SUBTRACT DOUBLE PRECISION REAL – SD

Opcode: 0EC0

Addressing mode: Format VI

Format:



Syntax definition:

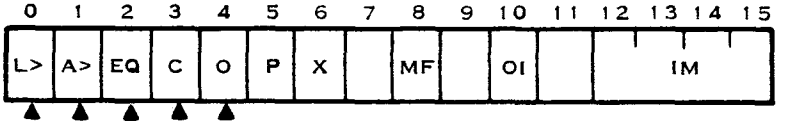
[<label>]b. . .SDb. . .<ga_s>b. . .[<comment>]

Example:

LABEL SD R5	Subtract the contents of workspace registers five through eight from the FPA and place the result in the FPA.

Definition: The four-word value at the source address is subtracted from the FPA (R0-R3). The result is placed in the FPA.

Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

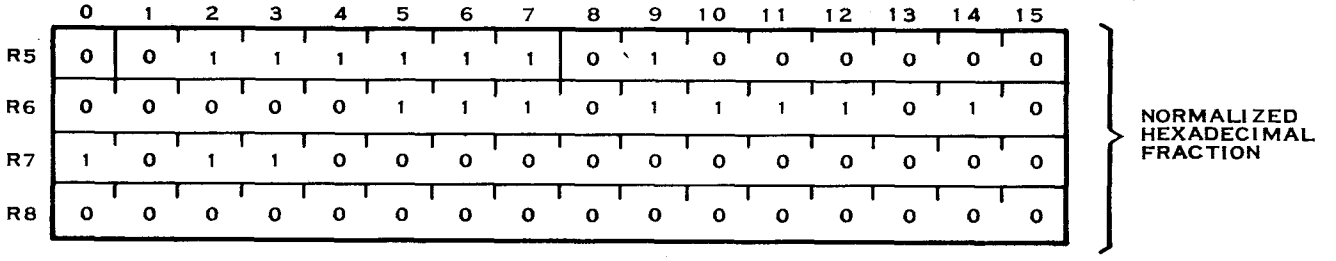


Execution results: (FPA) - (ga_s) →FPA

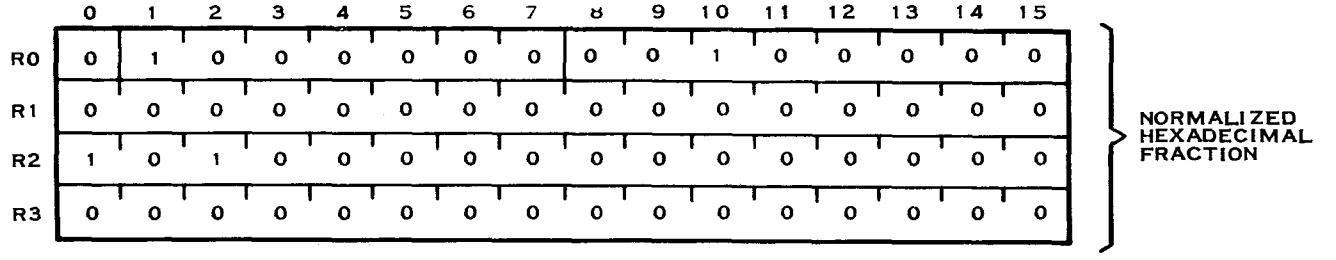
Application notes: The result of the SD instruction is compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If they are set to one, overflow has occurred.

If T_s is equal to three, the indicated register is incremented by eight.

An example of a subtract double precision real instruction is: If R5-R8, after normalization, contain the value .040077AB$_{16}$, as shown figuratively below,



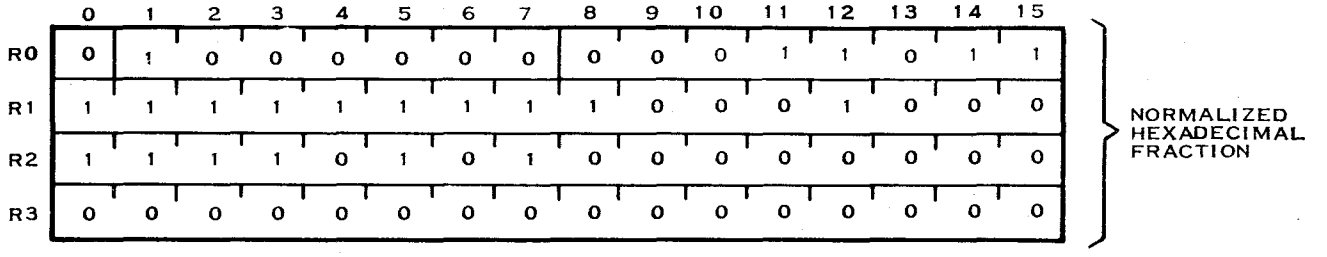
and the double precision FPA (R0-R3) contains, after normalization, the value .200000A$_{16}$, as shown figuratively below,



then the instruction

LABEL SD 5

will subtract the contents of R5-R8 from the FPA and place the result, .1BFF88F5$_{16}$, in the FPA, shown figuratively below.



The logical greater than and arithmetic greater than bits of the status register are set; and the equal, carry, and overflow bits of the status register are reset.

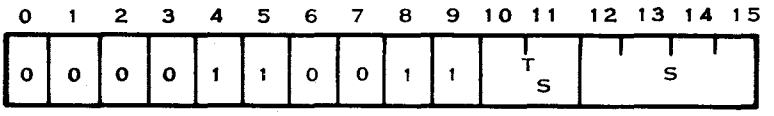
Refer to Section II for a detailed description of normalization and single precision floating point instructions.

H.3.24 SUBTRACT REAL – SR

Opcode: 0CC0

Addressing mode: Format VI

Format:



Syntax definition:

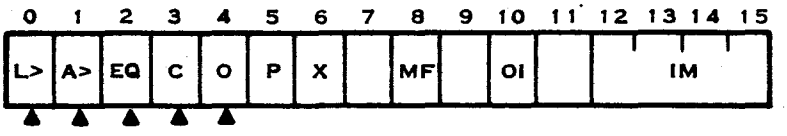
[<label>]␣. . .SR␣. . .<gas>␣. . .[<comment>]

Example:

LABEL SR @WORD	Subtract the contents of the two-word real number, beginning at location WORD, from the FPA and replace the FPA with the difference.

Definition: The two word normalized value at the source address is subtracted from the FPA and the result is stored in the FPA (R0-R1).

Status bits affected: Logical greater than, arithmetic greater than, equal, carry, and overflow.

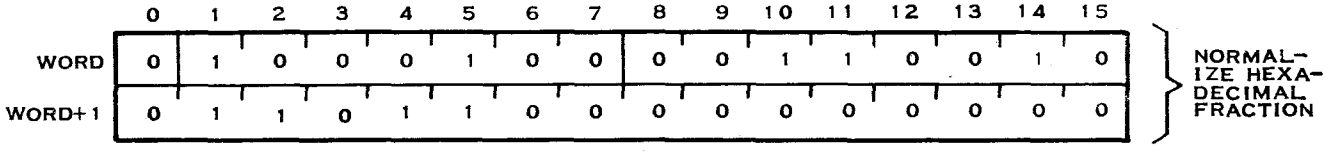


Execution results: FPA - $(ga_s) \rightarrow$FPA

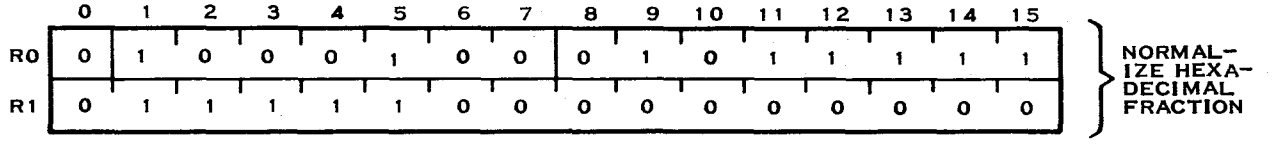
Application notes: The result of the SR instruction is compared to zero and status register bits zero, one, and two reflect the comparison. If status register bits three and four are set to zero and one, respectively, underflow has occurred. If they are set to ones, overflow has occurred.

If T_s is equal to three, the indicated register is incremented by four.

An example of the subtract real instruction is: If location WORD, after normalization, contains the value $326C_{16}$, as shown figuratively below,



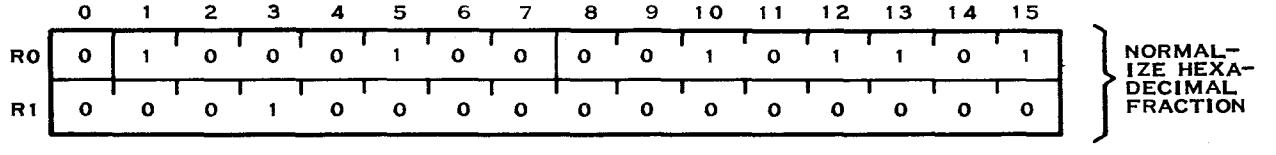
and the single precision FPA (R0-R1) after normalization, contains the value $5F7C_{16}$, as shown figuratively below,



then the instruction

LABEL SR @WORD

will subtract the contents of the two words, beginning at location WORD, from the FPA and place the result, $2D10_{16}$, in the FPA, shown figuratively below.

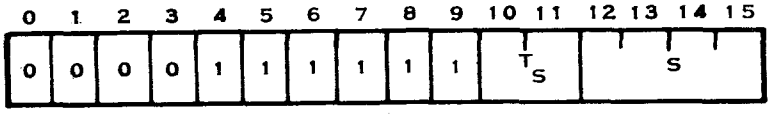


H.3.25 STORE DOUBLE PRECISION REAL – STD

Opcode: 0FC0

Addressing mode: Format VI

Format:



Syntax definition:

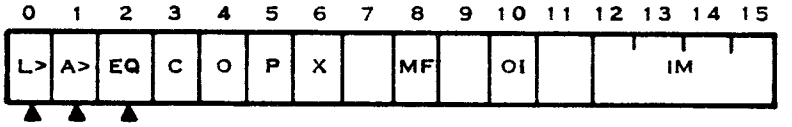
[<label>]b. . .STDb. . .<ga_s>b. . .[<comment>]

Example:

LABEL STD @WORD — Store the value in the floating point accumulator (R0-R3) in memory beginning at location WORD.

Definition: The value specified by FPA (R0-R3) is stored in the four words beginning at the address specified by the operand.

Status bits affected: Logical greater than, arithmetic greater than, and equal.



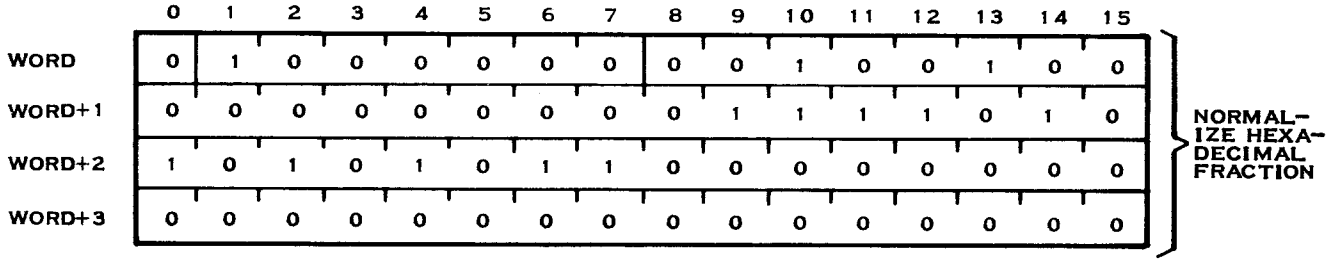
Execution results: FPA → (ga_s)

Application notes: The results of the STD instruction are compared to zero and status register bits zero, one, and two reflect the comparison. If T_s is equal to three, the indicated register is incremented by eight.

An example of the store double precision real instruction is: If the value contained in the four words of the double precision FPA (R0-R3), is .24007AAB$_{16}$, then the instruction

LABEL STD @WORD

will store the normalized fraction in the four words specified by WORD, as shown figuratively below.



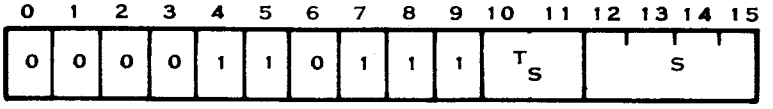
The logical greater than and arithmetic greater than bits of the status register are set; and the equal bit is reset.

H.3.26 STORE REAL – STR

Opcode: 0DC0

Addressing mode: Format VI

Format:



Syntax definition:

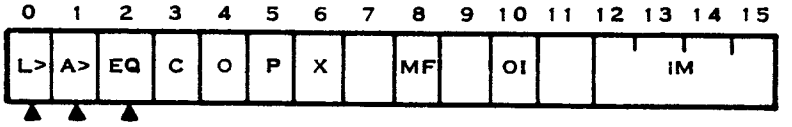
[<label>]b. . .STRb. . .<ga_s>b. . .[<comment>]

Example:

LABEL STR R6	Store the real number in the FPA in workspace registers six and seven.

Definition: The real number in the FPA (R0-R1) is stored at the source address.

Status bits affected: Logical greater than, arithmetic greater than, and equal.



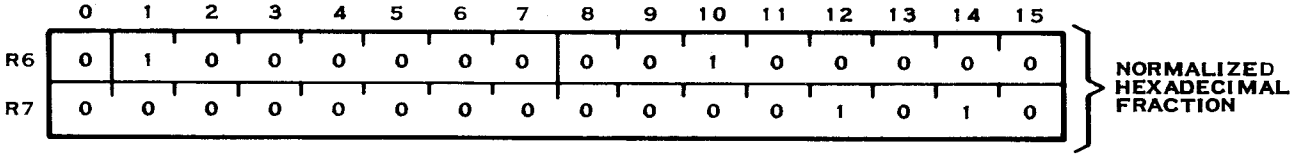
Execution results: FPA → (ga_s)

Application notes: The result of the STR instruction is compared to zero and status register bits zero, one, and two reflect the comparison. If T_s is equal to three, the indicated register is incremented by four.

An example of the store real instructions is: If the value contained in the single precision FPA (R0-R1), after normalization, is .20000A_{16}, then the instruction

LABEL STR R6

will store the normalized fraction in workspace register six and workspace register seven, as shown figuratively below:



The logical greater than and arithmetic greater than bits of the status register are set; and the equal bit is reset.

APPENDIX I

FLOATING-POINT BENCHMARKS EXECUTED ON TM 990/1481 AND TM 990/101MA BOARDS

I.1 GENERAL

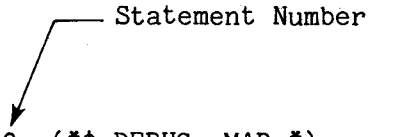
To measure the performance of a given processor to do a certain function, the same function must be performed by another processor so that a comparison can be made. This act is called "benchmarking." In comparing processor performance, one looks at how long it takes to execute a given set of instructions or a certain function (e.g., floating-point arithmetic). To determine the performance of the TM 990/1481 processor, it was compared to a TM 990/101MA performing a like function. The function is defined as a Pascal floating-point routine (listing in Figure I-1) which has been compiled to native code (TM 990 assembly language, shown in Figure I-2) and run under Texas Instruments Real Time Executive (RX). This program executed 39 times faster in a TM 990/1481 system than it did in a TM 990/101MA system. The Pascal program operated in the following environments on the respective boards:

- Because the TM 990/101MA does not execute floating point instructions, these instructions were emulated in software by the FP$INT subroutine. The FP$INT subroutine is part of the Microprocessor Pascal Run Time Library. The instructions were executed out of a TM 990/203-13 memory board.

- Calls to the FP$INT subroutine were deleted for TM 990/1481 system execution because the TM 990/1481 can execute floating point instructions in hardware. The instructions were executed out of a TM 990/203-13 memory board.

I.2 BENCHMARK LISTINGS

Figure I-1 is a listing of the Pascal program used. Figure I-2 is a listing of the same program with comparable assembly language statements along with the compiled Pascal statements.




```
 0  (*$ DEBUG, MAP *)
 0  PROGRAM BENCHW;
 0  VAR       X,Y,Z: REAL;
12
 0  PROCEDURE MESG; EXTERNAL;
 0
 1  BEGIN (*# STACKSIZE = 500 *)
 1  CRUBASE(#220);                  " SET TO PARALLEL I/O PORT
 2  LDCR(16,-1);                    " INITIALIZE 9901
 3  Y := 200;
 4  X := 100;
 5  Y := 200;
 6  MESG;
 7  WHILE TRUE DO
 8      BEGIN                                                    S
 8        SBZ(0);          " TRIGGER SOURCE: PIN 20 ON P-4       T
 9        SBO(0);          " RISE TO FALL TRANSITION             A  T
10        SBZ(1);          " START TIME TRACK: PIN 22 ON P-4     R  I
11        SBO(1);          " RISE TO FALL TRANSITION             T  M
12        Z := X +  Y;                                           /  E
13        Z := X -  Y;                                           S  R
14        Z := X *  Y;                                           T
15        Z := X /  Y;                                           O
16        Z := -X;                                               P
17        SBZ(3);          " STOP TIME TRACK: PIN 16 ON P-4
18        SBO(3);          " RISE TO FALL TRANSITION
19      END;
19  END.
```

FIGURE I-1. PASCAL BENCHMARK PROGRAM

```
*
       IDT  'BENCHW  '
*
*
       DEF  SYSTM$
       REF  MESG
       REF  S$PRCS
       REF  E$PRCS
       REF  CALL$
       REF  FP$INT
       REF  EXIT$P
       REF  IR$LO
*
       PSEG
SYSTM$ EQU  $
PR     EQU  R7
CODE   EQU  R8
LF     EQU  R9
SP     EQU  R10
LO     EQU  $
       DATA LO016-LO
       DATA LO0F4-LO
       DATA  0000
       DATA  0000
       DATA IR$LO
D000A  DATA  000C
D000C  DATA  0001
D000E  DATA  01F4
D0010  DATA  FFFF
D0012  DATA  42C8
D0014  DATA  4264
*
L0016  EQU  $
       MOV  @D000A-LO(CODE),*SP+
       MOV  @D000C-LO(CODE),*SP+
       SETO *SP+
       MOV  @D000E-LO(CODE),*SP+
       CLR  *SP+
       DATA CALL$,S$PRCS
*  *   *   *    *    *    *
* STATEMENT NUMBER     0
*
* INITIALIZATION CODE
*  *   *  *  *   *  *   * *
       MOV  LF,R6
       AI   R6, 0004
       MOV  LF,R5
       AI   R5, 0008
```

FIGURE I-2. PASCAL BENCHMARK PROGRAM SHOWING ASSEMBLY CODE (Sheet 1 of 4)

```
*   *   *   *   *   *   *  *
* STATEMENT NUMBER     1
*
* BEGIN (*# STACKSIZE = 500 *)
* CRUBASE(#220);                  " SET TO PARALLEL I/O PORT
*  *   *   *    *    *     *
       LI   R12, 0220
*  *   *   *    *   *    *
* STATEMENT NUMBER     2
*
* LDCR(16,-1);                    " INITIALIZE 9901
*  *   *    *    *    *    *
       LDCR @D0010-L0(CODE),0
*  *   *  *  *   *   *  *
* STATEMENT NUMBER     3
*
* Y := 200;
*   *   *   *    *    *    *
       MOV  @D0012-L0(CODE),*R6
       CLR  @ 0002(R6)
*   *   *   *   *    *    *
* STATEMENT NUMBER     4
*
* X := 100;
*   *    *    *    *    *
       MOV  @D0014-L0(CODE),*LF
       CLR  @ 0002(LF)
*  *    *    *    *    *
* STATEMENT NUMBER     5
*
* Y := 200;
*   *   *    *   *   *    *
       MOV  @D0012-L0(CODE),*R6
       CLR  @ 0002(R6)
*   *   *   *    *    *    *
* STATEMENT NUMBER     6
*
* MESG;
*   *   *    *    *   *    *
       DATA CALL$,MESG
*   *    *   *    *   *    *
* STATEMENT NUMBER     7
*
* WHILE TRUE DO
*   *   *    *    *    *
L007E  EQU  $
*   *    *    *    *    *
* STATEMENT NUMBER     8
*
* BEGIN
* SBZ(0);                         " TRIGGER SOURCE: PIN 20 ON P-4
*  *   *    *    *   *   *
       SBZ  0
```

FIGURE I-2. PASCAL BENCHMARK PROGRAM SHOWING ASSEMBLY CODE (Sheet 2 of 4)

```
*   *   *    *    *    *
* STATEMENT NUMBER     9
*
* SBO(0);                    " RISE TO FALL TRANSITION
*   *   *   *    *    *    *
       SBO  0
*   *   *    *    *    *
* STATEMENT NUMBER    10
*
* SBZ(1);                    " START TIME TRACK: PIN 22 ON P-4
*   *    *    *    *    *
       SBZ  1
*   *    *    *     *    *
* STATEMENT NUMBER    11
*
* SBO(1);                    " RISE TO FALL TRANSITION
*    *    *    *    *    *
       SBO  1
*   *    *    *    *    *
* STATEMENT NUMBER    12
*
* Z := X +  Y;
*   *    *   *    *    *   *
       DATA CALL$,FP$INT
       LR   *LF
       AR   *R6
       STR  *R5
       XIT
*   *    *   *     *     *
* STATEMENT NUMBER    13
*
* Z := X -  Y;
*   *   *   *    *    *    *
       DATA CALL$,FP$INT
       LR   *LF
       SR   *R6
       STR  *R5
       XIT
*   *   *    *    *   *    *
* STATEMENT NUMBER    14
*
* Z := X *  Y;
*   *    *   *    *    *
       DATA CALL$,FP$INT
       LR   *LF
       MR   *R6
       STR  *R5
       XIT
```

FIGURE I-2. PASCAL BENCHMARK PROGRAM SHOWING ASSEMBLY CODE (Sheet 3 of 4)

```
*  *   *   *    *    *    *
* STATEMENT NUMBER    15
*
* Z := X /  Y;
*  *   *   *   *   *   *   *
       DATA CALL$,FP$INT
       LR   *LF
       DR   *R6
       STR  *R5
       XIT
*  *   *   *   *   *   *   *
* STATEMENT NUMBER    16
*
* Z := -X;
*  *   *    *   *   *   *
       DATA CALL$,FP$INT
       LR   *LF
       NEGR
       STR  *R5
       XIT
*  *   *   *   *   *   *   *
* STATEMENT NUMBER    17
*
* SBZ(3);                   " STOP TIME TRACK: PIN 16 ON P-4
*  *   *   *   *   *   *   *
       SBZ  3
*  *   *   *   *   *   *   *
* STATEMENT NUMBER    18
*
* SBO(3);                   " RISE TO FALL TRANSITION
*  *   *  *   *  *   *   *
       SBO  3
       JMP  L007E
L00F4  EQU  $
*  *   *   *   *   *  *   *
* STATEMENT NUMBER    19
*
* END;
* END.
*  *   *   *   *  *   *    *
       MOV  @D000C-L0(CODE),*SP+
       DATA CALL$,E$PRCS
       B    @EXIT$P
       END
```

FIGURE I-2. PASCAL BENCHMARK PROGRAM SHOWING ASSEMBLY CODE (Sheet 4 of 4)