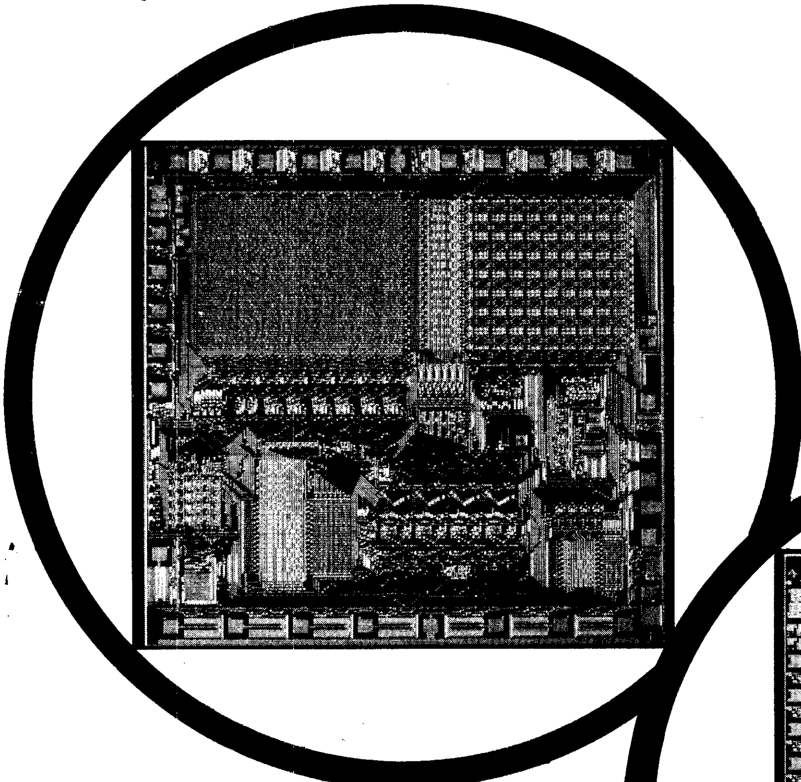
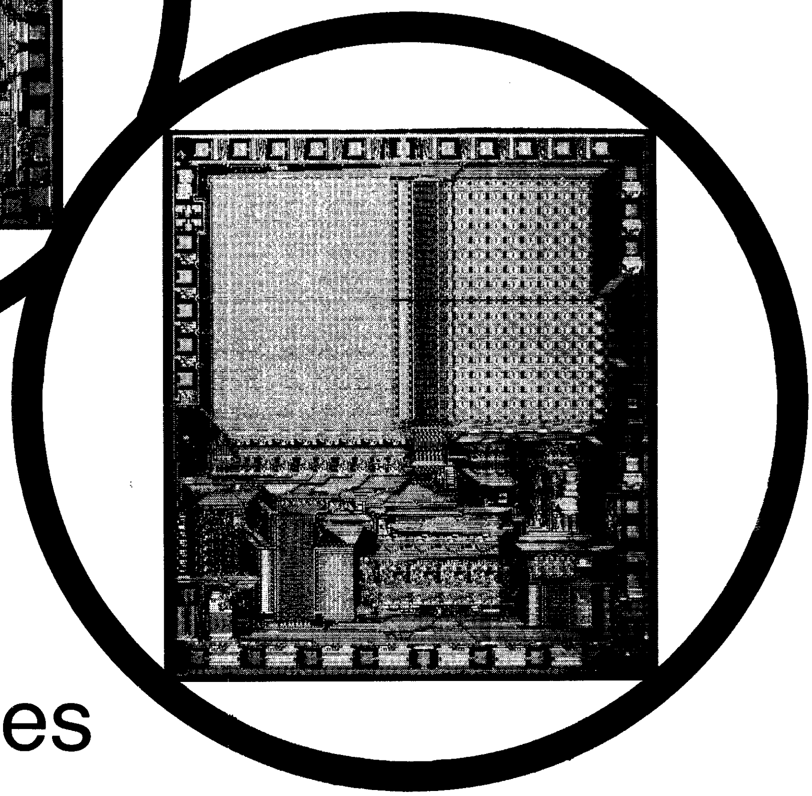
The Engineering Staff of
TEXAS INSTRUMENTS INCORPORATED
Semiconductor Group



Programmer's Reference Manual



TMS 1000 Series MOS/LSI One-Chip Microcomputers

TEXAS INSTRUMENTS
INCORPORATED

$4.95

CM122-1
1275

TEXAS INSTRUMENTS MOS
ONE-CHIP 4-BIT MICROCOMPUTER FAMILY

	TMS 1000	TMS 1200	TMS 1070	TMS 1270	TMS 1100	TMS 1300
Package Pin Count	28 Pins	40 Pins	28 Pins	40 Pins	28 Pins	40 Pins
Instruction Read Only Memory	1024 X 8 Bits (8,192 Bits)		1024 X 8 Bits (8,192 Bits)		*2048 X 8 Bits (16,384 Bits)*	
Data Random Access Memory	64 X 4 Bits (256 Bits)		64 X 4 Bits (256 Bits)		*128 X 4 Bits (512 Bits)*	
"R" Individually Addressed Output Latches	11	13	11	13	11	*16*
"O" Parallel Latched Data Outputs	8 Bits		8 Bits	10 Bits	8 Bits	
Maximum-Rated Voltage (O, R, and K)	20 V		*35 V*		20 V	
Working Registers	2-4 Bits Each		2-4 Bits Each		2-4 Bits Each	
Instruction Set	See Table 3-1, page 3-2		See Table 3-1, page 3-2		See Table 7-1, page 7-2	
Programmable Instruction Decoder	Yes		Yes		Yes	
On-Chip Oscillator	Yes		Yes		Yes	
Power Supply/Typical Dissipation	15 V/90 mW		15 V/90 mW		15 V/110 mW	
Time-Share Assembler Support	Yes		Yes		Yes	
Time-Share Simulator Support	Yes		Yes		Yes	
Hardware Evaluator and Debugging Unit	HE-2		HE-2		HE-2	
System Evaluator Device with External Instruction Memory	SE-1 (TMS 1099 JL)		SE-1 (TMS 1099 JL)		SE-2 (TMS 1098 JL)	

TI worldwide sales offices

ALABAMA

P.O. Box 2237, 304 Wynn Drive
Huntsville, Alabama 35804
205-837-7530

ARIZONA

4820 N. Black Canyon Hwy.
Suite 202
Phoenix, Arizona 85017
602-248-8028

CALIFORNIA

831 S. Douglass St.
El Segundo, California 90245
213-973-2571

Balboa Towers Bldg., Suite 805
5252 Balboa Avenue
San Diego, California 92117
714-279-2622

1505 East 17th St., Suite 201
Santa Ana, California 92701
714-835-9031

776 Palomar Avenue
Sunnyvale, California 94086
408-732-1840

COLORADO

9725 E. Hampden St., Suite 301
Denver, Colorado 80231
303-751-1780

CONNECTICUT

35 Worth Avenue
Hamden, Connecticut 06518
203-281-0074

FLORIDA

601 W. Oakland Park Blvd.
Fort Lauderdale, Florida 33311
305-566-3294

2221 Lee Road, Suite 108
Winter Park, Florida 32789
305-644-3535

GEORGIA

Route 1, Creekwood Pass
Dallas, Georgia 30132
404-445-4908

ILLINOIS

1701 Lake Avenue, Suite 300
Glenview, Illinois 60025
312-729-5710

INDIANA

3705 Rupp Drive
Arch Bldg.
Fort Wayne, Indiana 46805
219-484-0606

2346 S. Lynhurst Dr., Suite 101
Indianapolis, Indiana 46241
317-248-8555

MASSACHUSETTS

504 Totten Pond Road
Waltham, Mass. 02154
617-890-7400

MICHIGAN

Central Park Plaza
26211 Central Park Blvd., Suite 215
Southfield, Michigan 48076
313-353-0830

MINNESOTA

A.I.C. Bldg., Suite 202
7615 Metro Blvd.
Edina, Minn. 55435
612-835-2900

NEW JERSEY

1245 Westfield Ave.
Clark, New Jersey 07066
201-574-9800

NEW MEXICO

1101 Cardenas Drive, N.E.,
Room 215
Albuquerque, New Mexico 87110
505-265-8491

NEW YORK

144 Metro Pk.
Rochester, New York 14623
716-461-1800

6700 Old Collamer Rd.
East Syracuse, New York 13057
315-463-9291

P.O. Box 618, 112 Nanticoke Ave.
Endicott, New York 13760
607-785-9987

167 Main Street
Fishkill, New York 12524
914-896-6793

2 Huntington Quadrangle, Suite 2N04
Huntington Station, New York 11746
516-293-2560

NORTH CAROLINA

3631 Westfield
High Point, N.C. 27260
919-869-3651

OHIO

28790 Chagrin Blvd., Suite 120
Cleveland, Ohio 44122
216-464-2990

Hawley Bldg., Suite 101
4140 Linden Avenue
Dayton, Ohio 45432
513-253-3121

OREGON

10700 S.W. Beaverton Hwy., Suite 111
Beaverton, Oregon 97005
503-643-1182

PENNSYLVANIA

275 Commerce Drive
Fort Washington, Pa. 19034
215-643-6450

TEXAS

Headquarters – Gen. Offices
Dallas, Texas 75222
214-238-2011

MS366–P.O. Box 5012
Dallas, Texas 75222
214-238-6805

3939 Ann Arbor
Houston, Texas 77042
713-785-6906

VIRGINIA

8512 Trabue Road
Richmond, Virginia 23235
804-320-3830

WASHINGTON

700 112th N.E., Suite 101
Bellevue, Washington 98004
206-455-3480

WASHINGTON, D.C.

1500 Wilson Blvd., Suite 1100
Arlington, Virginia 22209
703-525-0336

ARGENTINA

Texas Instruments Argentina S.A.I.C.F.

C.C. Box 2296 Correo Central
Buenos Aires, Argentina
748-1141

ASIA

Texas Instruments Asia Limited

5F Aoyama Tower Bldg.
24-15 Minami Aoyama Chome
Minato-ku, Tokyo 107, Japan
402-6171

11A-15 Chatham Road
First Floor, Kowloon
Hong Kong
3-670061

Texas Instruments Singapore (PTE) Ltd.
27 Kallang Place
Singapore 1, Rep. of Singapore
258-1122

Texas Instruments Taiwan Limited
P.O. Box 3999
Taipei, Chung Ho, Taiwan
921 623

Texas Instruments Malaysia SDN. BHD.
Number 1 Lorong Enggang 33
Kuala Lampur 15-07, Malaysia
647 911

AUSTRALIA

Texas Instruments Australia Ltd.

Suite 205, 118 Great North Road
Five Dock N.S.W. 2046 Australia
831-2555

Box 63, Post Office
171-175 Philip Highway
Elizabeth 5112 South Australia
255-2066

BRAZIL

Texas Instrumentos Electronicos
do Brasil Ltda.

Rua Joao Annes, 153-Lapa
Caixa Postal 30.103, CEP 01.000
Sao Paulo, SP, Brasil
260-2956

CANADA

Texas Instruments Incorporated

935 Montee De Liesse
St. Laurent H4T 1R2
Quebec, Canada
514-341-3232

5F Caesar Avenue
Ottawa 12
Ontario, Canada
613-825-3716

280 Centre Str. East
Richmond Hill (Toronto)
Ontario, Canada
716-856-4453

DENMARK

Texas Instruments Denmark

46D, Marielundvej
2730 Herlev, Denmark
(01) 91 74 00

FINLAND

Texas Instruments Finland OY

Fredrikinkatu 75, A7
Helsinki 10, Finland
44 71 71

FRANCE

Texas Instruments France

Boite Postale 5
06 Villeneuve-Loubet, France
31 03 64

La Boursidiere, Bloc A
R.N. 186, 92350 Le Plessis Robinson
630.23.43

30-31 Quai Rambaud
69 Lyon, France
42 78 50

GERMANY

Texas Instruments Deutschland GmbH

Haggerty Str. 1
8050 Freising, Germany
08161/80-1

Frankfurter Ring 243
8000 Munich 40, Germany
089/325011-15

Lazarettstrasse, 19
4300 Essen, Germany
02141/20916

Krugerstrasse 24
1000 Berlin 49, Germany
0311/74 44 041

Akazlenstrasse 22-26
6230 Frankfurt-Griesheim
Germany
0611/39 90 61

Steimbker Hof 8A
3000 Hannover, Germany
0511/55 60 41

Krefelderstrasse 11-15
7000 Stuttgart 50, Germany
0711/54 70 01

ITALY

Texas Instruments Italia SpA

Via Della Giustizia 9
20125 Milan, Italy
02-688 31 41

Via L. Mancinella 65
00199 Roma, Italy
06-83 77 45

Via Montebello 27
10124 Torino, Italy
011-83 22 76

MEXICO

Texas Instruments de Mexico S.A.

Poniente 116 #489
Col. Industrial Vallejo
Mexico City, D.F., Mexico
567-92-00

NETHERLANDS

Texas Instruments Holland N.V.

Entrepot Gebouw-Kamer 225
P.O. Box 7603
Schiphol-Centrum
020-17 36 36

NORWAY

Texas Instruments Norway A/S
Sentrumskontorene
Brugaten 1
Oslo 1, Norway
33 18 80

SWEDEN

Texas Instruments Sweden AB

S-104 40 Stockholm 14
Skeppargatan 26
67 98 35

UNITED KINGDOM

Texas Instruments Limited

Manton Lane
Bedford, England
0234-67466

PROGRAMMER REFERENCE CARD for TMS1000 SERIES MICROCOMPUTERS

TEXAS INSTRUMENTS
INCORPORATED

P.O. BOX 1443
HOUSTON, TEXAS 77001
713 494-5115

TMS 1100/1300 STANDARD INSTRUCTION SET

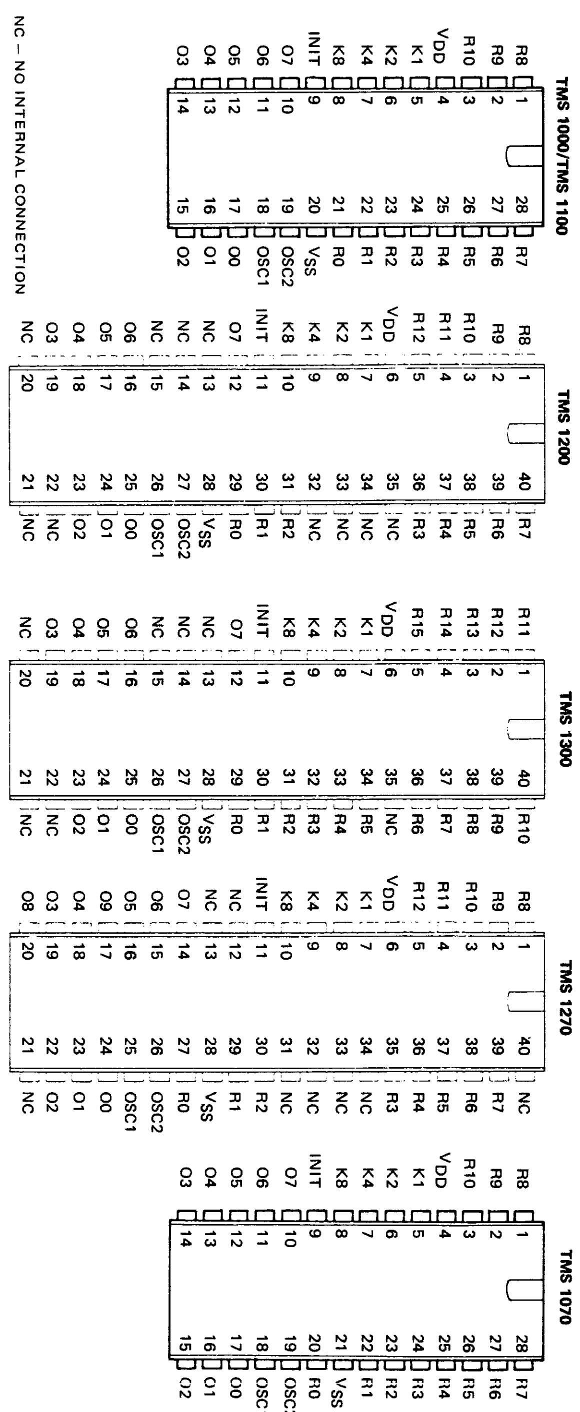
FUNCTION	MNEMONIC	STATUS EFFECT C	STATUS EFFECT N	DESCRIPTION	SYMBOLIC DESCRIPTION	FOR-MAT	OPE-RAND	HEX CODE
Register-to-Register Transfer	TAY			Transfer accumulator to Y register	A→Y	4		20
	TYA			Transfer Y register to accumulator	Y→A	4		23
	CLA			Clear accumulator	0→A	4		7F
Register to Memory	TAM			Transfer accumulator to memory	A→M(X,Y)	4		27
	TAMIYC	Y		Transfer accumulator to memory and increment Y register. If carry, one to status.	A→M(X,Y); Y+1→Y	4		25
	TAMDYN	Y		Transfer accumulator to memory and decrement Y register. If no borrow, one to status.	A→M(X,Y); Y−1→Y	4		24
	TAMZA			Transfer accumulator to memory and zero accumulator	A→M; 0→A	4		26
Memory to Register	TMY			Transfer memory to Y register	M(X,Y)→Y	4		22
	TMA			Transfer memory to accumulator	M(X,Y)→A	4		21
	XMA			Exchange memory and accumulator	M(X,Y)↔A	4		03
Arithmetic	AMAAC	Y		Add memory to accumulator, results to accumulator. If carry, one to status	M(X,Y)+A→A	4		06
	SAMAN	Y		Subtract accumulator from memory, results to accumulator. If no borrow, one to status.	M(X,Y)−A→A	4		3C
	IMAC	Y		Increment memory and load into accumulator. If carry, one to status	M(X,Y)+1→A	4		3E
	DMAN	Y		Decrement memory and load into accumulator. If no borrow, one to status.	M(X,Y)−1→A	4		07
	IAC	Y		Increment accumulator. If no carry, one to status.	A+1→A	4		70
	DAN	Y		Decrement accumulator. If no borrow, one to status.	A−1→A	4		77
	A2AAC	Y		Add 2 to accumulator. Results to accumulator. If carry one to status.	A+2→A	4		78
	A3AAC	Y		Add 3 to accumulator. Results to accumulator. If carry one to status.	A+3→A	4		74
	A4AAC	Y		Add 4 to accumulator. Results to accumulator. If carry one to status.	A+4→A	4		7C
	A5AAC	Y		Add 5 to accumulator. Results to accumulator. If carry one to status.	A+5→A Add	4		72
	A6AAC	Y		Add 6 to accumulator. Results to accumulator. If carry one to status.	A+6→A Immediate	4		7A
	A7AAC	Y		Add 7 to accumulator. Results to accumulator. If carry one to status.	A+7→A Value	4		76
	A8AAC	Y		Add 8 to accumulator. Results to accumulator. If carry one to status.	A+8→A To	4		7E
	A9AAC	Y		Add 9 to accumulator. Results to accumulator. If carry one to status.	A+9→A Accumulator	4		71
	A10AAC	Y		Add 10 to accumulator. Results to accumulator. If carry one to status.	A+10→A	4		79
	A11AAC	Y		Add 11 to accumulator. Results to accumulator. If carry one to status.	A+11→A	4		75
	A12AAC	Y		Add 12 to accumulator. Results to accumulator. If carry one to status.	A+12→A	4		7D
	A13AAC	Y		Add 13 to accumulator. Results to accumulator. If carry one to status.	A+13→A	4		73
	A14AAC	Y		Add 14 to accumulator. Results to accumulator. If carry one to status.	A+14→A	4		7B
	IYC	Y		Increment Y register. If carry, one to status.	Y+1→Y	4		05
	DYN	Y		Decrement Y register. If no borrow, one to status.	Y−1→Y	4		04
	CPAIZ	Y		Complement accumulator and increment. If then zero, one to status.	(Two's complement)	4		3D
Arithmetic Compare	ALEM	Y		If accumulator less than or equal to memory, one to status.	A≤M(X,Y)	4		01
Logical Compare	MNEA		Y	If memory is not equal to accumulator, one to status.	M(X,Y)≠A	4		00
	MNEZ		Y	If memory not equal to zero, one to status.	M(X,Y)≠0	4		3F
	YNEA		Y	If Y register not equal to accumulator, one to status and status latch.	Y≠A; S→SL	4		02
	YNEC		Y	If Y register not equal to a constant, one to status.	Y≠I(C)	2	C	5–
Bits in Memory	SBIT			Set memory bit.	1→M(X,Y,B)	3	B	3–
	RBIT			Reset memory bit.	0→M(X,Y,B)	3	B	3–
	TBIT1		Y	Test memory bit. If equal to one, one to status.	M(X,Y,B) = 1	3	B	3–
Constants	TCY			Transfer constant to Y register	I(C) → Y	2	C	4–
	TCMIY			Transfer constant to memory and increment Y	I(C) →M(X,Y); Y+1→Y	2	C	6–
Input	KNEZ		Y	If K inputs not equal to zero, one to status.	K≠0	4		0E
	TKA			Transfer K inputs to accumulator.	K→A	4		08
Output	SETR			Set R output addressed by Y.	1→R(Y)	4		0D
	RSTR			Reset R output addressed by Y.	0→R(Y)	4		0C
	TDO			Transfer data from accumulator and status latch to O outputs	A, SL→O REG	4		0A
RAM X Addressing	LDX			Load X with file address	I(F)→X	5	F	2–
	COMX			Complement the MSB of X	X̄MSB → XMSB	4		09
ROM Addressing	BR			Branch on status = one		1	W	––
	CALL			Call subroutine on status = one		1	W	––
	RETN			Return from subroutine		4		0F
	LDP			Load page buffer with constant	I(C)→PB	2	C	1–
	COMC			Complement chapter buffer	C̄B→CB	4		0B

CUT ALONG DOTTED LINE

FOLD

FOLD

TERMINALS



TMS 1000/1200 AND TMS 1070/1270 STANDARD INSTRUCTION SET

FUNCTION	MNEMONIC	STATUS EFFECTS C	STATUS EFFECTS N	DESCRIPTION	SYMBOLIC DESCRIPTION	FORMAT	OPERAND	HEX CODE
Register to Register	TAY			Transfer accumulator to Y register.	A→Y	4		24
	TYA			Transfer Y register to accumulator.	Y→A	4		23
	CLA			Clear accumulator.	0→A	4		2F
Transfer Register to Memory	TAM			Transfer accumulator to memory.	A→M(X,Y)	4		03
	TAMIY			Transfer accumulator to memory and increment Y register.	A→M(X,Y); Y+1→Y	4		20
	TAMZA			Transfer accumulator to memory and zero accumulator.	A→M(X,Y); 0→A	4		04
Memory to Register	TMY			Transfer memory to Y register.	M(X,Y)→Y	4		22
	TMA			Transfer memory to accumulator.	M(X,Y)→A	4		21
	XMA			Exchange memory and accumulator.	M(X,Y)↔A	4		2E
Arithmetic	AMAAC	Y		Add memory to accumulator, results to accumulator. If carry, one to status.	M(X,Y)+A→A	4		25
	SAMAN	Y		Subtract accumulator from memory, results to accumulator. If no borrow, one to status.	M(X,Y)–A→A	4		27
	IMAC	Y		Increment memory and load into accumulator. If carry, one to status.	M(X,Y)+1→A	4		28
	DMAN	Y		Decrement memory and load into accumulator. If no borrow, one to status.	M(X,Y)–1→A	4		2A
	IA			Increment accumulator, no status effect.	A+1→A	4		0E
	IYC	Y		Increment Y register. If carry, one to status.	Y+1→Y	4		2B
	DAN	Y		Decrement accumulator. If no borrow, one to status.	A–1→A	4		07
	DYN	Y		Decrement Y register. If no borrow, one to status.	Y–1→Y	4		2C
	A6AAC	Y		Add 6 to accumulator, results to accumulator. If carry, one to status.	A+6→A	4		06
	A8AAC	Y		Add 8 to accumulator, results to accumulator. If carry, one to status.	A+8→A	4		01
	A10AAC	Y		Add 10 to accumulator, results to accumulator. If carry, one to status.	A+10→A	4		05
	CPAIZ	Y		Complement accumulator and increment. If then zero, one to status.	(Two's complement)	4		2D
Arithmetic Compare	ALEM	Y		If accumulator less than or equal to memory, one to status.	A≤M(X,Y)	4		29
	ALEC	Y		If accumulator less than or equal to a constant, one to status.	A≤I(C)	2	C	7–
Logical Compare	MNEZ		Y	If memory not equal to zero, one to status.	M(X,Y)≠0	4		26
	YNEA		Y	If Y register not equal to accumulator, one to status and status latch.	Y≠A, S→SL	4		02
	YNEC		Y	If Y register not equal to a constant, one to status.	Y≠I(C)	2	C	5–
Bits in Memory	SBIT			Set memory bit.	1→M(X,Y,B)	3	B	3–
	RBIT			Reset memory bit.	0→M(X,Y,B)	3	B	3–
	TBIT1		Y	Test memory bit. If equal to one, one to status.	M(X,Y,B) = 1	3	B	3–
Constants	TCY			Transfer constant to Y register.	I(C)→Y	2	C	4–
	TCMIY			Transfer constant to memory and increment Y.	I(C)→M(X,Y); Y+1→Y	2	C	6–
Input	KNEZ		Y	If K inputs not equal to zero, one to status.	K≠0	4		09
	TKA			Transfer K inputs to accumulator.	K→A	4		08
Output	SETR			Set R output addressed by Y.	1→R(Y)	4		0D
	RSTR			Reset R output addressed by Y.	0→R(Y)	4		0C
	TDO			Transfer data from accumulator and status latch to O outputs.	A,SL→O REG	4		0A
	CLO			Clear O-output register.	0→O REG	4		0B
RAM 'X' Addressing	LDX			Load 'X' with a constant.	I(B)→X	2	B	3–
	COMX			Complement 'X'.	$\overline{X}$→X	4		00
ROM Addressing	BR			Branch on status = one.		1	W	––
	CALL			Call subroutine on status = one.		1	W	––
	RETN			Return from subroutine		4		0F
	LDP			Load page buffer with constant.	I(C)→PB	2	C	1–

FORMAT 1: W = BRANCH ADDRESS – I(2-7)
FORMAT 2: C = CONSTANT OPERAND I(7-4)
FORMAT 3: B = RAM-X OR BIT ADDRESS-I (7,6)
FORMAT 4: NO OPERANDS
FORMAT 5: F = FILE ADDRESS –I(7-5) TMS 1100 ONLY

TMS 1000 SERIES

PROGRAMMER'S REFERENCE MANUAL

IV TMS 1000/1200 STANDARD INSTRUCTION SET DEFINITIONS

TMS 1000 SERIES PROGRAMMER'S REFERENCE MANUAL (Continued)

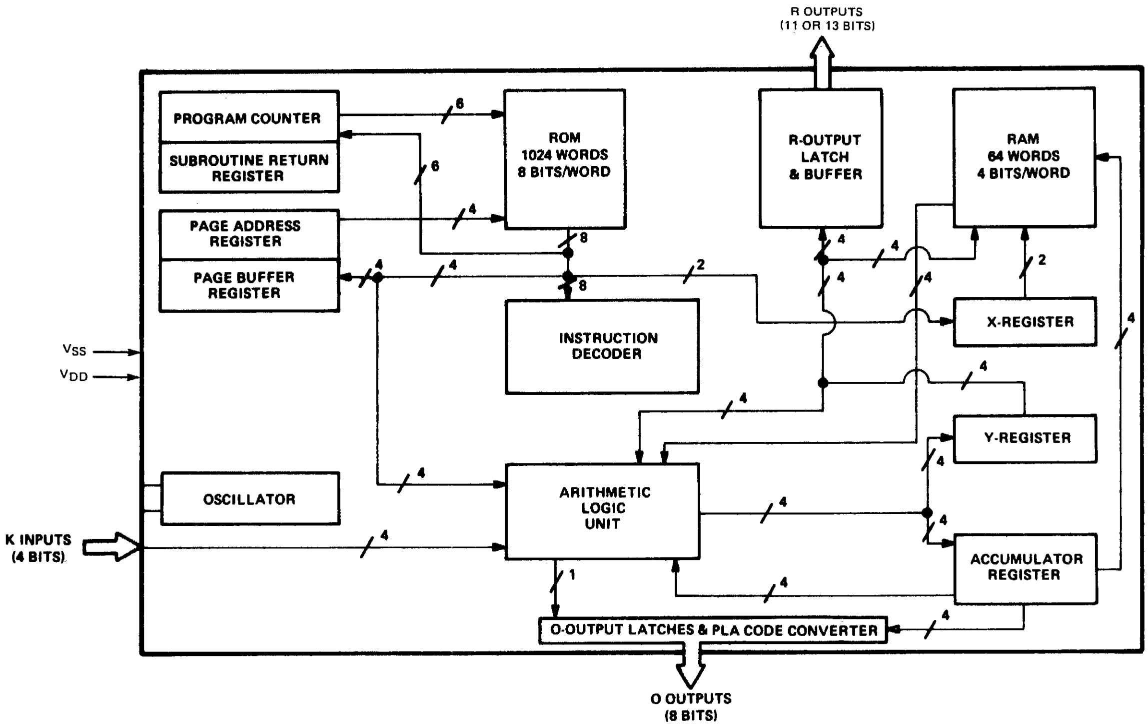


FIGURE 1-2.1 TMS1000/1200 LOGIC BLOCKS

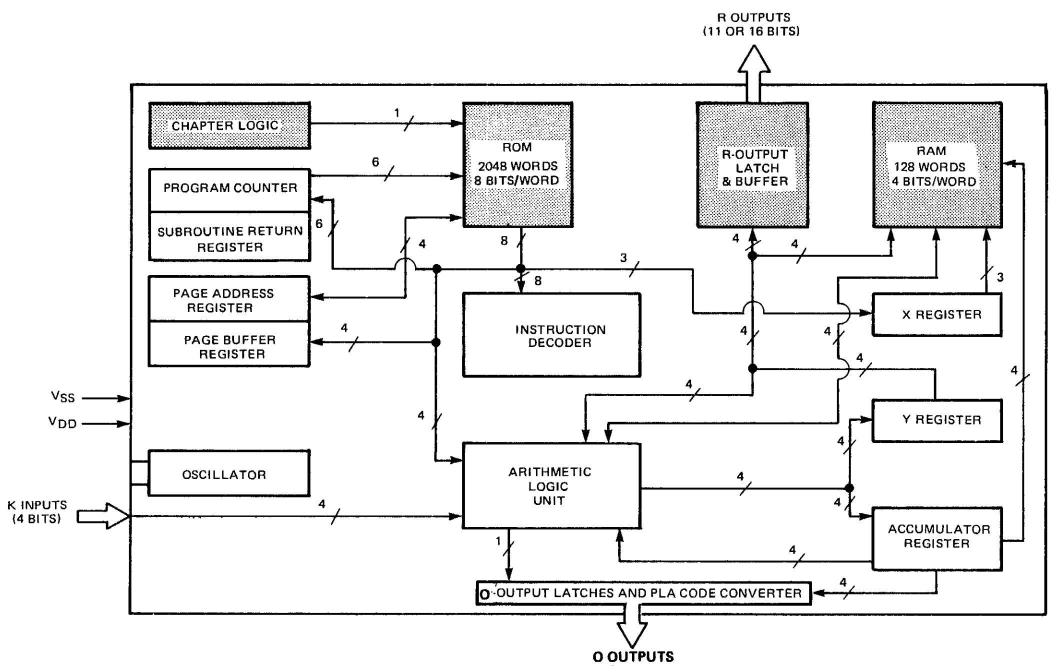


FIGURE 1-2.2 TMS1100/1300 LOGIC BLOCKS

SECTION I

INTRODUCTION

1-1 GENERAL.

This section introduces the TMS1000 series of one-chip microcomputers and outlines how an algorithm is developed and implemented to achieve cost effective designs. This introduction includes a definition of terms and conventions. This manual treats the devices as a system of logic blocks controlled by the programmer.

Since the hardware, detailed in Section 2, is so close to the software presented in Sections 3 to 9, it would be appropriate to label this book a "firmware" guide to TMS1000. After receiving the logical stepping-stone of hardware first, a user is presented with a detailed description of the standard instructions (Sections 3 and 4). Hints for efficient algorithms and example programs are presented last in Sections 10 to 14 since they require a thorough understanding of the standard instruction set.

In keeping with the teaching of firmware, the PLA programming concept is presented without assuming previous knowledge of MOS, and the appendices include electrical and timing specifications. This manual leads into a separate "TMS1000 Software User's Guide" which explains how to check out programs with software simulation before building prototypes of TMS1000 series circuits.

1-2 DESIGN FEATURES.

The TMS1000 series architecture is constructed to fit a wide variety of applications. The design is both cost effective and flexible because data input, processing and output are performed in one self-contained unit. An internal ROM, RAM and ALU comprise a single-chip microcomputer which functions according to the ROM program and the system inputs.

Systems with high volume requirements are inexpensive to produce and maintain since a system implemented with a single controlling device has high reliability, low pin count, and low power requirements. Several key features (seen in Figures 1-2.1 and 1-2.2) make low-cost products possible:

- Minimum system: One device containing ROM program, RAM, I/O control and ALU.
- 8-bit parallel O-output bus and up to 16 latched R-outputs.
- Format for the O-outputs is user defined by a PLA converting five input bits to an eight-bit code.
- Internal oscillator.
- Single power supply (15V).

The capabilities of the TMS1000 series four-bit microcomputer are limited by the magnitude of ROM instructions and RAM bits required. More complex systems are implemented cost effectively by using a multiple chip system with a central "master" controller chip and one or more "slave" devices. The slave devices controlled by the TMS1000 series microcomputer can be another TMS1000, PROM, RAM or other possibilities.

1-3 DESIGN STEPS.

It is important for the user to realize that each possible series or combination of inputs to the device must have a predetermined output forseen by the programmers and systems engineers. Upon completion of the programming phase of microcomputer ROM design, gate level tooling is generated for a fixed ROM pattern, and prototype devices are built at the expense of time and money. Thus, to help the designer be sure that his program is working correctly before releasing a ROM code to TI manufacturing, TI provides simulator and assembler programs. In addition, the software method of testing the program is supplemented with a hardware simulator which operates in real time. Whenever possible, the hardware simulator is preferred for final checkout because of simulation in real time. A software emulator, SE-1, which is a TMS1000 with external program memory, may be used for prototyping systems if the standard instruction set is employed.

Figure 1-3.1 shows typical design development steps for a TMS1000 series algorithm. The following numbered steps correspond to numbers in the figure.

(1) In the beginning of the program development, the inputs, outputs and RAM assignments are organized.

(2) After the I/O and RAM is organized, a flow chart of the program is generated to determine the instruction coding necessary to fulfill specification requirements.

(3) This ROM code is keypunched as a source program on a card deck, or it is entered through teletype keyboard on one of several national timeshare systems.

(4) The source program is assembled into an object program (mnemonics converted to machine-instruction bit-pattern), and assembler software generates a listing and possible error statements.

(5) After the source program errors have been removed, a simulator program duplicates the TMS1000 function as determined by the ROM program, and then the simulator generates a listing of the contents of the registers and the RAM in either a "snapshot" or an instruction-by-instruction trace.

(6) The ROM object program may be converted to a paper tape for input to the hardware simulator for real-time simulation.

(7) After the ROM code is approved, the assembler program generates the ROM object deck. The ROM object deck and the simulator option control cards specify how the microcomputer's instruction decoder, O-output decoder, and ROM patterns are to be defined. The definition files are sent to TI manufacturing for the generation of prototype gate masks, slices, and prototype circuits.

(8) After prototype devices are checked by the customer and approved, TI begins volume production from the masks that were used in the manufacture of prototype devices.

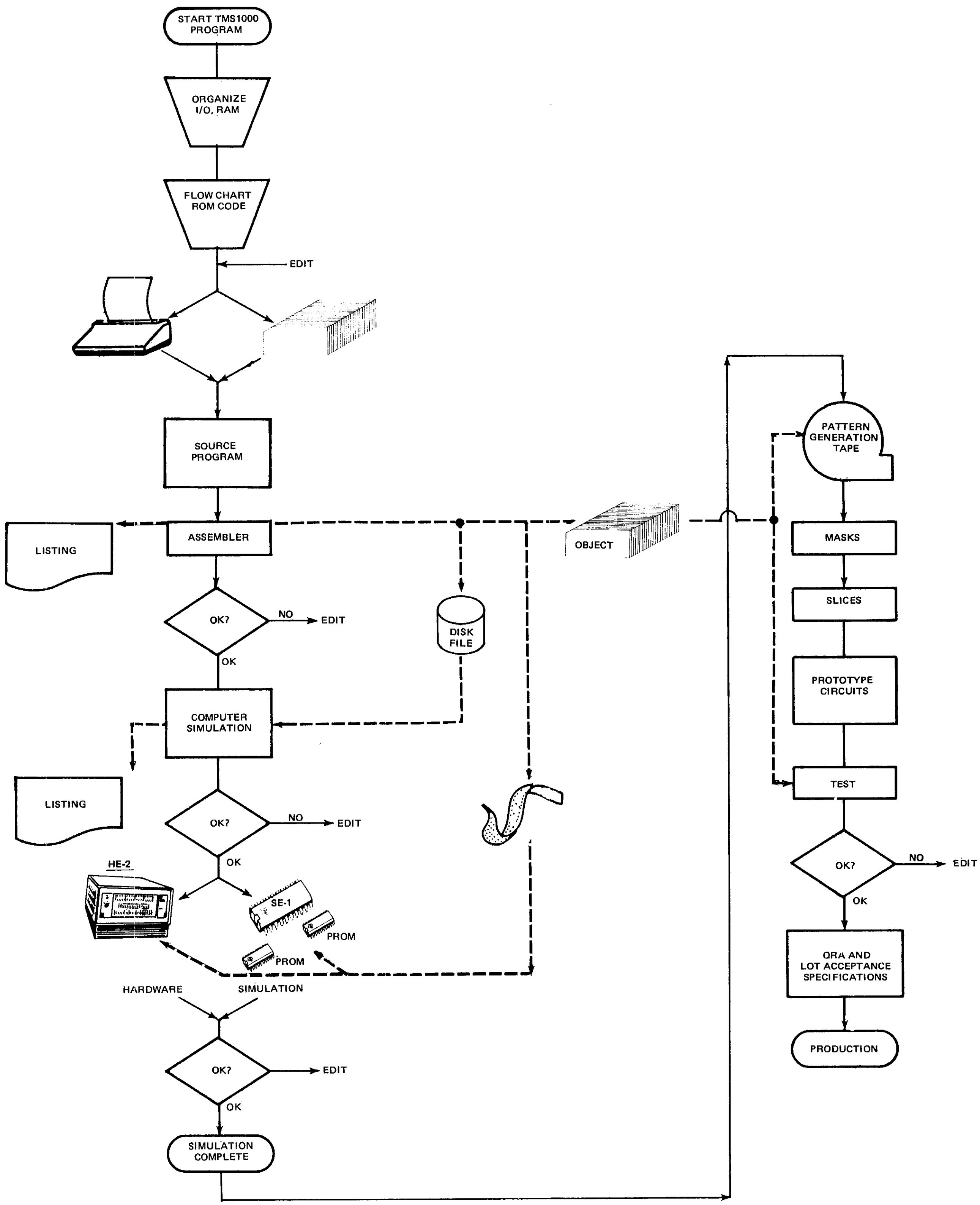


FIGURE 1-3.1 TMS 1000/1200 ALGORITHM DEVELOPMENT

1-4 SYMBOLS AND CONVENTIONS.

1-4.1 LIST OF ABBREVIATIONS.

A	Accumulator Register
ALU	Arithmetic Logic Unit (Adder-Comparator, P&N Inputs, ALU Select)
B	Bit Field of Instruction Word
C	Constant Field of Instruction Word
CA	Chapter Address Latch
CB	Chapter Buffer Latch
CKI	Constant and K-Input Logic (and Bus)
CL	Call Latch
CS	Chapter Subroutine Latch
DIP	Dual In-line Package
F	File Address Field of the Instruction Word
I( )	Instruction Field
Ki	K input terminals
LSB	Least Significant Bit
LSD	Least Significant Digit
LSI	Large Scale Integration
MOS	Metal Oxide Semiconductor
MSB	Most Significant Bit
MSD	Most Significant Digit
M(X,Y)	RAM Memory Location = X Address (0 to 7), Y Address (0 to F_{16})
M(X,Y,B)	RAM Memory Bit Location (B = 0, 1, 2, or 3)
O	Output Register
Ox	O-Output Terminal, x = 0-9.
O PLA	Output Programmable Logic Array
PA	Page Address Register (ROM)
PB	Page Buffer Register (ROM)
PC	Program Counter
PLA	Programmable Logic Array
R	R-Output Register
Rx	R-Output Terminal, x = 0-15
R(Y)	R-Output Latch Location = Y
RAM	Random Access Memory (Read/Write)
ROM	Read Only Memory
S	Status
SL	Status Latch
SR	Subroutine Return Register
W	Branch Address of Instruction Field
X	RAM X Address Register
Y	RAM Y Address Register

1-4.2 SYMBOLS AND LOGIC NOTATION.

$a \rightarrow b$	Transfer value a to b.
$c \rightarrow d$	Transfer the contents of register c to d.
$e \leftrightarrow f$	Exchange contents of e and f.
$\overline{X}$	One's complement of X.
$=$	equal
$\neq$	not equal
$>$	greater than
$\geqslant$	greater than or equal to
$<$	less than
$\leqslant$	less than or equal to
$+$	addition
$-$	subtraction
$+$	Boolean OR function
$\cdot$	Boolean AND function
ONE	set "1", high ($\approx V_{SS}$), Boolean true, logic one
ZERO	reset "0", low ($\approx V_{DD}$), Boolean false, logic zero
$PC + 1 \rightarrow PC$	PC value goes to next word address in the pseudo random sequence (0, 1, 3, 7, 15, etc.) The complete sequence is given on pages 14-5 and 14-6.

1-4.3 MACHINE-INSTRUCTION FLOWCHART CONVENTIONS. The conventions used for flowcharts are shown in Figure 1-4.1.

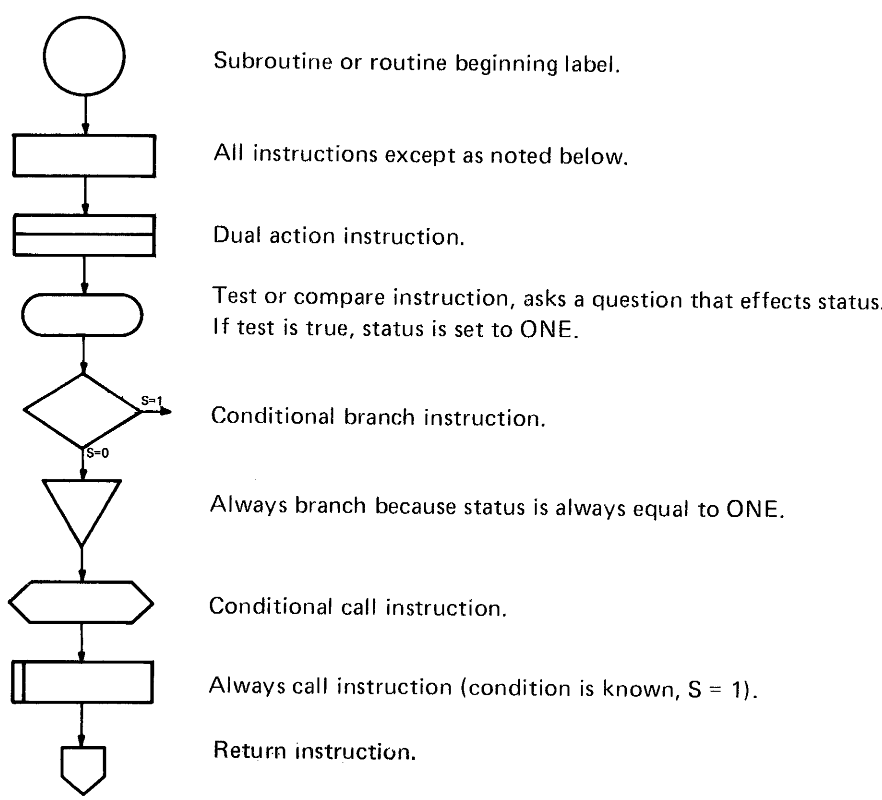


FIGURE 1-4.1. MACHINE INSTRUCTION FLOWCHART CONVENTIONS

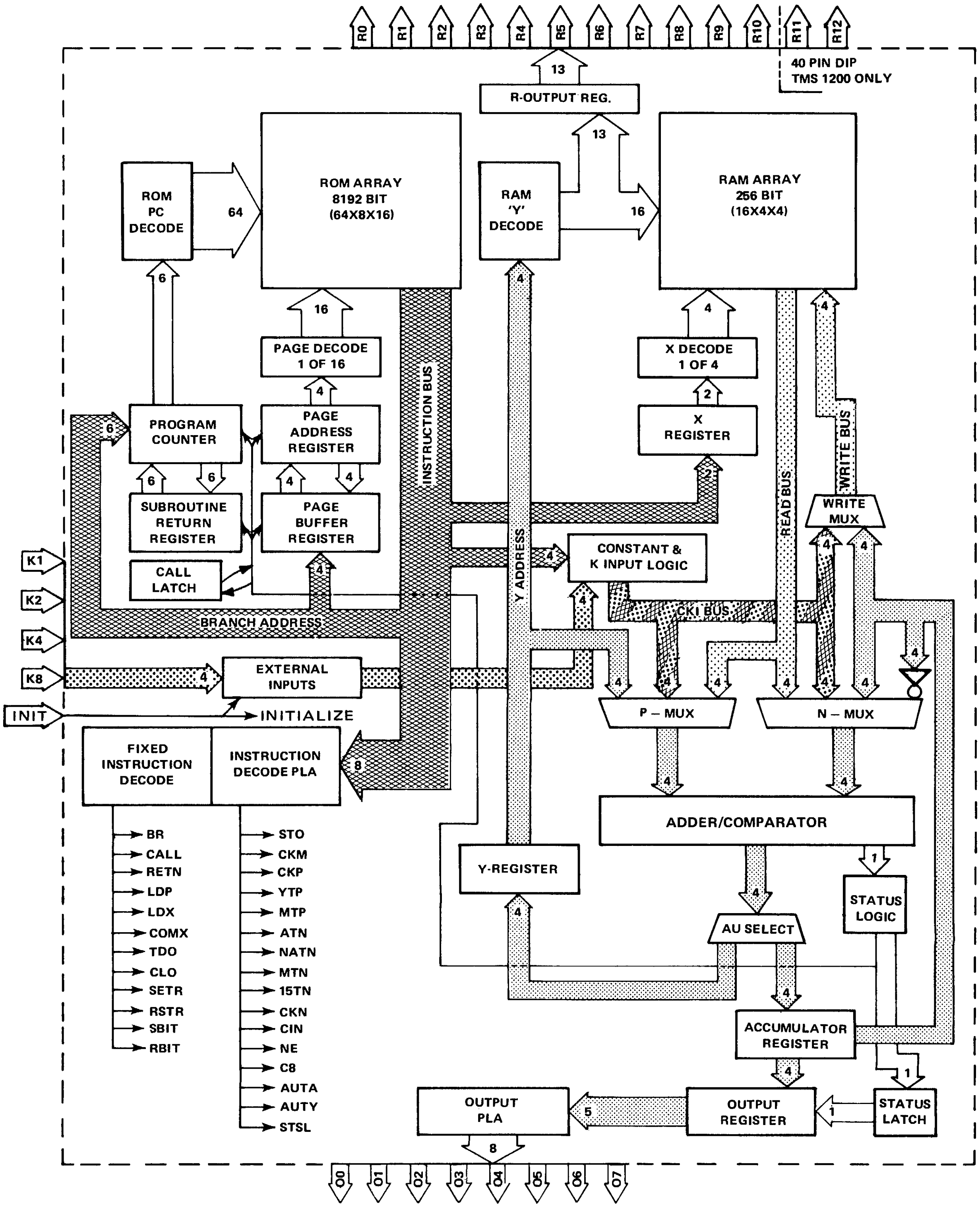


FIGURE 2-1.1 TMS1000/1200 BLOCK DIAGRAM

SECTION II
TMS1000/1200
CHIP ARCHITECTURE AND OPERATION

2-1 GENERAL.

The TMS1000/1200 functional block diagram (Figure 2-1.1) shows all major logic blocks and major data paths in the TMS1000/1200 architecture. The ROM, ROM addressing, and instruction decode are on the left side of the diagram. On the right side of the diagram are the adder/comparator, the RAM, the registers for addressing the RAM, and the accumulator which is the main working register. The major logic blocks are interconnected to the adder with four-bit parallel data paths. Table 2-1.1 identifies each major logic block and gives a brief description of its function. Each of these logic blocks is discussed in detail in the following paragraphs approximately in the numerical order shown in Figure 2-1.2 accompanying Table 2-1.1.

The instruction timing is fixed and each requires six oscillator cycles to execute. Each of the 43 basic instructions (listed in Table 3-1) is defined to enable one or more microinstructions that activate control lines during one instruction cycle. These microinstructions explain the firmware bridge between software instructions and the individual logic block capabilities. A hardwired logic decoder that cannot be modified decodes 12 "fixed" basic instruction codes into 12 fixed microinstructions for output instructions, branching, subroutines, RAM X addressing, reset and set bit instructions. The remaining 31 basic instructions activate a combination of 16 programmable microinstructions that are encoded by the instruction PLA. The concept of fixed and programmable microinstructions is used as a tool for understanding the software on the machine level and is used to increase the power of the instruction set to fit more applications (microprogramming the instruction set).

2-2 ROM ADDRESSING.

The ROM has 8,192 possible matrix points (1024 eight-bit words) where MOS transistors are placed to define the bit patterns of the machine language code. The ROM is organized into 16 pages of 64 words each (16 x 64 = 1024 words total). Each word contains eight bits.

Registers used to address the ROM include the following:

a. Page Address Register (PA). Contains the number of the page within the ROM being addressed. The contents of PA (four bits) are decoded into one of sixteen address lines by the page decoder.

b. Page Buffer Register (PB). The PB is loaded with a new page address which is then shifted into the PA for a successful branch or call. The PB is changed by the load page (LDP) instruction.

c. Program Counter (PC). Contains the current location of the word (within the page) being addressed. The contents of PC (six bits) are decoded by the PC decoder into one of 64 address lines selecting one instruction on a page.

Text continued on page 2-6

NOTE

Figure 2-1.2 identifies functional areas described in Table 2-1.1. These blocks are identified by numbers referenced in column 1 of Table 2-1.1.

This figure follows the outline of Figure 2-1.1, System Block Diagram.

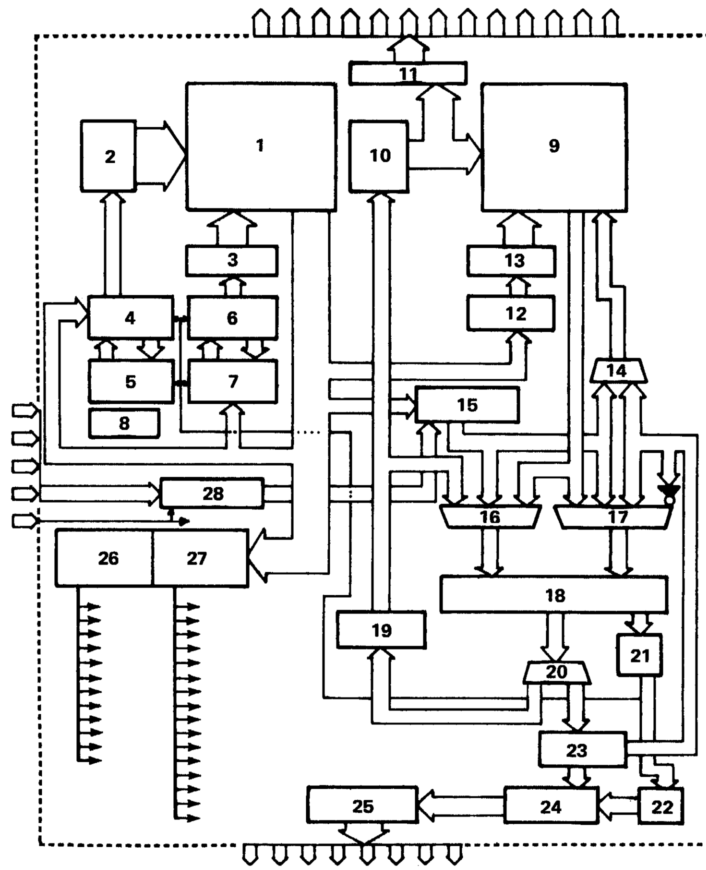


FIGURE 2-1.2 NUMBERED FUNCTIONAL BLOCKS

TABLE 2-1.1. TMS1000/1200 FUNCTIONAL BLOCKS

No. In Fig. 2-1.2	Block Name	Symbol (Abbr.)	Logic Type	Function and Organization
1	ROM Array	ROM	Virtual Ground ROM	Contains program bit pattern. 16 pages of 64 words, 8 bits each.
2	ROM PC Decode		Gates	Decodes program counter into one of 64 ROM addresses.
3	Page Decode		Gates	Decodes page address register into one of 16 page addresses.

TABLE 2-1.1. TMS1000/1200 FUNCTIONAL BLOCKS (CONTINUED)

No. In Fig. 2-1.2	Block Name	Symbol (Abbr.)	Logic Type	Function and Organization
4	Program Counter	PC	Shift Register	Contains the 6-bit code for the ROM instruction address.
5	Subroutine Return Register	SR	Storage Register	Contains 6-bit return address during the call state.
6	Page Address Register	PA	Storage Register	Contains 4-bit page address of the ROM instructions.
7	Page Buffer Register	PB	Storage Register	Used to set up page changes. Also contains 4-bit return page address during the call state.
8	Call Latch	CL	Latch	Stores the call state.
9	RAM Array	RAM M(X,Y)	Self Refresh RAM	Contains variable data. Organized by 64 four-bit words, four files of 16 words.
10	RAM Y Decode		Gates	Decodes the Y address register into one of 16 RAM address lines. Also selects one of 13 R lines for $0 \leqslant Y \leqslant 12$.
11	R-Output Register	R R(Y)	Single Bit RAM Cells	Latches for output to the R buffers.
12	X-Register	X	Storage Register	Contains 2 bits of RAM file address.
13	X Decode		Gates	Decodes X-register into one of four RAM page addresses.

TABLE 2-1.1. TMS1000/1200 FUNCTIONAL BLOCKS (CONTINUED)

No. In Fig. 2-1.2	Block Name	Symbol (Abbr.)	Logic Type	Function and Organization
14	Write MUX		Data Selector	Selects either constant and K inputs or the accumulator for writing into the RAM. Also performs bit setting and resetting.
15	Constant & K-Input Logic	CKI	Data Multiplexer	Selects either (1) constant field, (2) the K-Input to enter CKI data bus, or (3) a bit mask.
16	P-MUX		Data Multiplexer	Selects input to the adder from (1) Y, (2) CKI, or (3) RAM.
17	N-MUX		Data Multiplexer	Selects N input to the adder (1) RAM, (2) CK1, (3) accumulator, (4) $\overline{\text{accumulator}}$ or (5) F_{16}.
18	Adder/Comparator		Binary Adder (4 bit parallel)	Adds the P input and the N input with a possible carry. The resulting data and status effect are controlled by microinstructions. Also logically compares the P and N inputs.
19	Y-Register	Y	Storage Register	Four-bit multipurpose pointer and storage register. Y contains the RAM address for one of 16 possible words in a file. Y also addresses the R output register.
20	AU Select		Data Selector	Selects destination of the adder output to (1) Y-REG, (2) accumulator, or (3) neither.

TABLE 2-1.1. TMS1000/1200 FUNCTIONAL BLOCKS (CONTINUED)

No. In Fig. 2-1.2	Block Name	Symbol (Abbr.)	Logic Type	Function and Organization
21	Status Logic	S	Gates	Conditional branch control. Normal state = ONE. Branches are taken if S = ONE. Selectively outputs a ZERO when carry is false or when logical compare is true. A ZERO lasts for one instruction cycle only.
22	Status Output Latch	SL	Latch	Selectively stores status output.
23	Accumulator Register	A	Storage Register	Four-bit storage register, main data working register.
24	Output Register	O	Storage Register	Stores the accumulator and status latch data for transfer to the output buffers. Five bits.
25	Output PLA	O PLA	PLA	Decodes the O-register into a combination of the 8 output buffers. User defined.
26	Fixed Instruction Decoder			Fixed logic that decodes 8-bit instruction into the various fixed micro-instructions.
27	Instruction Decode PLA		PLA	30 term PLA that converts 8-bit instruction into a combination of 16 microinstructions.
28	External Inputs		Gates	Input buffers. Performs page and PC override for initializing and hardware reset.

d. Subroutine Return Register (SR). Contains the return word address in the call subroutine mode.

On power up, the program counter is reset to location zero, and the PA is set to 15. Then the program counter counts to the next ROM address in a pseudo random sequence. The sequence of addresses in the program counter can be altered by a branch instruction or a call instruction. A new branch address (W) can be stored into the program counter upon the completion of a successful branch or call instruction. If the branch instruction is not successful, then the program counter goes to the next ROM location within the current page.

In a successful call or branch execution the page address register (PA) receives its next page address from the buffer register (PB). The contents of the PB are changed by the load page instruction (LDP) which can be executed prior to the branch or call. If the PB is not changed, execution continues on the same page. In other words, when the program counter reaches the 64th word on a page, execution begins again at PC location 0 on that page.

2-3 BRANCHING.

All branches are conditional; a status logic path comes from the ALU to designate if a branch instruction should be successfully executed. A successfully executed branch or call is defined to be the case when the branch or call transfers control to an instruction address out of the normal sequence. An unsuccessful branch or call does not affect the normal sequence of the program counter.

- If the status logic equals ONE, then the branch is successfully executed. That is, six bits are transferred from the instruction bus from ROM into the program counter. These six bits are the branch address (W) which locates the next word on the page to be executed.

- If the status logic is equal to ZERO, then the branch instruction is unsuccessful. The program counter sequences to the next instruction, and then status reverts to a ONE.

When the branch is executed successfully and when not in the call mode (CL = 0), the page buffer register is loaded into the page address register. If the contents of the page buffer register had been modified previous to the branch instruction, then this instruction is called a long branch instruction, since it may branch anywhere in the ROM (a long branch, BL, directive in the source program generates two instructions - LDP, load page buffer and BR, branch). In the call mode (CL = 1), only "short" branches are possible, staying within a given page.

NOTE

The normal state of the status logic is ONE. Several instructions can alter this state to a ZERO; however, the ZERO state lasts for only one subsequent instruction cycle (which could be during a branch or call), then the status logic will normally revert back to its ONE state (unless the following instruction resets it to ZERO).

2-4 SUBROUTINES.

Similar to branch instructions, call instructions are conditional. One level of subroutine is permitted, and a call within a call does not execute properly. In the case of a successful call when status logic equals ONE:

(1) The call latch (CL) is set to ONE

(2) The contents of the page buffer register (PB) and the page address (PA) register are exchanged simultaneously.

(3) The return address is stored in SR and PB: the SR address is one address ahead of the program counter when the call instruction is executed. The return address is saved for a future return instruction.

(4) The branch address field of the instruction word writes into the program counter.

When a return instruction occurs:

(1) The subroutine return register (containing the call instruction address plus one) is always transferred to the program counter.

(2) The contents of the page buffer register (containing the page at call) is always transferred to the page address register.

(3) The call mode is reset (CL = 0).

If a call instruction is executed within a previous call (no return occurred and the call latch is still a ONE and status is a ONE), there is no transfer of the page buffer register to the page address register: instead contents of the page address register transfer to the page buffer register, although the branch address (W) loads into the program counter. For example:

(1) A call instruction is executed, transferring control from ROM page one to page two. Before execution, the PA and PB are as follows:

PA = 1 PB = 2

(2) After execution of the call:

PA = 2 PB = 1

(3) This subroutine contains another call. After execution of this second call:

PA = 2 PB = 2

Thus a call within a call to another page will cause the return page to change, losing the correct return page address (which is 1).

2-5 RAM ADDRESSING.

There are four RAM files, each containing 16 four-bit words in the RAM's 256-bit matrix (shown in the upper right of Figure 2-1.1 and in detail in Figure 2-5.1). Two registers are important in RAM addressing:

- The X register addresses (identifies) each file with a two-bit address (00 to 11), the address being decoded by the X decoder.

- The Y register identifies the particular word in the file with a four-bit address (0000 to 1111). The Y register is decoded by the Y decoder.

An X and Y address selects one four-bit RAM character, M(X,Y), this address being the storage location in the RAM matrix. The X register can be set to a constant equal to zero through three (LDX instruction), or X is complemented (COMX instruction) to flip the address of X to the $\overline{X}$ file (e.g., 00 to 11, or 01 to 10, etc.).

Besides going to the Y decoder, the Y-register data is also transferred to the adder/comparator and is incremented and decremented as well. The Y-register may be set to any constant between zero and fifteen (by the TCY instruction). Sometimes Y-register data is loaded from the memory (TMY instruction) or the accumulator (TAY). The Y-register is extremely versatile and is used as a working register as well as an index for the RAM address. One of its other major functions is to select an R output address.

Instructions using bit masks from the CKI logic enable additional RAM addressing capabilities. Any bit in a RAM word addressed by X and Y registers can be set, reset, or tested.

2-6 RAM DATA I/O.

There are two modes of RAM access (read and write) during the instruction cycle.

(1) Data may be read out of the RAM for the purpose of addition, subtraction, or transfer to the other registers.

(2) Data is stored in the RAM via the write bus.

Two sources of information are written into the RAM; these sources are selected by the write multiplexer (shown on the right side of the function diagram, Figure 2-1.1). In one mode the multiplexer selects the accumulator information to be written into the RAM (uses STO microinstruction). The accumulator data is transferred to memory after data is read from the RAM but before the ALU results are stored into the accumulator. In the second mode, the constant and K-input logic is written into the RAM (by the CKM microinstruction). The constants from the ROM instruction bus are transferred to the RAM directly, and an optional data path from K1, K2, K4, and K8 exists although not selected in the standard instruction set. Four RAM bits are carried on the read bus to either the P-multiplexer or to the N-multiplexer and then to the adder/comparator.

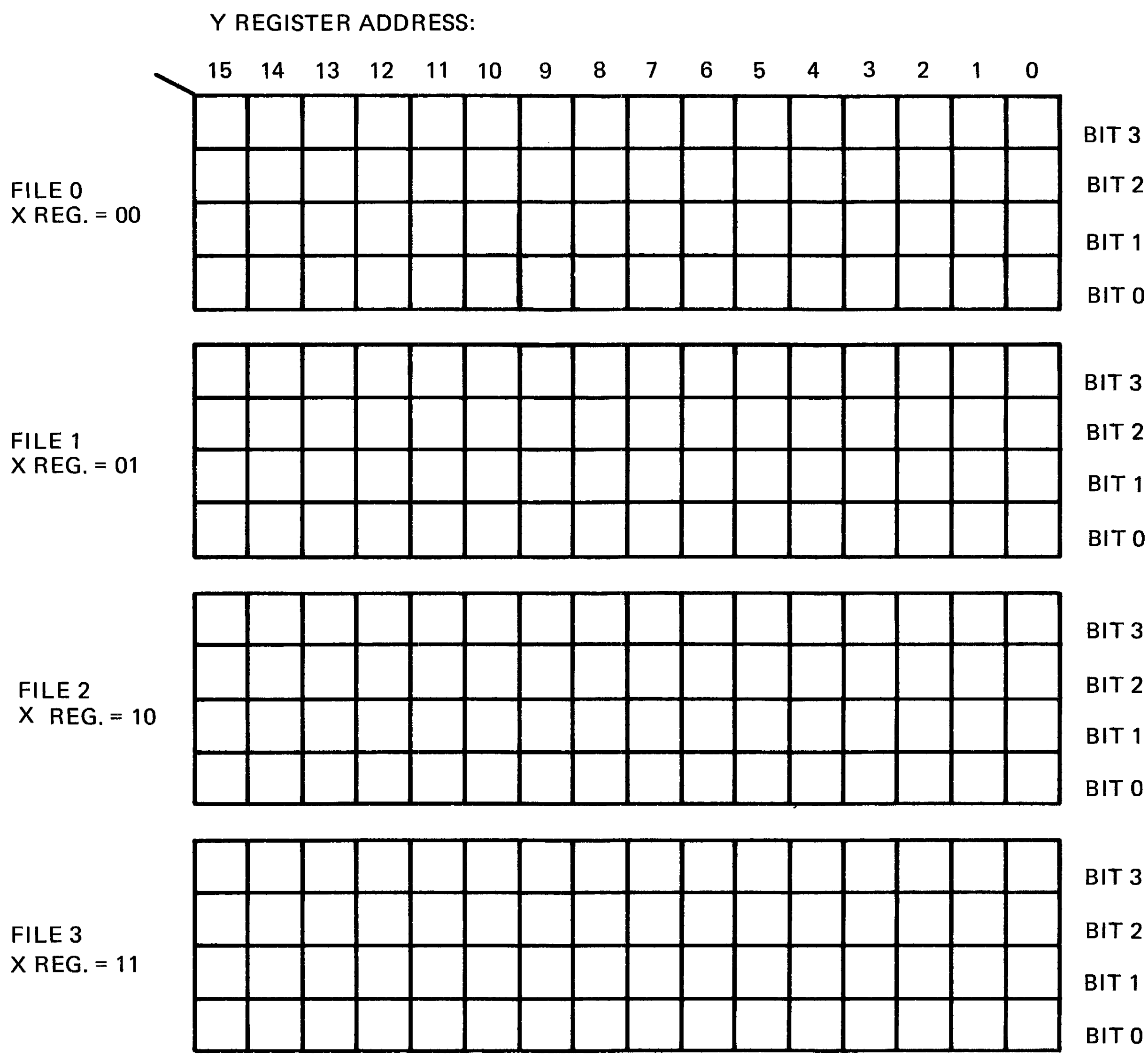


FIGURE 2-5.1 RAM FILE ORGANIZATION

2-7 CONSTANT AND K INPUT (CKI) LOGIC.

The purpose of the CKI logic is to select either the K-inputs or the four-bit constants from ROM (the C field of the instruction word) or a bit mask to go out to the CKI data bus.

The constant and K-input logic is used whenever microinstructions CKP, CKN, or CKM are selected by an instruction (see Section 2-17 for more details). The data going out on the CKI bus changes for predetermined instruction values, however, and this section details what the data is and the versatility of CKI microinstructions. Since the constant and K-input logic is not changeable, it is important to understand the four separate functions CKI controls before learning how CKI microinstructions are performed. Table 2-7.1 shows the binary decoded groupings of the instruction word and the particular output enabled by the CKI logic.

(1) First, for eight hexidecimal instruction values (08 to $0F_{16}$ as listed in Table 2-7.1), the K-inputs are active. That is, the constants from the ROM are shut off, and the four-bit external-input bus (center left of Figure 2-1.1) is made available to either the adder/comparator or the RAM. The instruction decoder determines how the available data is used.

(2) The second main function is to channel constant data from the instruction bus (from ROM) to the CKI bus output (instruction values 00 to 07 and 40_{16} to $7F_{16}$ as listed in Table 2-7.1). The CKI bus is available to the P adder input, the N adder input, or to the write multiplexer for the RAM as shown in Figure 2-1.1. The constant data from the ROM can be selected by 72 possible machine instruction values, although the standard instructions use only 68 of these.

(3) The constant logic is disabled (output at ZERO for values 20_{16} to $2F_{16}$).

(4) A bit mask is active. For example, the bit mask as used in the test bit instruction (TBIT1) determines if a bit from the RAM is a ONE by comparing it with ZERO. The bit mask has only one ZERO in the four-bit CKI output, as determined by the B field of the instruction word (see TBIT1 in Table 2-7.1). The B field is two-bits and points to the selected opening (ZERO) in the mask. Thus, if the least significant bit is to be tested, then the bit mask outputs the binary word 1110 to the CKI bus output. Then the CKI bus output goes into both sides of the adder/comparator, and the word at M(X,Y) is input simultaneously (logically ORed) with the CKI bus into the P side of the adder/comparator. The compare feature of the adder/comparator is activated and then the state of the tested bit transfers directly to status logic. The bit mask also selects RAM bits to be set or reset. For the set bit (SBIT) and reset bit (RBIT) instructions, the zero in the bit mask field (Table 2-7.1) also acts as a pointer to one of the four bits (identified by X and Y register contents) in a RAM character.

2-8 THE Y-REGISTER.

The Y-register has three purposes.

(1) The Y-register addresses the RAM in conjunction with the X-register for RAM I/O (see Figure 2-1.1).

TABLE 2-7.1 CONSTANT AND K-INPUT LOGIC TRUTH TABLE

Opcode (binary list) I (0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	Opcode (hex)		Mnemonic (Standard Instructions)	CKI Out	CKI Logic and Other Constant Operations	Comment
0	0	0	0	0	0	0	0	0	0	COMX			
0	0	0	0	0	0	0	1	0	1	A8AAC	Y		
0	0	0	0	0	0	1	0	0	2	YNEA			
0	0	0	0	0	0	1	1	0	3	TAM		I (7-4) →	I (7) = MSB
0	0	0	0	0	1	0	0	0	4	TAMZA		CKI BUS	I (4) = LSB
0	0	0	0	0	1	0	1	0	5	A10AAC	Y		
0	0	0	0	0	1	1	0	0	6	A6AAC	Y		
0	0	0	0	0	1	1	1	0	7	DAN	Y		
0	0	0	0	1	0	0	0	0	8	TKA	Y		
0	0	0	0	1	0	0	1	0	9	KNEZ	Y		
0	0	0	0	1	0	1	0	0	A	TDO			
0	0	0	0	1	0	1	1	0	B	CLO		$K_{1, 2, 4, 8}$ →	K_8 = MSB
0	0	0	0	1	1	0	0	0	C	RSTR		CKI BUS	
0	0	0	0	1	1	0	1	0	D	SETR			
0	0	0	0	1	1	1	0	0	E	IA			
0	0	0	0	1	1	1	1	0	F	RETN			
0	0	0	1	C				1	-	LDP		I (7-4)→ PB	NO EFFECT ON CKI, ONLY AFFECTS PB
0	0	1	0	0	0	0	0	2	0	TAMIY			
0	0	1	0	0	0	0	1	2	1	TMA			
0	0	1	0	0	0	1	0	2	2	TMY			
0	0	1	0	0	0	1	1	2	3	TYA			
0	0	1	0	0	1	0	0	2	4	TAY			
0	0	1	0	0	1	0	1	2	5	AMAAC			
0	0	1	0	0	1	1	0	2	6	MNEZ			
0	0	1	0	0	1	1	1	2	7	SAMAN			
0	0	1	0	1	0	0	0	2	8	IMAC		0 → CKI	
0	0	1	0	1	0	0	1	2	9	ALEM		BUS	
0	0	1	0	1	0	1	0	2	A	DMAN			
0	0	1	0	1	0	1	1	2	B	IYC			
0	0	1	0	1	1	0	0	2	C	DYN			
0	0	1	0	1	1	0	1	2	D	CPAIZ			
0	0	1	0	1	1	1	0	2	E	XMA			
0	0	1	0	1	1	1	1	2	F	CLA			
0	0	1	1	0	0	B		3	–	SBIT		BIT MASK →	B = 0 CKI = 1110
0	0	1	1	0	1	B		3	–	RBIT		CKI BUS	1 1101
0	0	1	1	1	0	B		3	–	TBIT 1	Y		2 1011 3 0111
0	0	1	1	1	1	B		3	–	LDX		I (7-6)→X	NO EFFECT ON CKI.
0	1	0	0	C				4	–	TCY	Y	I (7-4) →	I (7) = MSB
0	1	0	1	C				5	–	YNEC	Y	CKI BUS	I (4) = LSB
0	1	1	0	C				6	–	TCMIY	Y		
0	1	1	1	C				7	–	ALEC	Y		C → CKI BUS; C = 0 to 15
1	0	W						–	–	BR			NOT USED
1	1	W						–	–	CALL			

NOTE: I = Instruction (op code), C = Constant, W = Branch Address, Y = Yes (CKP, CKN, or CKM microinstruction is used).
PB = Page Buffer Register (ROM)

(2) The Y-register addresses the R-output register for setting and resetting individual latches. Whenever a particular R-output needs to be set, the constant bus inputs the R's address (0 through 12) to Y (TCY instruction), and then a set R-output (SETR) instruction is executed.

(3) The Y-register is used as a working register. As a working register, ROM words can be saved. For example, when a long delay time is desired, the Y-register is used as a counter.

The Y register may be set at any constant from 0 to 15, decremented (DYN), or incremented (IYC) in a single instruction cycle.

Note that in the functional block diagram (Figure 2-1.1), the Y-register has no inverted adder input. Thus, the Y-register cannot be subtracted from the accumulator or memory.

2-9 R-OUTPUT REGISTER.

The TMS1000 has two outputs:

- R outputs used for control
- O outputs used to transmit data (covered in paragraph 2-14)

The purpose of the R outputs is to control the following:

- External devices
- Display scans
- Input encoding
- Dedicated status logic outputs (such as overflow)

Each R output has a latch that stores a ONE or ZERO, and each latch may be set (ONE) or reset (ZERO) individually by the set R (SETR) or reset R (RSTR) instructions. The Y register points to which R output is set by these instructions.

The R-output can be strobed by the ROM program to scan a key matrix. Figure 2-9.1 represents the maximum key matrix possible without external logic. A simple short from an R line to a K-input can be detected by the ROM program and interpreted as any function or data entry. Expanding the matrix is possible by external logic such as using a 4-line to 16-line decoder.

2-10 ACCUMULATOR REGISTER.

The accumulator is a four-bit register that interacts with the adder, the RAM, and the output registers. The accumulator is the main working register for addition and subtraction. It is the only register which is inverted before its contents are sent to the adder for subtraction. Subtraction is accomplished by two's complement arithmetic. It is a storage register for inputs from the constant and K-input logic as well as the Y register.

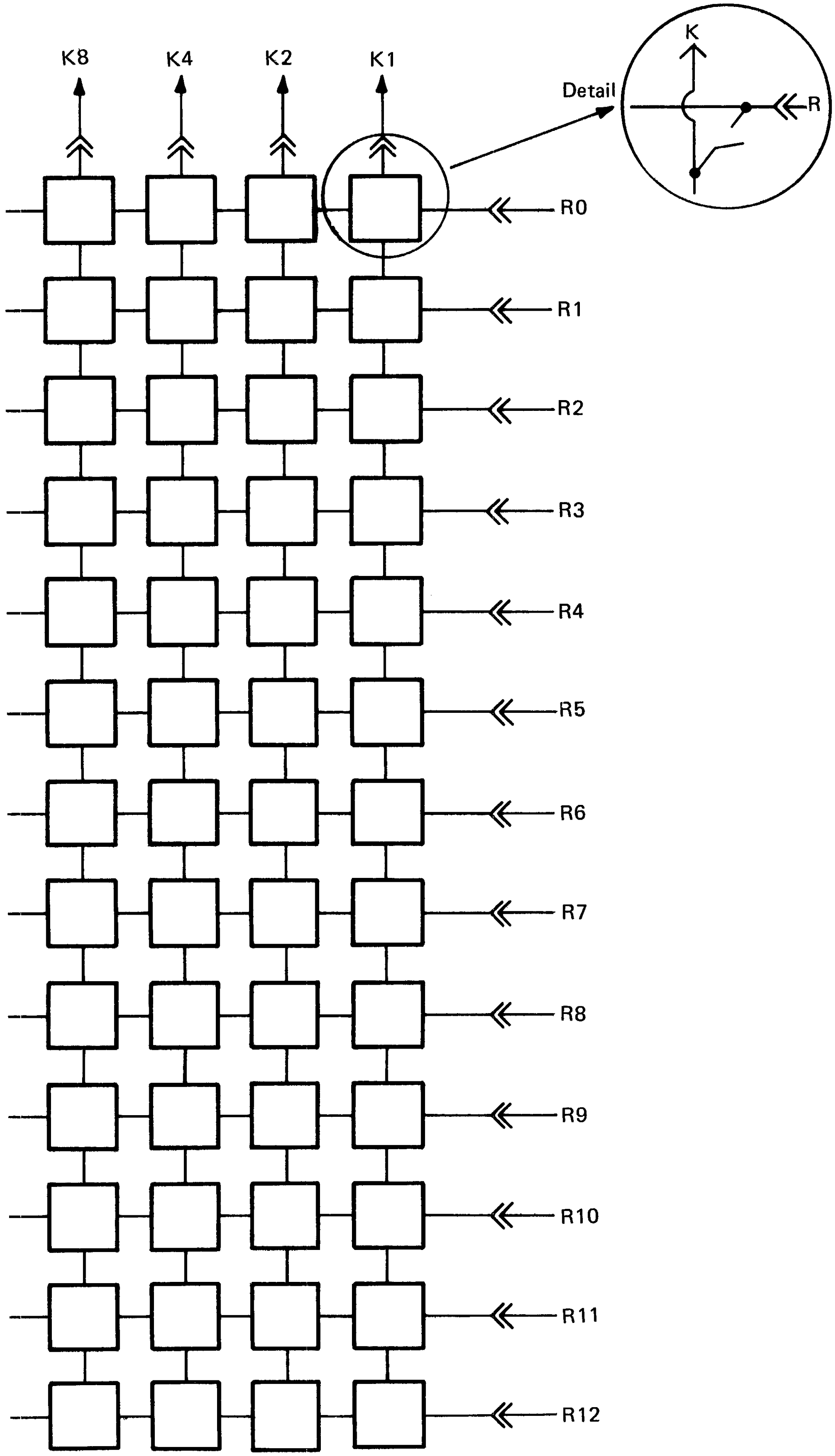


FIGURE 2-9.1 KEYBOARD MATRIX CONNECTIONS

Variable data from the K inputs are also stored via the accumulator into the RAM array. Therefore, any variable data input from the K inputs or from the adder output must pass through the accumulator to the RAM array for storage. Likewise, any data to the O outputs must come through the accumulator. Four accumulator register bits may be latched by the O-output register (where the status latch information is also latched) for decode by the O-output decoder.

2-11 ARITHMETIC LOGIC UNIT OPERATION.

Arithmetic and logic operations are performed by the Arithmetic Logic Unit (ALU) which is a four-bit adder/comparator and associated random logic (this is shown in the center right of Figure 2-1.1). The arithmetic unit performs logical comparison, add, subtract, and arithmetic comparison functions on its P and N inputs. The arithmetic logic unit and interconnects are shown in Figure 2-11.1. These two four-bit parallel inputs (P and N) may be added together or logically compared. The accumulator has a complemented output to the N selector for subtraction by two's complement arithmetic. The other N inputs are from the true output of the accumulator, the RAM, constants, and the K inputs. The P inputs come from the Y register, the RAM, the constants, and the K inputs.

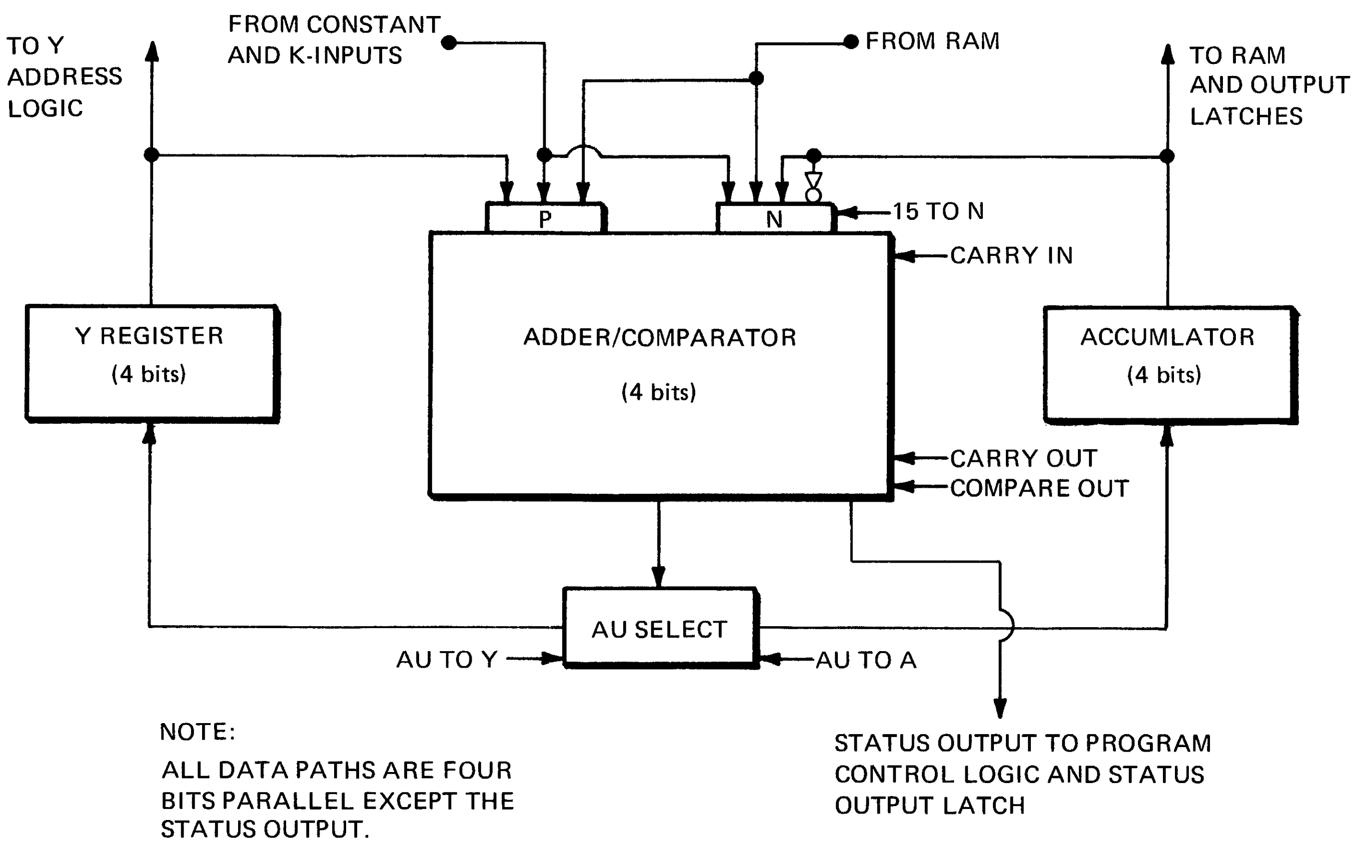


FIGURE 2-11.1 ARITHMETIC LOGIC UNIT

Addition and subtraction results (shown in Table 2-11.1) are stored in either the Y register or the accumulator. Either an arithmetic function or a logical comparison may generate an output to status logic. If either logical or arithmetic comparison functions are used, only the status logic bit affects the program control, and neither the Y register's nor the accumulator register's contents are affected. If a branch or call is attempted when the status logic bit is a logic ONE (which is the normal state), the conditional branch or call is executed.

If an instruction calls for a carry output to status and the carry does not occur, then status will go to a ZERO level for one instruction cycle. Likewise, if an instruction calls for the logical comparison function and the bits compared are all equal (EXORED), then status will go to a ZERO level for one instruction cycle. If status is a logic ZERO, then branches and calls are not performed.

The arithmetic unit has a carry-in feature in which a ONE is added to the sum of the P and N adder inputs.

2-11.1 N-INPUT TO ADDER. There are five possible microinstruction selections for adder "N" input control as shown in Figure 2-1.1. The first comes from the RAM. The second input is from the constant and K-input logic. Also, the accumulator and $\overline{\text{accumulator}}$ may be selected through the

TABLE 2-11.1 ADDER OUTPUTS

M-A′ OR M+A

A′	A \ M	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	CARRY = 0
F	1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	CARRY = 1
E	2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	
D	3	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	
C	4	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	
B	5	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	
A	6	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	
9	7	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	
8	8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	
7	9	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	
6	A	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	
5	B	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	
4	C	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	
3	D	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	
2	E	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	
1	F	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	

NOTE:

A′ IS THE TWO'S COMPLEMENT OF A (WHICH IS $\bar{A}$+1)

N-multiplexer. A fifth function selects fifteen (binary 1111) as an input to the adder. If more than one input is selected in the same instruction cycle, then those inputs are logically ORed through the N multiplexer.

2-11.2 P-INPUT TO ADDER. P selections may come from the Y register, constant and K-input logic, or from the RAM array. If a combination of these inputs is designated, then they are logically ORed.

2-11.3 ADDER/COMPARATOR OUTPUT. The adder/comparator output (see Table 2-11.1) is selected by ROM control to go to the Y-register, the accumulator register, or neither. The Y register is selected by the microinstruction AUTY. The accumulator register is selected by the microinstruction AUTA. Addition or subtraction instructions select either the Y register or the accumulator as a destination for results. If neither is selected while the adder is performing an operation, then this instruction is one of the test instructions. In the test instructions, the adder is used to generate a status output to control the program, but the results are not stored in either the Y register or accumulator.

2-12 STATUS LOGIC.

There are 18 instructions that affect status logic, either setting it (to ONE) or resetting it (to ZERO). In turn, the status logic will permit the successful execution of a branch or call instruction (if status logic = ONE) or prevent successful execution of these instructions (if reset to ZERO). Status logic will remain at a ZERO level only for the following instruction cycle and then automatically be set to the normal ONE state (unless reset to ZERO by the next instruction).

There are two microinstructions (NE and C8) that are used by instructions affecting status. If the microinstruction C8 is used and a carry occurs in the addition of two four-bit words, the carry goes from the MSB sum to status, setting status logic to a ONE. If no carry occurs, status logic is ZERO. In a logic compare instruction (using microinstruction NE), status logic is set to ONE if the four-bit words at the N and P adder/comparator inputs are not equal; conversely, status logic is ZERO if the inputs are equal.

2-13 STATUS LATCH.

The status latch buffers the status-logic bit to the O-output register for decode by the O-output PLA. Status-logic output is selectively loaded into the status latch by special microinstruction STSL (used in a logical-compare test instruction that causes the status logic to output a ONE or ZERO). For example, if the test instruction YNEA (in the standard instruction set) causes status to be a ONE (if Y register is not equal to A), then the ONE writes into the status latch. If a ZERO is output by that instruction from status logic, then the ZERO writes into the status latch.

The status latch transfers to the O register with the accumulator bits when TDO, transfer data out, is executed.

2-14 O-OUTPUT REGISTER.

Paragraph 2-13 describes how the status output is stored in the status latch. The status latch and the accumulator data are loaded into the O-output register (bottom right of Figure 2-1.1) by a fixed instruction from the ROM (TDO), when the programmer decides to change output data. A separate instruction clears the O-output register. This instruction (CLO) causes all five output register bits to be reset to ZERO. The five bits from the O register are converted to a parallel eight-bit code by the O PLA.

NOTE
The O output register transfers accumulator and status latch data, the R output register (paragraph 2-9) transfers control data.

2-15 PROGRAMMABLE LOGIC ARRAY (PLA).

There are two PLA's in the TMS1000 series:

- The O-output PLA (paragraph 2-16)
- The instruction decoder PLA (2-17.1)

For those who may need a review or are unfamiliar with PLA's, the following discusses the PLA concept.

A matrix of gates first decodes a number of input bits into a set of output lines (also called "terms"). Each term can select a combination of output lines from a second matrix of gates (see Figure 2-15.1).

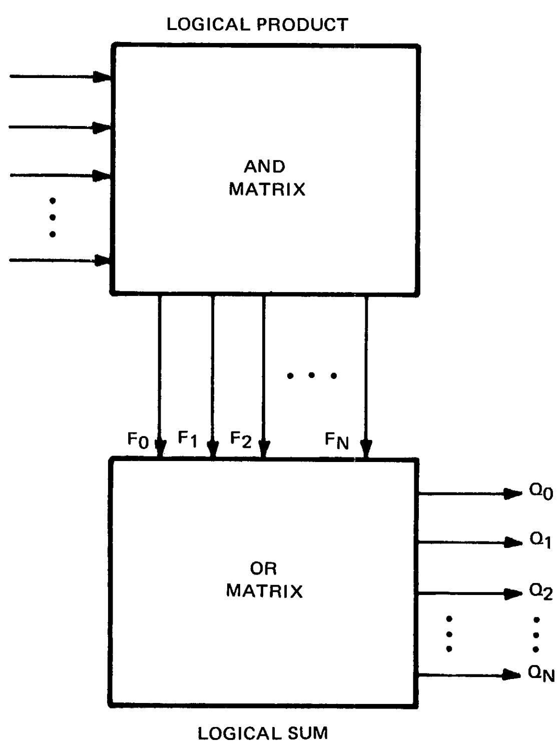


FIGURE 2-15.1 PLA BLOCK DIAGRAM

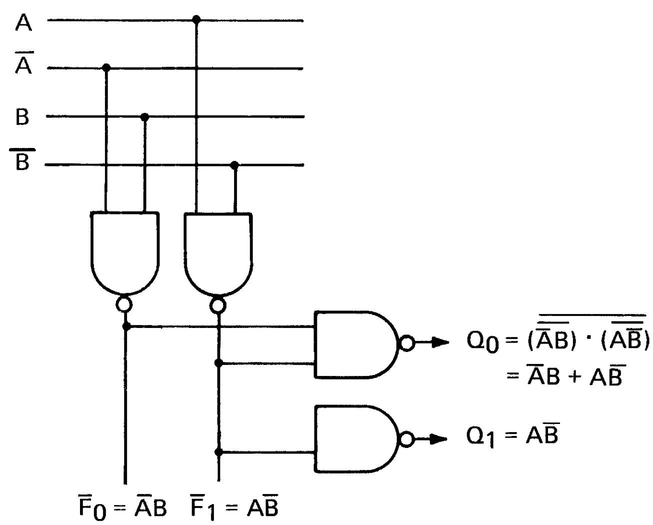


FIGURE 2-15.2 STANDARD LOGIC PLA CIRCUIT SCHEMATIC

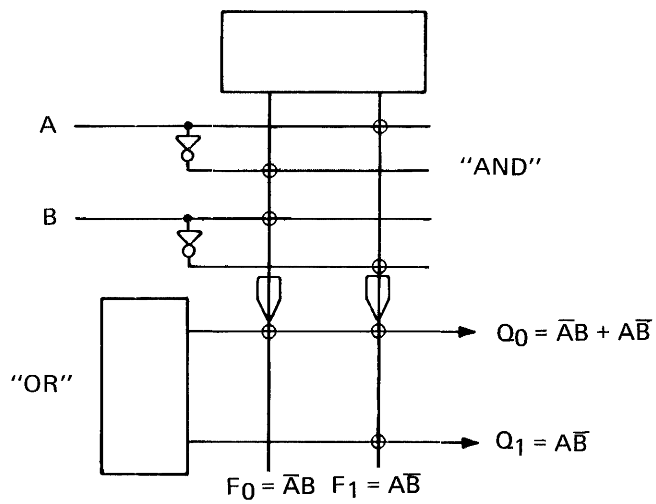


FIGURE 2-15.3 ARRAY LOGIC EQUIVALENT SCHEMATIC

Both matrices are implemented by programmable input NAND gates (Figure 2-15.2). Since we are concerned only with the input-to-output code conversion, positive logic AND and OR functions are used herein.

Figure 2-15.2 shows two AND matrix terms, F_0 and F_1, which are encoding two output OR matrix terms, Q_0 and Q_1. The simplified method of presenting the same circuit is shown in Figure 2-15.3. Each circle in the diagram represents a MOSFET which selects a gate input to a matrix term.

User programming of these PLA's requires inputs to the TMS1000 simulator for O-output PLA programming and to the assembler and simulator for instruction PLA programming. User inputs are covered in detail in the TMS1000 Software Users Guide.

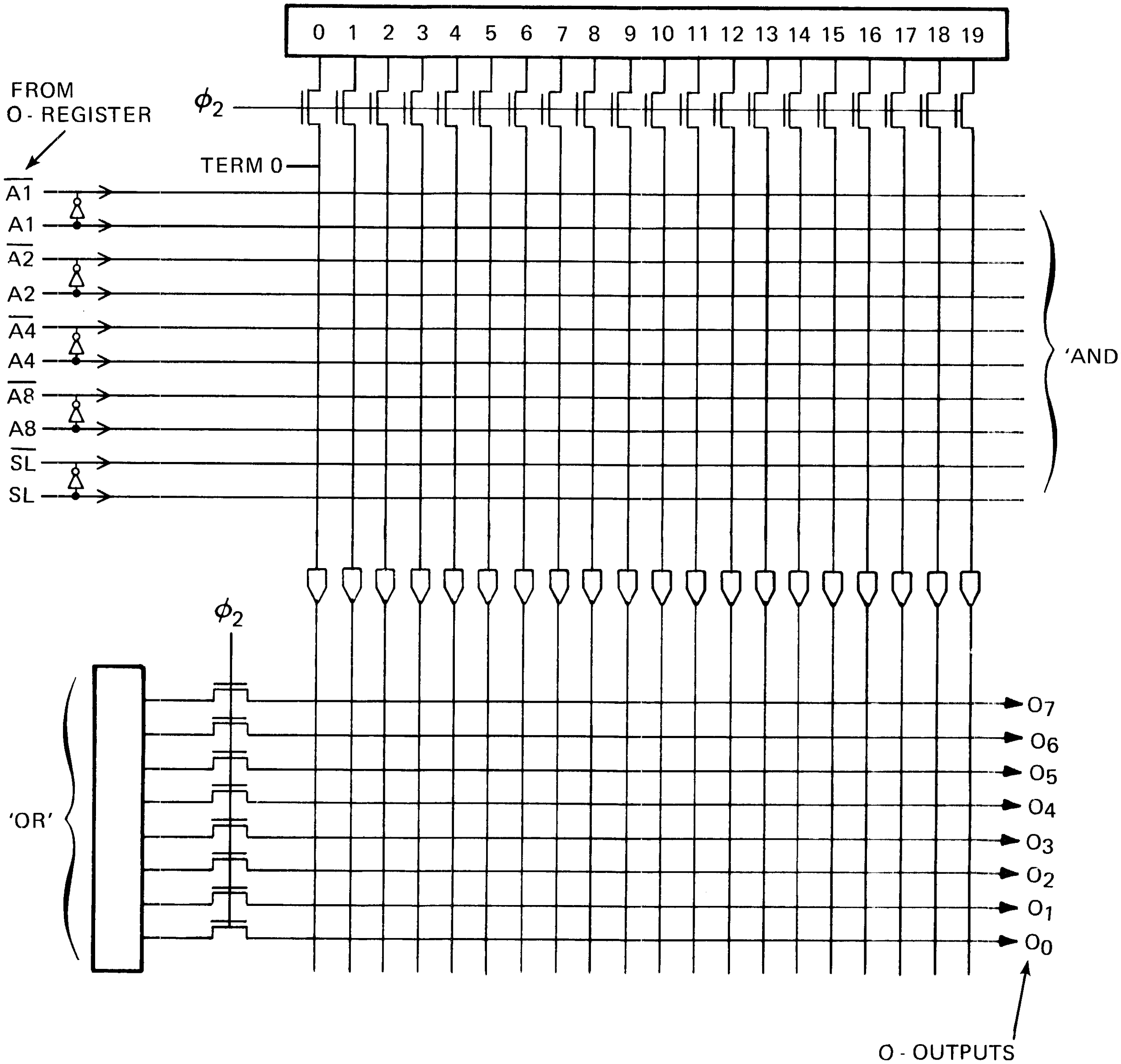


FIGURE 2-16.1 OUTPUT PLA

2-16 O-OUTPUT PLA, CODE CONVERTER.

The O-output PLA determines the parallel output definition for each TMS1000 series program. Thus, a user understanding the capabilities can define an efficient output organization before designing an algorithm. The organization of the outputs is a necessary starting point for new system designs.

The O-output register sends five bits to the O-output PLA (bottom of Figure 2-1.1). Figure 2-16.1 shows the five corresponding O-register bits from accumulator and status latch) going into the AND matrix in true and complemented form. The AND matrix has 20 terms available for decoding a prescribed pattern of inputs to the OR matrix. The pattern is stored in the matrix by placing MOS transistors (gates) to select inputs and not placing a gate where an input is not desired (see section 2-15).

Each AND matrix term may decode a subset of the following Boolean equation:

$$F_N = (A1 \cdot \overline{A1}) \cdot (A2 \cdot \overline{A2}) \cdot (A4 \cdot \overline{A4}) \cdot (A8 \cdot \overline{A8}) \cdot (SL \cdot \overline{SL})$$

Either the true, or the complement (not both), or neither (don't care) of the two inputs enclosed in parentheses can be selected. The AND matrix may decode up to 20 of these Boolean equations.

Each OR matrix line determines the O-output pattern for each AND term used. If an AND term is true, the output selection (represented by a circle) is a subset of the following expression:

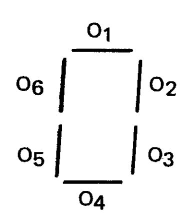
$$O \text{ output} = O_0 + O_1 + O_2 + O_3 + O_4 + O_5 + O_6 + O_7$$

If any two or more AND term equations are satisfied, then their ORed output functions are logically ORed together.

The example coding shown in Figure 2-16.2 shows an output classified into seven-segment information and binary information. If the status latch bit is ZERO, then the PLA sends binary information out. If the status latch bit is ONE, then the PLA encodes seven-segment display information. Note that there are 20 input terms to the OR matrix; four terms encode the binary value of the accumulator bits, 16 terms encode the characters zero to F.

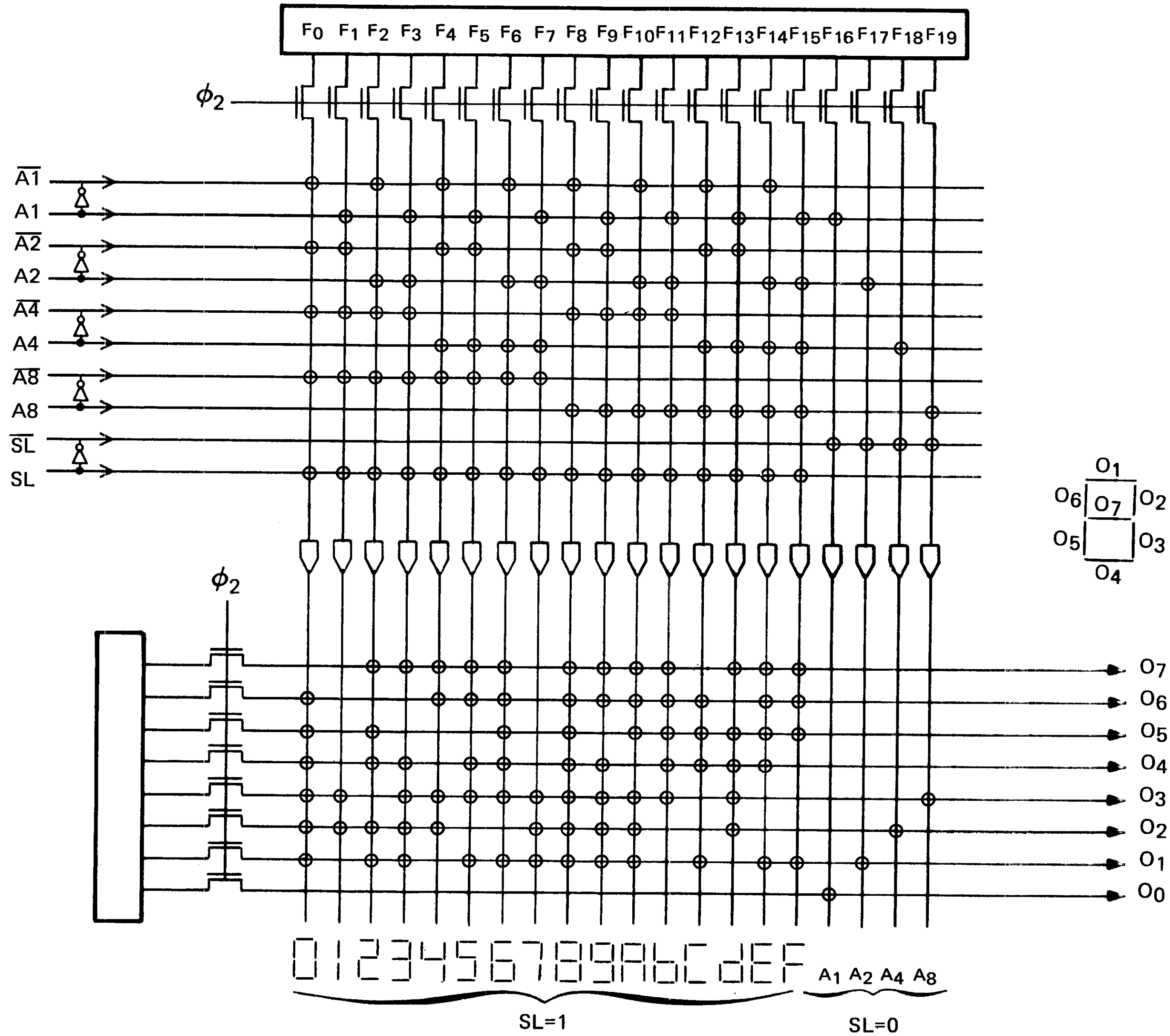
The TDO instruction latches the status latch and the accumulator bits in the O register. In the case of term zero (F_0), a ONE from the status latch and zero from the accumulator encodes the seven-segment character for zero:

$$O \text{ output} = O_1 + O_2 + O_3 + O_4 + O_5 + O_6$$



NOTE

Positive logic is used on all outputs. A true output drives toward V_{SS}. Definition for the O-PLA to the simulators is covered in the TMS1000 Series Software User's Guide.



NOTE: IF THE CLO INSTRUCTION IS USED,
THE DECODER OUTPUTS A BLANK (O_0 TO O_7 = ZERO)

FIGURE 2-16.2 TYPICAL CODING EXAMPLE OF O-OUTPUT PLA

2-17 INSTRUCTION DECODERS.

Two logic blocks decode the eight-bit instructions into the various microinstructions.

- Fixed instruction decoder
- Programmable instruction PLA

The fixed instruction decoder cannot be modified and enables 12 fixed controls affecting ROM addressing, RAM X register, output control, set bit and reset bit instructions. Every program must use these instructions with their corresponding fixed microinstructions. Refer to Table 2-17.1, and notice that each "fixed" instruction has a corresponding fixed microinstruction described by an identical mnemonic.

The remaining 31 basic instructions in the standard set (43 basic instructions - 12 fixed basic instructions = 31 programmable instructions) have their operations determined by combining one or more microinstructions as determined by the instruction PLA. The combinations used in the standard instructions are listed in Table 2-17.1.

The programmable instructions are defined for the user to the assembler and simulator programs by default definition when the standard instructions are used. When one or more instructions are redefined, the user specifies the entire set of instruction mnemonics to the assembler, and new PLA implementation is defined to the simulator. Changing these definitions is covered in detail in the TMS1000 Series Software User's Guide.

2-17.1 THE PROGRAMMABLE MICROINSTRUCTIONS. In the preceding sections of this document, specific controls for each logic block are explained. These controls are enabled by the microinstructions coming out of the OR matrix of the Instruction Programmable Logic Array. Figure 2-17.1 summarizes the controls by showing an arrow pointing to the logic block or the particular data path affected. Table 2-17.2 defines operation of the programmable microinstructions, and the logic block controlled.

In one instruction cycle the sequence of microinstruction execution is in the following order:

(1) Read RAM, select the inputs to the adder/comparator.
Microinstructions: CIN, MTP, MTN, CKP, CKN, YTP, ATN, 15TN, NATN, C8, NE

(2) Write accumulator contents or CKI bus information into the RAM.
Microinstructions: CKM, STO

(3) Add or compare, then store results into the Y register, accumulator, status logic, or status latch.
Microinstructions: AUTY, AUTA, STSL

Thus the MTP (RAM memory contents to P-adder input) microinstruction is executed before STO (store accumulator data into RAM). The adder can perform one operation per instruction cycle. If

TABLE 2-17.1 MICROINSTRUCTION INDEX

Mnemonic	Opcode								Microinstructions Fixed	Microinstructions Programmable	Reference Paragraph
ALEC	0	1	1	1	C					CKP, NATN, CIN, C8	4-5.2
ALEM	0	0	1	0	1	0	0	1		MTP, NATN, CIN, C8	4-5.1
AMAAC	0	0	1	0	0	1	0	1		MTP, ATN, C8, AUTA	4-4.1
A6AAC	0	0	0	0	0	1	1	0		CKP,ATN,C8,AUTA	4-4.11
A8AAC	0	0	0	0	0	0	0	1		CKP, ATN, C8, AUTA	4-4.9
A10AAC	0	0	0	0	0	1	0	1		CKP, ATN, C8, AUTA	4-4.10
BR	1	0			W				BR		4-12.1
CALL	1	1			W				CALL		4-12.2
CLA	0	0	1	0	1	1	1	1		AUTA	4-2.3
CLO	0	0	0	0	1	0	1	1	CLO		4-10.4
COMX	0	0	0	0	0	0	0	0	COMX		4-11.2
CPAIZ	0	0	1	0	1	1	0	1		NATN,CIN,C8,AUTA	4-4.12
DAN	0	0	0	0	0	1	1	1		CKP, ATN, CIN, C8, AUTA	4-4.7
DMAN*	0	0	1	0	1	0	1	0		MTP, 15TN, C8, AUTA	4-4.4
DYN	0	0	1	0	1	1	0	0		YTP, 15TN, C8, AUTY	4-4.8
IA	0	0	0	0	1	1	1	0		ATN, CIN, AUTA	4-4.5
IMAC*	0	0	1	0	1	0	0	0		MTP, CIN, C8, AUTA	4-4.3
IYC	0	0	1	0	1	0	1	1		YTP, CIN, C8, AUTY	4-4.6
KNEZ	0	0	0	0	1	0	0	1		CKP, NE	4-9.1
LDP	0	0	0	1	C				LDP		4-12.4
LDX	0	0	1	1	1	1	B		LDX		4-11.1
MNEZ	0	0	1	0	0	1	1	0		MTP, NE	4-6.1
RBIT	0	0	1	1	0	1	B		RBIT		4-7.2
RETN	0	0	0	0	1	1	1	1	RETN		4-12.3
RSTR	0	0	0	0	1	1	0	0	RSTR		4-10.2
SAMAN	0	0	1	0	0	1	1	1		MTP, NATN, CIN, C8, AUTA	4-4.2
SBIT	0	0	1	1	0	0	B		SBIT		4-7.1
SETR	0	0	0	0	1	1	0	1	SETR		4-10.1
TAM	0	0	0	0	0	0	1	1		STO	4-3.1
TAMIY	0	0	1	0	0	0	0	0		STO, YTP, CIN, AUTY	4-3.2
TAMZA	0	0	0	0	0	1	0	0		STO, AUTA	4-3.3
TAY	0	0	1	0	0	1	0	0		ATN, AUTY	4-2.1
TBIT 1	0	0	1	1	1	0	B			CKP, CKN, MTP, NE	4-7.3
TCY	0	1	0	0	C					CKP, AUTY	4-8.1
TCMIY	0	1	1	0	C					CKM, YTP, CIN, AUTY	4-8.2
TDO	0	0	0	0	1	0	1	0	TDO		4-10.3
TKA	0	0	0	0	1	0	0	0		CKP, AUTA	4-9.2
TMA	0	0	1	0	0	0	0	1		MTP, AUTA	4-3.5
TMY	0	0	1	0	0	0	1	0		MTP, AUTY	4-3.4
TYA	0	0	1	0	0	0	1	1		YTP, AUTA	4-2.2
XMA	0	0	1	0	1	1	1	0		MTP, STO, AUTA	4-3.6
YNEA	0	0	0	0	0	0	1	0		YTP, ATN, NE, STSL	4-6.2
YNEC	0	1	0	1	C					YTP, CKN, NE	4-6.3

*Execution of the DMAN or IMAC instruction does not change (increment or decrement) the content of the addressed memory cell.

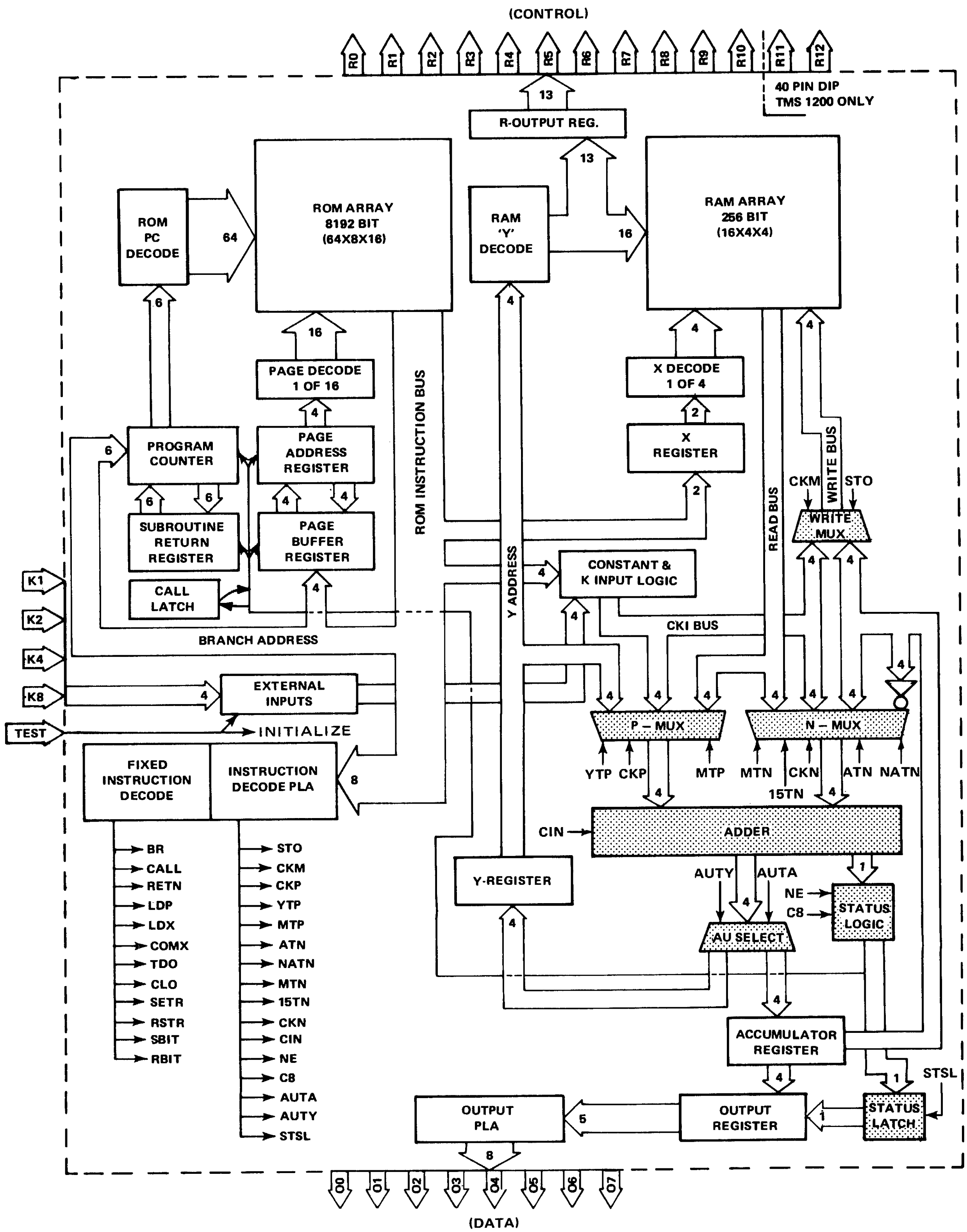


FIGURE 2-17.1 TMS1000 FUNCTIONAL BLOCKS AND PROGRAMMABLE MICROINSTRUCTIONS

TABLE 2-17.2 TMS1000 SERIES PROGRAMMABLE MICROINSTRUCTIONS

Execution Sequence	Mnemonic	Logic Affected	Function
1	CKP	P-MUX	CKI to P adder input.
	YTP	P-MUX	Y-Reg to P adder input.
	MTP	P-MUX	Memory (X,Y) to P adder input.
1	ATN	N-MUX	Accumulator to N adder input.
	NATN	N-MUX	$\overline{\text{Accumulator}}$ to N adder input.
	MTN	N-MUX	Memory (X,Y) to N adder input.
	15TN	N-MUX	F_{16} to N adder input.
	CKN	N-MUX	CKI to N adder input.
1	CIN	Adder	One is added to sum of P plus N inputs (P+N+1).
	NE	Adder/Status	Adder compares P and N inputs. If they are identical, status is set to zero.
	C8	Adder/Status	Carry is sent to status (MSB only).
2	STO	Write MUX	Accumulator data to memory.
	CKM	Write MUX	CKI to memory.
3	AUTA	AU Select	Adder result stored into accumulator.
	AUTY	AU Select	Adder result stored into Y-Reg.
	STSL	Status Latch	Status is stored into status latch.

two input buses are selected for the same side of the adder, the inputs are logically ORed together (e.g., TBIT1, section 4-7.3).

The programmable microinstructions are an aid to learning how instructions work. For example, the IA instruction (increment accumulator) enables three microinstructions; ATN, CIN, and AUTA:

(1) ATN transfers the accumulator data to the N-adder input (P=0)

(2) CIN causes one to be added to the P and N-adder inputs.

(3) AUTA causes the result of the addition to be stored in the accumulator.

Knowing the hardware and how TI combined the microinstructions explains all 31 programmable instructions. For example, the YNEC instruction activates three microinstructions.

(1) CKN causes the constant from ROM (immediate operand) to go into the N-input.

(2) YTP enables Y to the P-input

(3) NE sends the comparison to status

Therefore, if Y is logically compared to a constant operand and is *not equal* to the CKI data, status equals ONE.

Figure 2-17.2 illustrates the PLA implementation designed by TI for the standard instruction set. The 31 instructions are translated by 30 PLA terms into a combination of the 16 microinstructions possible (the A8AAC and the A10AAC are combined on a single PLA line).

The instruction PLA can be reprogrammed in cases where timing or other requirements dictate an instruction redefinition. Microprogramming this PLA should be considered only when the standard definition is insufficient to accomplish the program objectives. Contact the MOS division in Houston, Texas, to obtain help in such cases.

2-17.2 FIXED INSTRUCTION DECODER. This decoder is a block of logic that cannot be changed and is needed to decode the twelve basic instructions that every program must use (i.e., the machine code of these fixed instructions cannot be changed). Figure 2-17.3 presents the functional block diagram again with arrows showing which logic blocks are affected by the fixed microinstructions. The mnemonics are the same as the ROM instructions since the standard instruction set uses a one-to-one correspondence between the fixed instructions and their microinstructions.

The 12 instructions, decoded by the fixed instruction decoder, can be modified by adding other programmable microinstructions to those that are already enabled. These additional microinstructions can be coded into the programmable instruction decoder. For example, the set R-output command could be coded to also decrement the Y register. The SETR instruction code is OC_{16}, and if decoded by both the fixed instruction decoder and the programmable instruction decoder, this command can perform two operations.

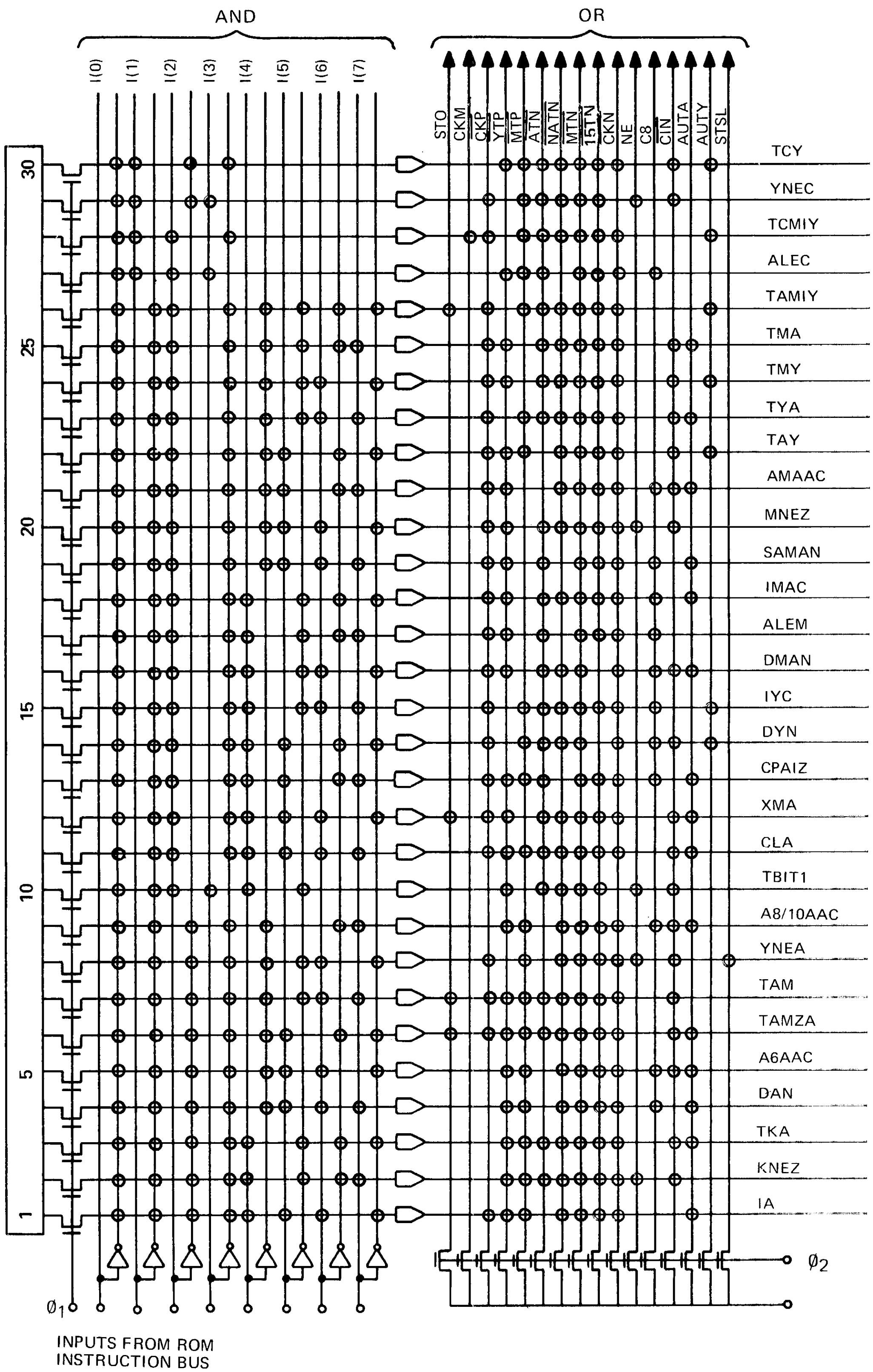


FIGURE 2-17.2 TMS1000/1200 STANDARD INSTRUCTION DECODE PLA

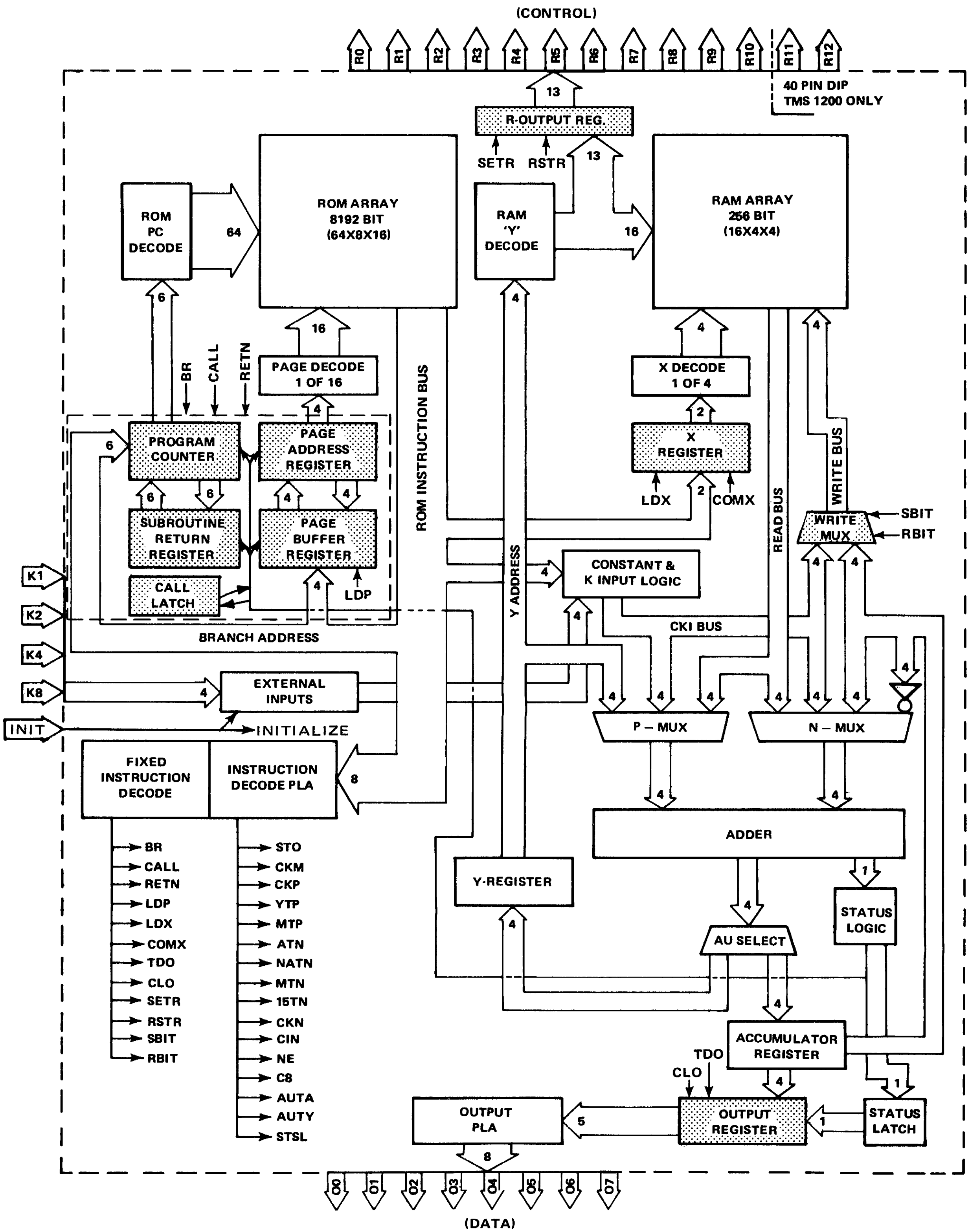


FIGURE 2-17.3 TMS1000/1200 FUNCTIONAL BLOCKS & FIXED MICROINSTRUCTIONS

Note that there are up to 30 PLA terms available, all of which are used up by the standard instruction set, so additional decoding for fixed instructions will displace some programmable instruction or must be combined on the same PLA term.

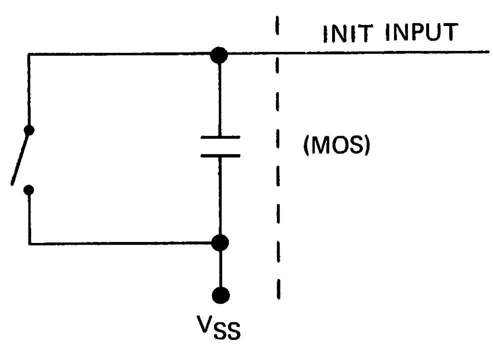
2-18 EXTERNAL INPUTS.

External-inputs logic buffers the K inputs as shown in Figure 2-1.1. Each input has a pull-down resistor (to V_{DD}) equal to 50 kilohms. V_{DD} represents a ZERO input; a V_{SS} level signifies a ONE.

2-19 INITIALIZING THE TMS 1000 SERIES DEVICES.

The INIT input pin initializes the hardware and resets the page address register, program counter, and the O- and R-output registers. The external-inputs logic forces binary 1111 into the page address register and the program counter is reset to zero when a minimum of V_{SS} −1 volt is applied to the INIT input for at least six consecutive instruction cycles if K1, K2, K4, K8, and R10 equal ZERO (V_{DD}). In addition, the page buffer register is set to binary 1111 and the O-output register, R-output register, and the call latch are reset to all ZEROes.

The INIT pin is used in some applications as a hardware reset since the aforementioned procedure sets the program counter and page address register addresses to the initial power-up address if all K inputs and R10 are held at a low level. The following diagram shows the circuitry to accomplish hardware clear with all K-inputs at a logic ZERO. A capacitor reduces bounce noise from the key contacts since INIT must be at a high level for at least six instruction cycles after key bounce has ceased.



2-20 POWER-UP LATCH.

The TMS1000 contains a power-up latch (not shown in Figure 2-1.1) which presets the PC to zero and the PA and PB to F_{16}, presets the call latch to ZERO, resets the O-output register to all ZEROes, and resets the R-output register to all ZEROes.

If the system power supply settles slowly in systems that require frequent power-up and power-down cycles, the circuit connected to the INIT input in Figure 2-19.1 will help ensure a proper power-on procedure, since systems require executing a special block of code for clearing all the RAM characters, clearing the accumulator, resetting external devices, etc. A capacitor connected to the INIT pin causes the V_{SS} level to charge slowly through a clocked load device to V_{DD}, internal to the TMS 1000. The diode discharges the capacitor when the system is turned off. The capacitance required varies from system to system, but the capacitance should be large enough to hold the V_{SS} –1 volt for six instruction cycles longer than the rise time of the power supply as a minimum. All K-inputs must be at V_{DD} to accomplish all the effects of the power-on latch, and output pin R10 must not be pulled high.

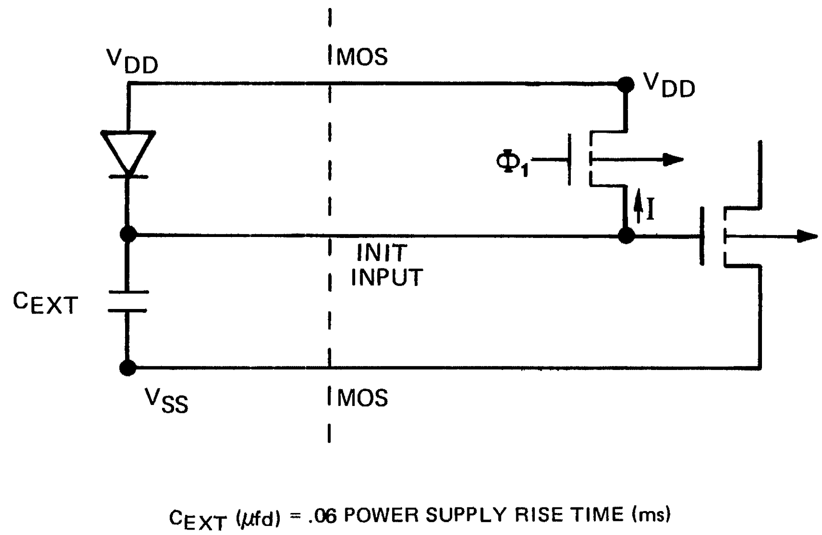


FIGURE 2-19.1 TYPICAL POWER ON CIRCUIT

SECTION III

INSTRUCTION CROSS REFERENCE TABLES

This section is a quick-reference introduction to the 43 TMS1000 series instructions that are defined in Section 4. These tables facilitate instruction comprehension and are arranged in the following order:

- Table 3-1 lists the instructions by function.
- Table 3-2 lists the instructions alphabetically.
- Table 3-3 lists the microinstructions for each instruction.
- Table 3-4 lists the instructions by binary machine code.
- Figure 3-1 is the instruction code map in hexadecimal.

NOTE

These tables use abbreviations and symbols explained in paragraphs 1-4.1 and 1-4.2.

TABLE 3-1 TMS1000/1200 STANDARD INSTRUCTION SET

Function	Mnemonic	Status* Effect C8	Status* Effect NE	Description	Explained in Para.
Register to Register	TAY			Transfer accumulator to Y register.	4-2.1
	TYA			Transfer Y register to accumulator.	4-2.2
	CLA			Clear accumulator.	4-2.3
Transfer Register to Memory	TAM			Transfer accumulator to memory.	4-3.1
	TAMIY			Transfer accumulator to memory and increment Y register.	4-3.2
	TAMZA			Transfer accumulator to memory and zero accumulator.	4-3.3
Memory to Register	TMY			Transfer memory to Y register.	4-3.4
	TMA			Transfer memory to accumulator.	4-3.5
	XMA			Exchange memory and accumulator.	4-3.6
Arithmetic	AMAAC	Y		Add memory to accumulator, results to accumulator. If carry, one to status.	4-4.1
	SAMAN	Y		Subtract accumulator from memory, results to accumulator. If no borrow, one to status.	4-4.2
	IMAC**	Y		Increment memory and load into accumulator. If carry, one to status.	4-4.3
	DMAN**	Y		Decrement memory and load into accumulator. If no borrow, one to status.	4-4.4
	IA			Increment accumulator, no status effect.	4-4.5
	IYC	Y		Increment Y register. If carry, one to status.	4-4.6
	DAN	Y		Decrement accumulator. If no borrow, one to status.	4-4.7
	DYN	Y		Decrement Y register. If no borrow, one to status.	4-4.8
	A8AAC	Y		Add 8 to accumulator, results to accumulator. If carry, one to status.	4-4.9
	A10AAC	Y		Add 10 to accumulator, results to accumulator. If carry, one to status.	4-4.10
	A6AAC	Y		Add 6 to accumulator, results to accumulator. If carry, one to status.	4-4.11
	CPAIZ	Y		Complement accumulator and increment. If then zero, one to status.	4-4.12
Arithmetic Compare	ALEM	Y		If accumulator less than or equal to memory, one to status.	4-5.1
	ALEC	Y		If accumulator less than or equal to a constant, one to status	4-5.2
Logical Compare	MNEZ		Y	If memory not equal to zero, one to status.	4-6.1
	YNEA		Y	If Y register not equal to accumulator, one to **status and status latch.**	4-6.2
	YNEC		Y	If Y register not equal to a constant, one to status	4-6.3
Bits in Memory	SBIT			Set memory bit.	4-7.1
	RBIT			Reset memory bit.	4-7.2
	TBIT1		Y	Test memory bit. If equal to one, one to status.	4-7.3
Constants	TCY			Transfer constant to Y register.	4-8.1
	TCMIY			Transfer constant to memory and increment Y.	4-8.2
Input	KNEZ		Y	If K inputs not equal to zero, one to status.	4-9.1
	TKA			Transfer K inputs to accumulator.	4-9.2
Output	SETR			Set R output addressed by Y.	4-10.1
	RSTR			Reset R output addressed by Y.	4-10.2
	TDO			Transfer data from accumulator and status latch to O-outputs.	4-10.3
	CLO			Clear O-output register.	4-10.4
RAM X Addressing	LDX			Load X with a constant.	4-11.1
	COMX			Complement X.	4-11.2
ROM Addressing	BR			Branch on status = one.	4-12.1
	CALL			Call subroutine on status = one.	4-12.2
	RETN			Return from subroutine.	4-12.3
	LDP			Load page buffer with constant.	4-12.4

*NOTE A:

C8 (microinstruction C8 is used) – Y (Yes) means that if there is a carry out of the MSB, status output goes to the ONE state. If no carry is generated, status output goes to the ZERO state.

NE (microinstruction NE is used) – Y (Yes) means that if the bits compared are not equal, status output goes to the ONE state. If the bits are equal, status output goes to the ZERO state.

A ZERO in status remains through the next instruction cycle only. If the next instruction is a branch or call and status is a ZERO, then the branch or call is not executed.

**NOTE B:

Execution of the DMAN or IMAC instruction does not change (increment or decrement) the content of the addressed memory cell.

TABLE 3-2. ALPHABETICAL MNEMONIC REFERENCE

Mnemonic	Opcode (binary)	Opcode (hex)	Action	Status C8	Status NE	Reference Paragraph
ALEC	0 1 1 1 C	7 –	A ≤ C	Y		4-5.2
ALEM	0 0 1 0 1 0 0 1	2 9	A ≤ M(X,Y)	Y		4-5.1
AMAAC	0 0 1 0 0 1 0 1	2 5	M(X,Y) + A → A	Y		4-4.1
A6AAC	0 0 0 0 0 1 1 0	0 6	A + 6 → A	Y		4-4.11
A8AAC	0 0 0 0 0 0 0 1	0 1	A + 8 → A	Y		4-4.9
A10AAC	0 0 0 0 0 1 0 1	0 5	A + 10 → A	Y		4-4.10
BR	1 0 W	–	S=1, CL=0: I(W) → PC, PB → PA S = 0: 1 → S, PC + 1 → PC S = 1 CL = 1: I(W) → PC			4-12.1
CALL	1 1 W	–	S=1, CL=0: PC + 1 → SR, PB ↔ PA, 1 → CL, I(W) → PC S = 0: 1 → S, PC + 1 → PC S = 1, CL = 1: SEE PARA 2.4			4-12.2
CLA	0 0 1 0 1 1 1 1	2 F	0 → A			4-2.3
CLO	0 0 0 0 1 0 1 1	0 B	0 → O Register			4-10.4
COMX	0 0 0 0 0 0 0 0	0 0	$\bar{X}$ → X			4-11.2
CPAIZ	0 0 1 0 1 1 0 1	2 D	$\bar{A}$ + 1 → A	Y		4-4.12
DAN	0 0 0 0 0 1 1 1	0 7	A - 1 → A	Y		4-4.7
DMAN*	0 0 1 0 1 0 1 0	2 A	M(X,Y) - 1 → A	Y		4-4.4
DYN	0 0 1 0 1 1 0 0	2 C	Y - 1 → Y	Y		4-4.8
IA	0 0 0 0 1 1 1 0	0 E	A + 1 → A			4-4.5
IMAC*	0 0 1 0 1 0 0 0	2 8	M(X,Y) + 1 → A	Y		4-4.3
IYC	0 0 1 0 1 0 1 1	2 B	Y + 1 → Y	Y		4-4.6
KNEZ	0 0 0 0 1 0 0 1	0 9	K8, 4, 2, 1 ≠ 0		Y	4-9.1
LDP	0 0 0 1 C	1 –	I(C) → PB			4-12.4
LDX	0 0 1 1 1 1 B	3 –	I(B) → X			4-11.1
MNEZ	0 0 1 0 0 1 1 0	2 6	M(X,Y) ≠ 0		Y	4-6.1
RBIT	0 0 1 1 0 1 B	3 –	0 → M(X,Y,B)			4-7.2
RETN	0 0 0 0 1 1 1 1	0 F	CL = 1: SB → PC, PB → PA, 0 → CL CL = 0: PC + 1 → PC, PB → PA			4-12.3
RSTR	0 0 0 0 1 1 0 0	0 C	0 → R(Y), 0 ≤ Y ≤ 12			4-10.2
SAMAN	0 0 1 0 0 1 1 1	2 7	M(X,Y) - A → A	Y		4-4.2
SBIT	0 0 1 1 0 0 B	3 –	1 → M(X,Y,B)			4-7.1
SETR	0 0 0 0 1 1 0 1	0 D	1 → R(Y), 0 ≤ Y ≤ 12			4-10.1
TAM	0 0 0 0 0 0 1 1	0 3	A → M(X,Y)			4-3.1
TAMIY	0 0 1 0 0 0 0 0	2 0	A → M(X,Y), Y + 1 → Y			4-3.2
TAMZA	0 0 0 0 0 1 0 0	0 4	A → M(X,Y), 0 → A			4-3.3
TAY	0 0 1 0 0 1 0 0	2 4	A → Y			4-2.1
TBIT1	0 0 1 1 1 0 B	3 –	M(X,Y,B) = 1		Y	4-7.3
TCY	0 1 0 0 C	4 –	I(C) → Y			4-8.1
TCMIY	0 1 1 0 C	6 –	I(C) → M(X,Y), Y + 1 → Y			4-8.2
TDO	0 0 0 0 1 0 1 0	0 A	{SL, A} → O Register			4-10.3
TKA	0 0 0 0 1 0 0 0	0 8	K8, 4, 2, 1 → A			4-9.2
TMA	0 0 1 0 0 0 0 1	2 1	M(X,Y) → A			4-3.5
TMY	0 0 1 0 0 0 1 0	2 2	M(X,Y) → Y			4-3.4
TYA	0 0 1 0 0 0 1 1	2 3	Y → A			4-2.2
XMA	0 0 1 0 1 1 1 0	2 E	M(X,Y) ↔ A			4-3.6
YNEA	0 0 0 0 0 0 1 0	0 2	Y ≠ A, S → SL**		Y	4-6.2
YNEC	0 1 0 1 C	5 –	Y ≠ C		Y	4-6.3

*Execution of the DMAN or IMAC instruction does not change (increment or decrement) the content of the addressed memory cell.
**Only one instruction sets or resets Status Latch.

TABLE 3-3 MICROINSTRUCTION INDEX

Mnemonic	Opcode								Microinstructions: Fixed	Microinstructions: Programmable	Reference Paragraph
ALEC	0	1	1	1	C					CKP, NATN, CIN, C8	4-5.2
ALEM	0	0	1	0	1	0	0	1		MTP, NATN, CIN, C8	4-5.1
AMAAC	0	0	1	0	0	1	0	1		MTP, ATN, C8, AUTA	4-4.1
A6AAC	0	0	0	0	0	1	1	0		CKP,ATN,C8,AUTA	4-4.11
A8AAC	0	0	0	0	0	0	0	1		CKP, ATN, C8, AUTA	4-4.9
A10AAC	0	0	0	0	0	1	0	1		CKP, ATN, C8, AUTA	4-4.10
BR	1	0			W				BR		4-12.1
CALL	1	1			W				CALL		4-12.2
CLA	0	0	1	0	1	1	1	1		AUTA	4-2.3
CLO	0	0	0	0	1	0	1	1	CLO		4-10.4
COMX	0	0	0	0	0	0	0	0	COMX		4-11.2
CPAIZ	0	0	1	0	1	1	0	1		NATN,CIN,C8,AUTA	4-4.12
DAN	0	0	0	0	0	1	1	1		CKP, ATN, CIN, C8, AUTA	4-4.7
DMAN*	0	0	1	0	1	0	1	0		MTP, 15TN, C8, AUTA	4-4.4
DYN	0	0	1	0	1	1	0	0		YTP, 15TN, C8, AUTY	4-4.8
IA	0	0	0	0	1	1	1	0		ATN, CIN, AUTA	4-4.5
IMAC*	0	0	1	0	1	0	0	0		MTP, CIN, C8, AUTA	4-4.3
IYC	0	0	1	0	1	0	1	1		YTP, CIN, C8, AUTY	4-4.6
KNEZ	0	0	0	0	1	0	0	1		CKP, NE	4-9.1
LDP	0	0	0	1	C				LDP		4-12.4
LDX	0	0	1	1	1	1	B		LDX		4-11.1
MNEZ	0	0	1	0	0	1	1	0		MTP, NE	4-6.1
RBIT	0	0	1	1	0	1	B		RBIT		4-7.2
RETN	0	0	0	0	1	1	1	1	RETN		4-12.3
RSTR	0	0	0	0	1	1	0	0	RSTR		4-10.2
SAMAN	0	0	1	0	0	1	1	1		MTP, NATN, CIN, C8, AUTA	4-4.2
SBIT	0	0	1	1	0	0	B		SBIT		4-7.1
SETR	0	0	0	0	1	1	0	1	SETR		4-10.1
TAM	0	0	0	0	0	0	1	1		STO	4-3.1
TAMIY	0	0	1	0	0	0	0	0		STO, YTP, CIN, AUTY	4-3.2
TAMZA	0	0	0	0	0	1	0	0		STO, AUTA	4-3.3
TAY	0	0	1	0	0	1	0	0		ATN, AUTY	4-2.1
TBIT 1	0	0	1	1	1	0	B			CKP, CKN, MTP, NE	4-7.3
TCY	0	1	0	0	C					CKP, AUTY	4-8.1
TCMIY	0	1	1	0	C					CKM, YTP, CIN, AUTY	4-8.2
TDO	0	0	0	0	1	0	1	0	TDO		4-10.3
TKA	0	0	0	0	1	0	0	0		CKP, AUTA	4-9.2
TMA	0	0	1	0	0	0	0	1		MTP, AUTA	4-3.5
TMY	0	0	1	0	0	0	1	0		MTP, AUTY	4-3.4
TYA	0	0	1	0	0	0	1	1		YTP, AUTA	4-2.2
XMA	0	0	1	0	1	1	1	0		MTP, STO, AUTA	4-3.6
YNEA	0	0	0	0	0	0	1	0		YTP, ATN, NE, STSL	4-6.2
YNEC	0	1	0	1	C					YTP, CKN, NE	4-6.3

*Execution of the DMAN or IMAC instruction does not change (increment or decrement) the content of the addressed memory cell.

TABLE 3-4. BINARY CODING OF STANDARD INSTRUCTIONS

Opcode (binary list)	Opcode (hex)	Mnemonic	Action	Status C8	Status NE	Reference Paragraph
0 0 0 0 0 0 0 0	0 0	COMX	$\overline{X}$ → X			4-11.2
0 0 0 0 0 0 0 1	0 1	A8AAC	A + 8 → A	Y		4-4.9
0 0 0 0 0 0 1 0	0 2	YNEA	Y ≠ A, S → SL		Y	4-6.2
0 0 0 0 0 0 1 1	0 3	TAM	A → M(X,Y)			4-3.1
0 0 0 0 0 1 0 0	0 4	TAMZA	A → M(X,Y), 0 → A			4-3.3
0 0 0 0 0 1 0 1	0 5	A10AAC	A + 10 → A	Y		4-4.10
0 0 0 0 0 1 1 0	0 6	A6AAC	A + 6 → A	Y		4-4.11
0 0 0 0 0 1 1 1	0 7	DAN	A - 1 → A	Y		4-4.7
0 0 0 0 1 0 0 0	0 8	TKA	K8, 4, 2, 1 → A			4-9.2
0 0 0 0 1 0 0 1	0 9	KNEZ	K8, 4, 2, 1 ≠ 0		Y	4-9.1
0 0 0 0 1 0 1 0	0 A	TDO	{SL, A} → O Register			4-10.3
0 0 0 0 1 0 1 1	0 B	CLO	0 → O Register			4-10.4
0 0 0 0 1 1 0 0	0 C	RSTR	0 → R(Y), 0 ≤ Y ≤ 12			4-10.2
0 0 0 0 1 1 0 1	0 D	SETR	1 → R(Y), 0 ≤ Y ≤ 12			4-10.1
0 0 0 0 1 1 1 0	0 E	IA	A + 1 → A			4-4.5
0 0 0 0 1 1 1 1	0 F	RETN	CL = 1: SR → PC, PB → PA, 0 → CL CL = 0: PB → PA, PC + 1 → PC			4-12.3
0 0 0 1 C	1 –	LDP	I (C) → PB			4-12.4
0 0 1 0 0 0 0 0	2 0	TAMIY	A → M(X,Y), Y + 1 → Y			4-3.2
0 0 1 0 0 0 0 1	2 1	TMA	M(X,Y) → A			4-3.5
0 0 1 0 0 0 1 0	2 2	TMY	M(X,Y) → Y			4-3.4
0 0 1 0 0 0 1 1	2 3	TYA	Y → A			4-2.2
0 0 1 0 0 1 0 0	2 4	TAY	A → Y			4-2.1
0 0 1 0 0 1 0 1	2 5	AMAAC	M(X,Y) + A → A	Y		4-4.1
0 0 1 0 0 1 1 0	2 6	MNEZ	M(X,Y) ≠ 0		Y	4-6.1
0 0 1 0 0 1 1 1	2 7	SAMAN	M(X,Y) - A → A	Y		4-4.2
0 0 1 0 1 0 0 0	2 8	IMAC*	M(X,Y) + 1 → A	Y		4-4.3
0 0 1 0 1 0 0 1	2 9	ALEM	A ≤ M(X,Y)	Y		4-5.1
0 0 1 0 1 0 1 0	2 A	DMAN*	M(X,Y) - 1 → A	Y		4-4.4
0 0 1 0 1 0 1 1	2 B	IYC	Y + 1 → Y	Y		4-4.6
0 0 1 0 1 1 0 0	2 C	DYN	Y - 1 → Y	Y		4-4.8
0 0 1 0 1 1 0 1	2 D	CPAIZ	$\overline{A}$ + 1 → A	Y		4-4.12
0 0 1 0 1 1 1 0	2 E	XMA	M(X,Y) ↔ A			4-3.6
0 0 1 0 1 1 1 1	2 F	CLA	0 → A			4-2.3
0 0 1 1 0 0 B	3 –	SBIT	1 → M(X,Y,B)			4-7.1
0 0 1 1 0 1 B	3 –	RBIT	0 → M(X,Y,B)			4-7.2
0 0 1 1 1 0 B	3 –	TBIT 1	M(X,Y,B) = 1		Y	4-7.3
0 0 1 1 1 1 B	3 –	LDX	I(B) → X			4-11.1
0 1 0 0 C	4 –	TCY	I(C) → Y			4-8.1
0 1 0 1 C	5 –	YNEC	Y ≠ C		Y	4-6.3
0 1 1 0 C	6 –	TCMIY	I(C) → M(X,Y), Y + 1 → Y			4-8.2
0 1 1 1 C	7 –	ALEC	A ≤ C	Y		4-5.2
1 0 W	– –	BR	S = 1: I(W) → PC, PB → PA S = 0: PC + 1 → PC, 1 → S			4-12.1
1 1 W	– –	CALL	S = 1: PC+1 → SR, PA↔PB, 1→CL, I(W)→PC S = 0: PC + 1 → PC, 1 → S			4-12.2

*Execution of the DMAN or IMAC instruction does not change (increment or decrement) the content of the addressed memory cell.

MACHINE INSTRUCTION CODE

A				Z			
I(0)	I(1)	I(2)	I(3)	I(4)	I(5)	I(6)	I(7)
MSB							LSB

A \ Z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	*OPERAND
0	COMX	A8AAC	YNEA	TAM	TAMZA	A10AAC	A6AAC	DAN	TKA	KNEZ	TDO	CLO	RSTR	SETR	IA	RETN	
1	LDP																C
2	TAMIY	TMA	TMY	TYA	TAY	AMAAC	MNEZ	SAMAN	IMAC	ALEM	DMAN	IYC	DYN	CPAIZ	XMA	CLA	
3	SBIT				RBIT				TBIT1				LDX				B
4	TCY																C
5	YNEC																C
6	TCMIY																C
7	ALEC																C
8–B	BR																W
C–F	CALL																W

*C = constant; B = B field; W = memory address.

FIGURE 3-1 STANDARD INSTRUCTION MAP, TMS1000/1200

SECTION IV

TMS 1000/1200
STANDARD INSTRUCTION SET DEFINITIONS

4-1 GENERAL.

4-1.1 INSTRUCTION SET. An instruction set of 43 basic instructions programs the TMS1000 ROM. The instruction mnemonics relate directly to the instruction effects to reinforce the user's knowledge of the hardware.

The instructions are grouped in this section according to function as listed in Tables 2-1.2 and 3-1. Each instruction is described in a common format that defines the mnemonic, status effects, formats, operands, symbolic description, purpose, execution description, and microinstructions performed.

4-1.2 EFFECT ON STATUS. Eighteen instructions conditionally affect the machine status logic. The mnemonics for these instructions contain a one- or two-character descriptor to indicate how status logic is affected. Each descriptor (shown in Table 4-1.1) indicates the condition where status will remain set (logic ONE). The conditional instructions, branch and call, are successful only if status is set. The mnemonic descriptor therefore indicates the conditions under which an immediately following branch or call will be performed. If the instruction results do not meet the descriptor's condition, then status is reset (logic ZERO) and any immediately following branch or call will not be performed. [Recall that status logic in the reset (ZERO) state affects only branches or calls in the next instruction cycle before returning to the normal (logic ONE) state.]

TABLE 4-1.1. DESCRIPTOR ACTION

Descriptor		Cause/Result that Transfers ONE to Status
Last Character In Mnemonic	C	Carry out during addition or increment instructions
	N	No borrow during subtraction or decrement instructions
	Z	Zero result from 2's complement
	1	Tested memory bit is a logic ONE
Middle of Mnemonic	-LE-	Is less than or equal to
	-NE-	Is not equal to

Each instruction description in this section contains a status description. The way in which the instruction depends upon status or sets status is defined as follows:

- Set: The instruction unconditionally forces status to ONE and is not conditional upon status.
- Carry Into Status: The value of the carry from the adder is transferred to status. In the subtraction instructions, carry = $\overline{\text{borrow}}$.

- Comparison Result Into Status: The logical comparison value from the ALU is transferred to status (equal: ZERO to status; unequal: ONE to status).

- Conditional On Status: The instruction's execution results are conditional upon the state of the status. After executing the instruction, status is unconditionally equal to ONE.

4-1.3 INSTRUCTION FORMATS. The machine instructions have been divided into four instruction formats. A format subdivides the eight bits of each instruction into fields. These fields contain the operation code and operands.

4-1.3.1 Instruction Format I:

0	1	2	3	4	5	6	7
OP		W					
		MSB					LSB

This format has a two-bit operation-code field, and the operand is a six-bit ROM-word address field. This format is used for program control by branch and call instructions. The operand, the branch address, has a value of 0 to 63.

4-1.3.2 Instruction Format II:

0	1	2	3	4	5	6	7
OP				C			
				LSB			MSB

This format has a four-bit operation-code field, the operand is a four-bit constant field. This format is used for instructions that contain an immediate value that loads RAM memory or a register with a constant.

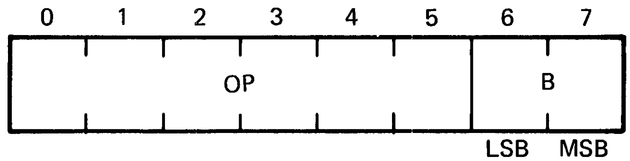
NOTE

The constant value (from 0 to 15) is reversed in the C field. The assembler properly converts any decimal value into this machine code format.

Example: The constant value 8 would appear in the machine instruction as follows.

0	1	2	3	4	5	6	7
OP				0	0	0	1

4-1.3.3 Instruction Format III:



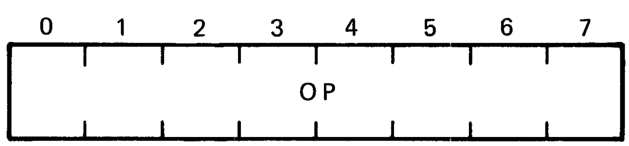
This format has a six-bit operation code, and the operand is a two-bit RAM bit address field. This format is used for addressing a bit in a RAM word. Also, B describes the two-bit X address operand for the LDX command.

NOTE

The bit address, B, is inverted. The assembler converts decimal value into this machine format as shown below:

BIT ADDRESS	B-FIELD I(6)	B-FIELD I(7)	RAM WORD MSB			RAM WORD LSB
0	0	0				X
1	1	0			X	
2	0	1		X		
3	1	1	X			

4-1.3.4 Instruction Format IV:



This format defines an eight-bit operation code field only. Instructions of this format have no constant operands. The instruction always performs the same action, for example transferring the accumulator to the Y register.

4-1.4 MICROINSTRUCTIONS. In paragraphs 4-2 to 4-12, the mnemonics for the microinstructions performed by each machine instruction are listed to refer the user to the hardware steps performed. The programmable microinstructions (16) are used in 31 instructions. The 12 fixed microinstructions, which are decoded by hardwired logic (non-programmable), are used in the 12 "fixed" instructions.

4-1.5 CODING FORMAT. Coding programs are covered in detail in the TMS1000 Software Users Guide which covers the assembler and the simulator programs. The following rules should be followed in writing a program on a coding form.

a. Label fields are a maximum of eight alphanumeric characters starting with an alphabetic character. The label field begins in column one.

b. The operation code is to the right of a label, the two separated by at least one blank space. If no label is used, the operation code begins after the first column (second column or further over).

c. The operand is to the right of the operation code, the two separated by at least one blank space.

d. A comment is to the right of the operand, the two separated by at least one blank space. If a comment occupies a separate line, it must begin with an asterisk in column one.

Figure 4-1.1 is a sample of a filled out coding form. For legibility, it is recommended that the fields begin in the following columns:

a. Label fields begin in column one.

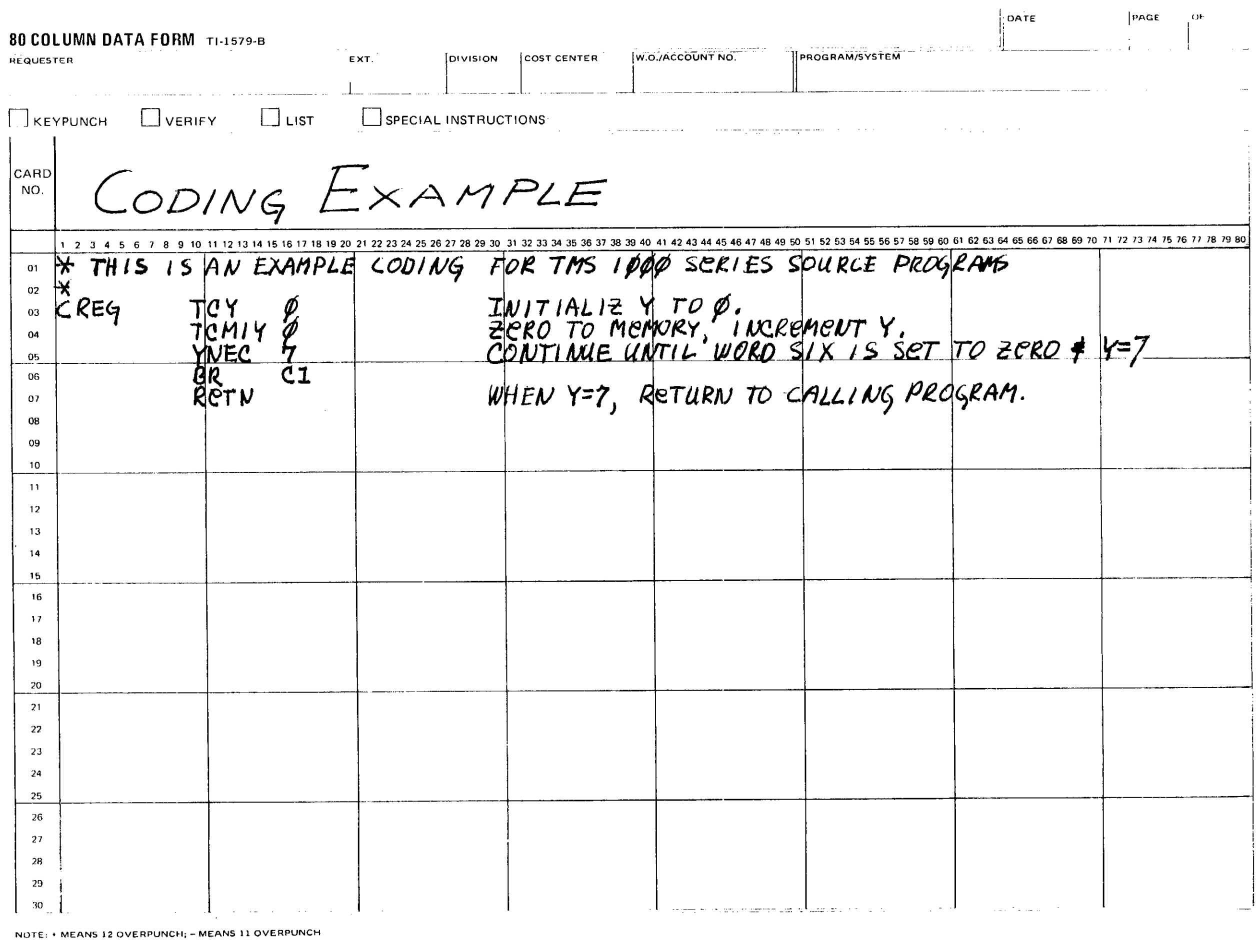
b. Operation codes should begin in column 10.

c. Operands should begin in column 16.

d. Comments to an instruction should begin in column 30.

e. Comment lines begin in column one with an asterisk.

4-1.6 EXAMPLES. Examples are provided for various instructions in paragraphs 4-2 through 4-12. These examples illustrate typical applications and how the instructions are combined to perform a function. The contents of the affected hardware registers and memory are illustrated so the reader can follow through an example, step by step if necessary.



80 COLUMN DATA FORM TI-1579-B

REQUESTER | EXT. | DIVISION | COST CENTER | W.O./ACCOUNT NO. | PROGRAM/SYSTEM | DATE | PAGE | OF

☐ KEYPUNCH ☐ VERIFY ☐ LIST ☐ SPECIAL INSTRUCTIONS

CARD NO. — CODING EXAMPLE

```
01 * THIS IS AN EXAMPLE CODING FOR TMS 1ØØØ SERIES SOURCE PROGRAMS
02 *
03 CREG      TCY   Ø      INITIALIZ Y TO Ø.
04           TCMIY Ø      ZERO TO MEMORY, INCREMENT Y.
05           YNEC  7      CONTINUE UNTIL WORD SIX IS SET TO ZERO ≠ Y=7
06           BR    CI
07           RETN         WHEN Y=7, RETURN TO CALLING PROGRAM.
```

NOTE: + MEANS 12 OVERPUNCH; – MEANS 11 OVERPUNCH

FIGURE 4-1.1 SAMPLE CODING FORM

4-2 REGISTER TO REGISTER TRANSFER INSTRUCTION.

Register to register instructions affect only the indicated register. These instructions are all format IV type and have no operands.

The only data path for these instructions is through the adder. The reader should refer to the data paths on the functional block diagram (Figure 2-1.1) as necessary.

4-2.1 TRANSFER ACCUMULATOR TO Y REGISTER.

0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	0

MENMONIC: TAY

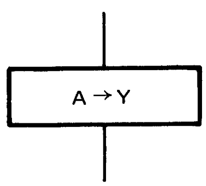
STATUS: Set

FORMAT: IV

ACTION: A → Y

DESCRIPTION: The accumulator contents are unconditionally transferred to the Y register. The accumulator contents are unaltered.

MICROINSTRUCTIONS: ATN, AUTY



	X	Y	A	M(XY)	S
*Initial	0	5	E	0	1
TAY	0	E	E	9	1

*In this and all subsequent examples, the initial conditions are specified above the dotted line. Status, if reset to zero by an instruction, is shown with the affected instruction - the following instruction cycle. Register contents are shown in hexadecimal.

4-2.2 TRANSFER Y REGISTER TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	0	0	1	1

MNEMONIC: TYA

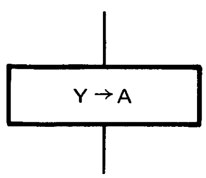
STATUS: Set

FORMAT: IV

ACTION: Y → A

DESCRIPTION: The four-bit contents of the Y register are unconditionally transferred to the accumulator. The contents of the Y register are unaltered.

MICROINSTRUCTIONS: YTP, AUTA



	X	Y	A	M(XY)	S
	2	9	6	2	1
TYA	2	9	9	2	1

4-2.3 CLEAR ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	1	1	1	1

MNEMONIC: CLA

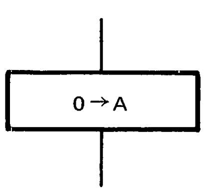
STATUS: Set

FORMAT: IV

ACTION: 0 → A

DESCRIPTION: The contents of the accumulator are unconditionally cleared to zero.

MICROINSTRUCTIONS: AUTA



	X	Y	A	M(XY)	S
	1	4	7	0	1
CLA	1	4	0	0	1

4-3 REGISTER TO MEMORY, MEMORY TO REGISTER TRANSFER INSTRUCTIONS.

These instructions are used to transfer four-bit data information between registers and RAM memory for storage or retrieval of information. These instructions are all format IV type and have no operands.

The only register-to-memory data path is from the accumulator into memory. Notice that the Y register may not be transferred into memory directly. Data transferred from memory to the registers always passes through the adder and may then be directed into either the accumulator or the Y register.

No operands are used.

4-3.1 TRANSFER ACCUMULATOR TO MEMORY.

0	1	2	3	4	5	6	7
0	0	0	0	0	0	1	1

MNEMONIC: TAM

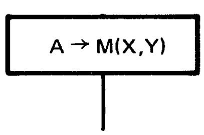
STATUS: Set

FORMAT: IV

ACTION: A → M(X,Y)

DESCRIPTION: The four-bit contents of the accumulator are stored in the memory (RAM) location addressed by the X and Y registers. The accumulator contents are unaltered.

MICROINSTRUCTION: STO



	X	Y	A	M(XY)	S
	0	6	4	2	1
TAM	0	6	4	4	1

4-3.2 TRANSFER ACCUMULATOR TO MEMORY AND INCREMENT Y REGISTER.

0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	0

MNEMONIC: TAMIY

STATUS: Set

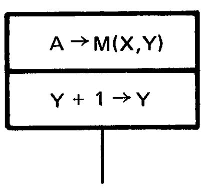
FORMAT: IV

ACTION: A → M(X,Y)
Y + 1 → Y

PURPOSE: Y register sequentially addresses a file of sixteen RAM words, and the addressed words are set to the accumulator value(s), during initialization routines for example.

DESCRIPTION: The contents of the accumulator are stored in the memory location addressed by the X and Y registers. Then the contents of the Y register are incremented by one. The accumulator contents are unaltered.

MICROINSTRUCTIONS: STO, YTP, CIN, AUTY.



	X	Y	A	M(XY)	S
	3	9	0	2	1
TAMIY	3	9	0	0	1
	3	A	0	0	1

4-3.3 TRANSFER ACCUMULATOR TO MEMORY AND ZERO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	0	1	0	0

MNEMONIC: TAMZA

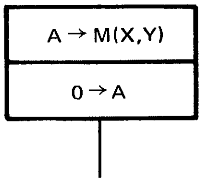
STATUS: Set

FORMAT: IV

ACTION: A → M(X,Y)
0 → A

DESCRIPTION: The contents of the accumulator are stored in the RAM location addressed by the X and Y registers. The contents of the accumulator are then cleared to zero.

MICROINSTRUCTIONS: STO, AUTA



	X	Y	A	M(XY)	S
	0	E	B	3	1
TAMZA	0	E	B	B	1
	0	E	0	B	1

4-3.4 TRANSFER MEMORY TO Y REGISTER.

0	1	2	3	4	5	6	7
0	0	1	0	0	0	1	0

MNEMONIC: TMY

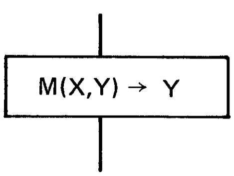
STATUS: Set

FORMAT: IV

ACTION: M(X,Y) → Y

DESCRIPTION: The contents of the RAM location addressed by the X and Y registers are loaded into the Y register. Memory contents are unaltered.

MICROINSTRUCTIONS: MTP, AUTY



	X	Y	A	M(X,Y)	S
	1	3	9	A	1
TMY	1	A	9	2	1

4-3.5 TRANSFER MEMORY TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	0	0	0	1

MNEMONIC: TMA

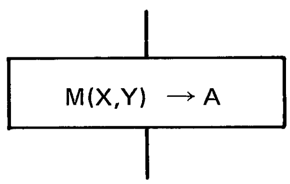
STATUS: Set

FORMAT: IV

ACTION: M(X,Y) → A

DESCRIPTION: The contents of the RAM location addressed by the X and Y registers are loaded into the accumulator. Memory contents are unaltered.

MICROINSTRUCTIONS: MTP, AUTA



	X	Y	A	M(X,Y)	S
	0	9	3	2	1
TMA	0	9	3	3	1

4-3.6 EXCHANGE MEMORY AND ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	1	1	1	0

MNEMONIC: XMA

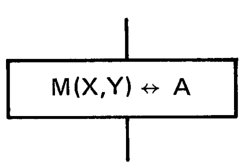
STATUS: Set

FORMAT: IV

ACTION: M(X,Y) ↔ A

DESCRIPTION: The memory contents (addressed by the X and Y registers) are exchanged with the accumulator contents. For example, this instruction is useful to retrieve a memory word into the accumulator for an arithmetic operation and save the current accumulator contents in the RAM. The accumulator may be restored by a second XMA instruction.

MICROINSTRUCTIONS: MTP, STO, AUTA

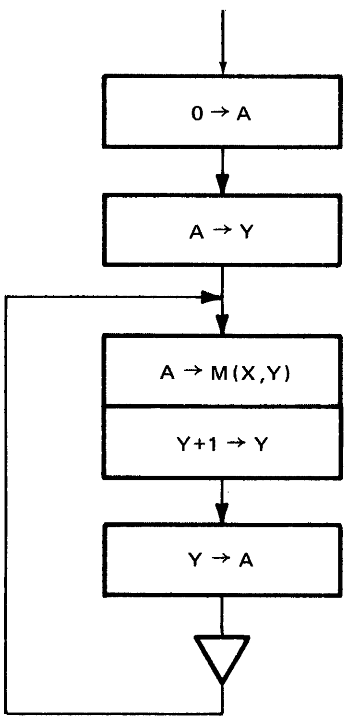


	X	Y	A	M(X,Y)	S
	1	9	3	F	1
XMA	1	9	F	3	1

4-3.7 REGISTER/MEMORY TRANSFER EXAMPLE. The following is a simple example that combines some of the instructions into a small program.

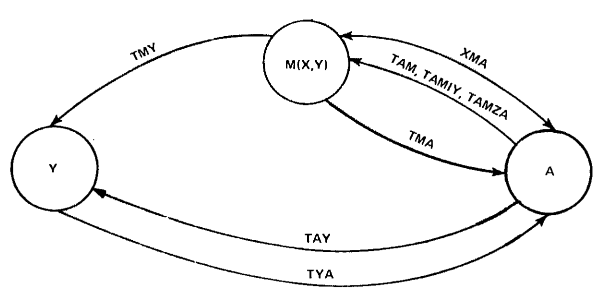
The program is designed to load the 16 words in a RAM X-file with their Y addresses. For example, memory word M(0,5) will contain a 5. This program contains a branch instruction to allow the program to loop. To simplify this illustration, this program is written to loop continuously. The address in the Y register always contains a value in the range $0 \leqslant Y \leqslant 15$. (If the Y register contains F_{16} and is incremented, it returns to 0.)

```
LABEL    OP CODE   OPERAND   COMMENT

         CLA                 ACCUMULATOR TO ZERO
         TAY                 Y REG TO ZERO
LOOP     TAMIY               STORE A IN MEMORY AND INCREMENT Y.
         TYA                 TRANSFER Y TO A.
         BR        LOOP      BRANCH TO LOOP
```



	X	Y	A	M(XY)
	0	F	F	B
CLA	0	F	0	B
TAY	0	0	0	3
TAMIY	0	0	0	0
	0	1	0	?
TYA	0	1	1	?
BR	0	1	1	?
TAMIY	0	1	1	1
	0	2	1	?
TYA	0	2	2	?
BR	0	2	2	?

The following diagram summarizes data transfers that may take place in one instruction cycle:



4-4 ARITHMETIC INSTRUCTIONS.

The instructions herein define a class of arithmetic operations. All arithmetic operations are performed by the adder. The arithmetic operands originate from memory, registers, or instruction constants. The results from the adder are stored into the accumulator or Y register, and carry (and borrow) information is transferred to status. The carry out bit is important for multi-precision arithmetic operations and loop control.

The adder is the center of the arithmetic operations. Because the adder can only perform add operations, subtraction is performed by the two's complement system.

All arithmetic instructions are in format IV and have no variable operands. Although in some cases, microinstruction CKP enables CKI logic to the adder and allows the four least significant bits of the instruction to be used in the operation.

4-4.1 ADD MEMORY TO ACCUMULATOR, RESULTS TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	1

MNEMONIC: AMAAC

STATUS: Carry into status

FORMAT: IV

ACTION:
M(X,Y) + A → A
1 → S if sum > 15
0 → S if sum ⩽ 15

DESCRIPTION: The contents of the memory location addressed by the X and Y registers are added to the contents of the accumulator. The result is stored into the accumulator. The resulting carry information is transferred to status. A sum that is greater than 15 results in a carry and a ONE to status. Memory contents are unaltered.

MICROINSTRUCTIONS: MTP, ATN, C8, AUTA

EXAMPLE: Assume that the RAM word contains 15 and the accumulator contains a one.

```
                1 1 1 1   M(X,Y)
              + 0 0 0 1   A
CARRY = 1       -------
                0 0 0 0   A CONTENTS
```

Status is set (to ONE) as a result of the carry.

4-4.2 SUBTRACT ACCUMULATOR FROM MEMORY, RESULT TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	0	1	1	1

MNEMONIC: SAMAN

STATUS: Carry into status

FORMAT: IV

ACTION:

M(X,Y) - A → A

1 → S if A ⩽ M(X,Y) } Initial Conditions
0 → S if A > M(X,Y) }

DESCRIPTION: The contents of the accumulator are subtracted from the memory word addressed by the X and Y registers via two's complement addition. The result is stored into the accumulator. Status is set if the accumulator is less than or equal to the memory word, indicating that no-borrow occurred. A borrow occurs when the accumulator is greater than the memory word and status is reset (to ZERO).

MICROINSTRUCTIONS: MTP, NATN, CIN, C8, AUTA.

EXAMPLE: Assume that the current RAM word contains a 5 and the accumulator contains a 2. The SAMAN instruction will perform as follows:

```
              0 1 0 1     M(X,Y)
              1 1 0 1     +Ā  } -|A|  =  -2
                    1     +1  }
              -------
CARRY = 1     0 0 1 1     A = 3
```

Status is ONE, indicating that no borrow occurred.

4-4.3 INCREMENT MEMORY AND LOAD INTO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	1	0	1	0	0	0

MNEMONIC: IMAC

STATUS: Carry into status

FORMAT: IV

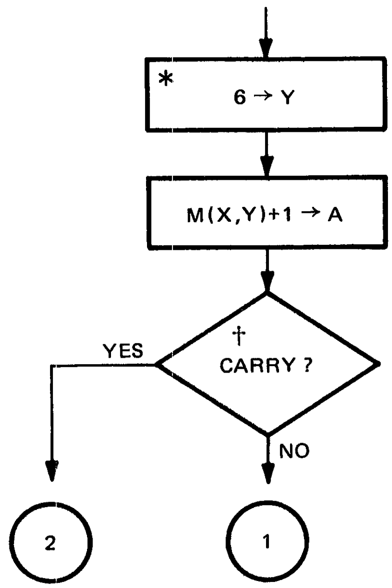
ACTION:

M(X,Y) + 1 → A

1 → S if M(X,Y) = 15 } Initial Conditions
0 → S if M(X,Y) < 15 }

DESCRIPTION: The contents of memory addressed by the X and Y registers are fetched. One is added to this word and the result is stored in the accumulator. The resulting carry information is transferred to status. Status is set if the sum is greater than 15. Memory is left unaltered.

MICROINSTRUCTIONS: MTP, CIN, C8, AUTA



1. NO CARRY

	X	Y	A	M(XY)	S	BRANCH
	2	5	3	7	1	
TCY 6	2	6	3	8	1	
IMAC	2	6	9	8	1	
BR	2	6	9	8	0	NO

2. CARRY OCCURS

	X	Y	A	M(XY)	S	BRANCH
	2	7	4	0	1	
TCY 6	2	6	4	F	1	
IMAC	2	6	0	F	1	
BR	2	6	0	F	1	YES

*See paragraph 4-8.1 for RAM Y addressing explanation.
†See paragraph 4-12.1 for branch instruction description.

4-4.4 DECREMENT MEMORY AND LOAD INTO ACCUMULATOR.

MNEMONIC: DMAN

0	1	2	3	4	5	6	7
0	0	1	0	1	0	1	0

STATUS: Carry into status

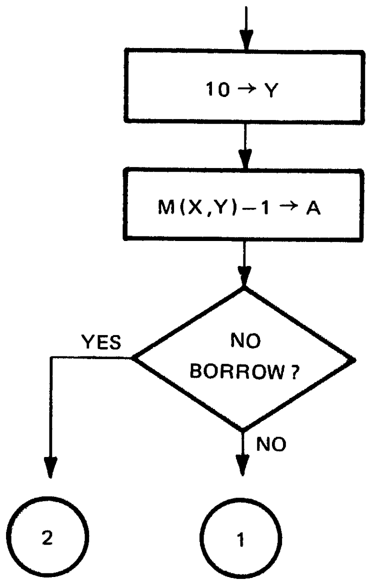
FORMAT: IV

ACTION: M(X,Y) - 1 → A

1 → S if M(X,Y) ≥ 1 }
0 → S if M(X,Y) = 0 } Initial Conditions

DESCRIPTION: The contents of memory addressed by the X and Y registers are fetched. One is subtracted from this word (add F_{16}), and the result is placed in the accumulator. The resulting carry information is transferred to status. If memory is greater than or equal to one, status is set indicating that no borrow occurred. Memory contents are unaltered.

MICROINSTRUCTIONS: MTP, 15TN, C8, AUTA



1. BORROW OCCURS

	X	Y	A	M(XY)	S	BRANCH
	0	4	3	9	1	
TCY 10	0	A	3	0	1	
DMAN	0	A	F	0	1	
BR	0	A	F	0	0	NO

2. NO BORROW

	X	Y	A	M(XY)	S	BRANCH
	0	5	0	7	1	
TCY 10	0	A	5	2	1	
DMAN	0	A	1	2	1	
BR	0	A	1	2	1	YES

4-4.5 INCREMENT ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	1	0

MNEMONIC: IA

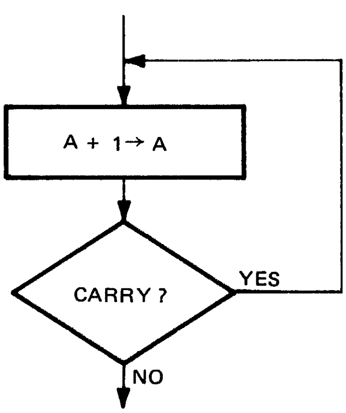
STATUS: Set

FORMAT: IV

ACTION: A + 1 → A

DESCRIPTION: The contents of the accumulator are incremented by one. The result is placed back into the accumulator. Carry to status is not performed.

MICROINSTRUCTIONS: ATN, CIN, AUTA



WARNING

Do not use this example. An infinite loop will result. Do not attempt to use this sequence because status is always a ONE.

4-4.6 INCREMENT Y REGISTER.

0	1	2	3	4	5	6	7
0	0	1	0	1	0	1	1

MNEMONIC: IYC

STATUS: Carry into status

FORMAT: IV

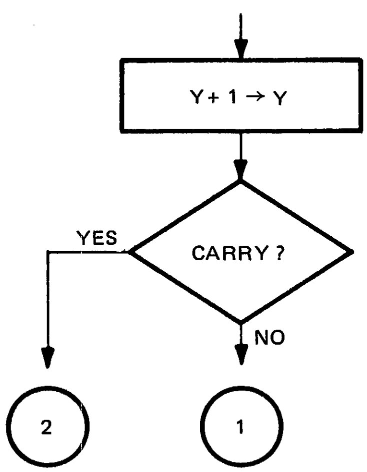
ACTION:

$Y + 1 \rightarrow Y$

$\left.\begin{array}{l} 1 \rightarrow S \text{ if } Y = 15 \\ 0 \rightarrow S \text{ if } Y < 15 \end{array}\right\}$ Initial Conditions

DESCRIPTION: The contents of the Y register are incremented by one. The result is placed back into the Y register. Resulting carry information is transferred to status. A sum greater than 15 results in status being set.

MICROINSTRUCTIONS: YTP, CIN, C8, AUTY.



1. NO CARRY

	X	Y	A	M(XY)	S	BRANCH
	0	E	2	4	1	
IYC	0	F	2	6	1	
BR	0	F	2	6	0	NO

2. CARRY OCCURS

	X	Y	A	M(XY)	S	BRANCH
	1	F	0	5	1	
IYC	1	0	0	9	1	
BR	1	0	0	9	1	YES

4-4.7 DECREMENT ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	0	1	1	1

MNEMONIC: DAN

STATUS: Carry into status

FORMAT: IV

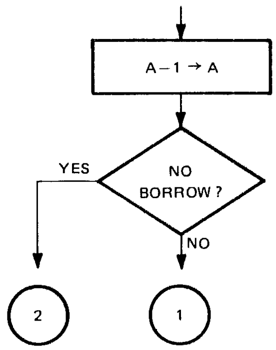
ACTION:

$A - 1 \rightarrow A$

$\left.\begin{array}{l} 1 \rightarrow S \text{ if } A \geqslant 1 \\ 0 \rightarrow S \text{ if } A = 0 \end{array}\right\}$ Initial Conditions

DESCRIPTION: The contents of the accumulator are decremented by one (add F_{16}). If a borrow results, status is reset to a logic ZERO. If accumulator contents are greater than one, there is no borrow, and status is set to a ONE.

MICROINSTRUCTIONS: CKP, ATN, CIN, C8, AUTA



1. BORROW OCCURS

	X	Y	A	M(XY)	S	BRANCH
	3	B	0	0	1	
DAN	3	B	F	0	1	
BR	3	B	F	0	0	NO

2. NO BORROW

	X	Y	A	M(XY)	S	BRANCH
	3	C	5	5	1	
DAN	3	C	4	5	1	
BR	3	C	4	5	1	YES

4-4.8 DECREMENT Y REGISTER.

MNEMONIC: DYN

0	1	2	3	4	5	6	7
0	0	1	0	1	1	0	0

STATUS: Carry into status

FORMAT: IV

ACTION:

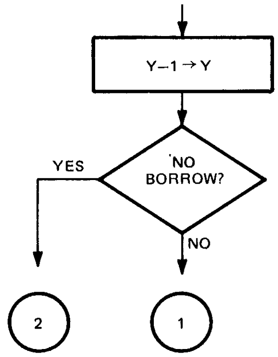
$Y - 1 \rightarrow Y$

$\left.\begin{array}{l} 1 \rightarrow S \text{ if } Y \geqslant 1 \\ 0 \rightarrow S \text{ if } Y = 0 \end{array}\right\}$ Initial Conditions

PURPOSE: To decrement the contents of the Y register by one.

DESCRIPTION: The contents of the Y register are decremented by one. This is performed by adding a minus one (F_{16}). Resulting carry information is transferred into Status. If the result is not equal to 15, status will be set indicating no borrow.

MICROINSTRUCTIONS: YTP, 15TN, C8, AUTY



1. BORROW OCCURS

	X	Y	A	M(XY)	S	BRANCH
	1	0	5	2	1	
DYN	1	F	5	4	1	
BR	1	F	5	4	0	NO

2. NO BORROW

	X	Y	A	M(XY)	S	BRANCH
	1	2	6	4	1	
DYN	1	1	6	3	1	
BR	1	1	6	3	1	YES

4-4.9 ADD 8 TO ACCUMULATOR, RESULTS TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	1
				LSB			MSB

MNEMONIC: A8AAC

STATUS: Carry into status

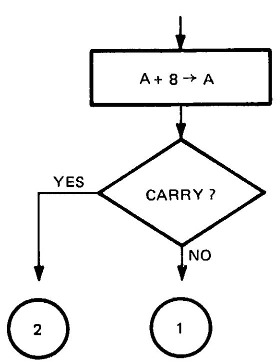
FORMAT: IV

ACTION: A + 8 → A
1 → S if sum > 15
0 → S if sum ⩽ 15

PURPOSE: To add the constant eight (8) to the accumulator, flipping the most significant bit of the accumulator.

DESCRIPTION: The constant eight (8), from the four low order bits of the instruction, is added to the accumulator contents. Carry information is transferred into status. A sum greater than 15 will generate a carry and will set status.

MICROINSTRUCTIONS: CKP, ATN, C8, AUTA



1. NO CARRY

	X	Y	A	M(XY)	S	BRANCH
	0	4	2	E	1	
A8AAC	0	4	A	E	1	
BR	0	4	A	E	0	NO

2. CARRY OCCURS

	X	Y	A	M(XY)	S	BRANCH
	1	F	9	2	1	
A8AAC	1	F	1	2	1	
BR	1	F	1	2	1	YES

4-4.10 ADD 10 TO ACCUMULATOR, RESULTS TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	0	1	0	1
				LSB			MSB

MNEMONIC: A10AAC

STATUS: Carry into status

FORMAT: IV

ACTION:
A + 10 → A
1 → S if sum > 15
0 → S if sum ⩽ 15

PURPOSE: To add the constant 10 to the accumulator. This is useful for BCD correction during subtraction.

DESCRIPTION: The constant ten (10), from the four low order bits of the instruction, is added to the accumulator's contents. Carry information is transferred into status and a sum greater than 15 sets status.

MICROINSTRUCTIONS: CKP, ATN, C8, AUTA

EXAMPLE: See paragraph 4-4.14.

4-4.11 ADD 6 TO ACCUMULATOR, RESULT TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	0	1	1	0
				LSB			MSB

MNEMONIC: A6AAC

STATUS: Carry into status

FORMAT: IV

ACTION:
A + 6 → A
1 → S if sum > 15
0 → S if sum ⩽ 15

PURPOSE: To add the constant 6 to the accumulator. This is useful for BCD correction during addition.

DESCRIPTION: The constant six (6), from the four low order bits of the instruction, is added to the accumulator contents. Carry information is transferred into status. A sum greater than 15 will result in a carry and set status.

MICROINSTRUCTIONS: CKP, ATN, C8, AUTA

EXAMPLE: See paragraph 4-4.13.

4-4.12 COMPLEMENT ACCUMULATOR AND INCREMENT (two's complement accumulator)

MNEMONIC: CPAIZ

0	1	2	3	4	5	6	7
0	0	1	0	1	1	0	1

STATUS: Carry into status

FORMAT: IV

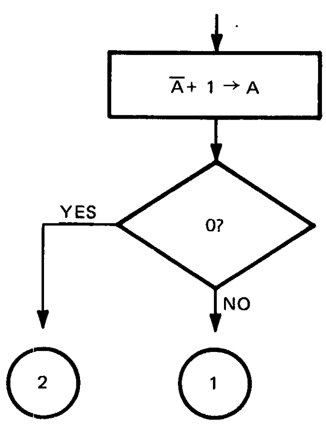
ACTION: $\overline{A} + 1 \rightarrow A$

$1 \rightarrow S$ if $A = 0$ } Initial Conditions
$0 \rightarrow S$ if $A \neq 0$ }

PURPOSE: To obtain the two's complement of the word in the accumulator.

DESCRIPTION: The two's complement of the accumulator is computed by adding one to the one's complement of the accumulator and storing the result in the accumulator. Carry information is transferred into status. If the accumulator contents are ZERO, carry occurs, and status is set (to ONE).

MICROINSTRUCTIONS: NATN, CIN, C8, AUTA



1. NO CARRY

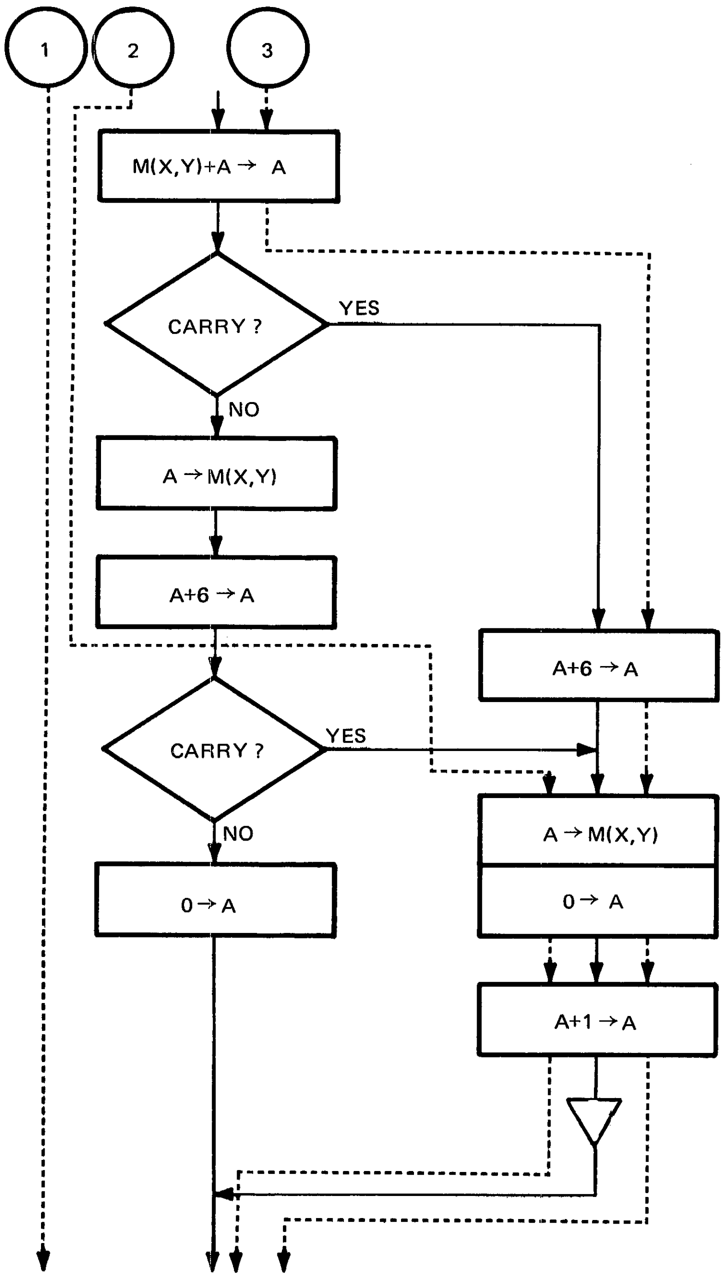
	X	Y	A	M(XY)	S	BRANCH
	0	1	7	5	1	
CPAIZ	0	1	9	5	1	
BR	0	1	9	5	0	NO

2. CARRY OCCURS

	X	Y	A	M(XY)	S	BRANCH
	0	3	0	6	1	
CPAIZ	0	3	0	6	1	
BR	0	3	0	6	1	YES

4-4.13 ADDITION INSTRUCTION EXAMPLE. The following example illustrates the addition arithmetic instructions. This example shows adding a word to a BCD number in memory. BCD correction is performed to keep the digit in the range 0 to 9. Upon exit from this routine the accumulator contains a one if a carry has resulted or a zero if no carry has resulted.

```
LABEL     OP CODE   OPERAND   COMMENT

          AMAAC               ADD CURRENT DIGIT TO A
          BR        FIXUP     BRANCH IF CARRY (SUM > 15)
          TAM                 TRANSFER A TO MEMORY
          A6AAC               ADD 6, TEST FOR DIGIT 10 TO 15
          BR        CORRECT   BRANCH IF CARRY
          CLA                 CLEAR ACCUMULATOR
CONTU                         EXIT
          .
          .
          .
FIXUP     A6AAC               ADD 6 TO CORRECT TO BCD
CORRECT   TAMZA               TRANSFER A TO MEMORY, CLEAR A
          IA                  INCREMENT ACCUMULATOR
          BR        CONTU     EXIT
          .
          .
          .
```



After execution:

A=0 NO CARRY

A=1 CARRY

1. BCD CORRECTION NOT NECESSARY

	X	Y	A	M(X,Y)	S	BR
	2	4	3	6	1	
AMAAC	2	4	9	6	1	
BR	2	4	9	6	0	NO
TAM	2	4	9	9	1	
A6AAC	2	4	F	9	1	
BR	2	4	F	9	0	NO
CLA	2	4	0	9	1	

2. BCD CORRECTION NECESSARY

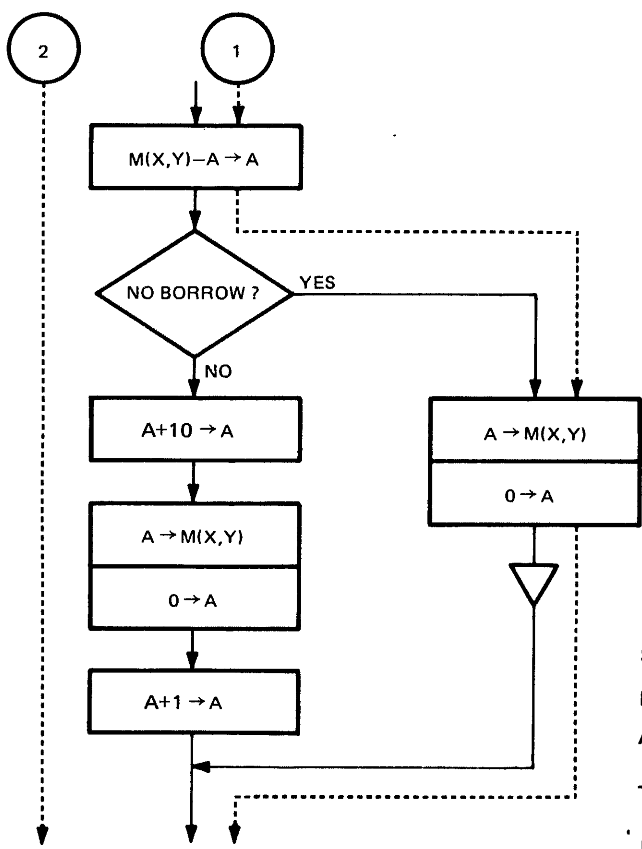
	X	Y	A	M(X,Y)	S	BR
AMAAC	0	5	6	5	1	
BR	0	5	B	5	0	NO
TAM	0	5	B	B	1	
A6AAC	0	5	1	B	1	
BR	0	5	1	B	1	YES
TAMZA	0	5	1	1	1	
	0	5	0	1	1	
IA	0	5	1	1	1	
BR	0	5	1	1	1	YES

3. BCD CORRECTION NECESSARY

	X	Y	A	M(X,Y)	S	BR
	1	7	8	9	1	
AMAAC	1	7	1	9	1	
BR	1	7	1	9	1	YES
A6AAC	1	7	7	9	1	
TAMZA	1	7	7	7	1	
	1	7	0	7	1	
IA	1	7	1	7	1	
BR	1	7	1	7	1	YES

4-4.14 **SUBTRACTION EXAMPLE.** The following example illustrates using arithmetic instructions to perform subtraction from a BCD value in memory. The example uses BCD correction to keep the values in the range of 0 to 9. Upon exit from the routine, the accumulator contains a one if a borrow has occurred or a zero if no borrow.

```
LABEL     OP CODE   OPERAND   COMMENT

          SAMAN               SUBTRACT A FROM MEMORY WORD
          BR        NOFIX     BRANCH IF NO BORROW
* BCD CORRECTION REQUIRED IF BORROW
*
          A10AAC              ADD 10
          TAMZA               STORE A IN MEMORY, ZERO A
          INA                 INCREMENT A
CONTU     .                   CONTINUE
*         .
NOFIX     TAM                 TRANSFER A INTO MEMORY
          CLA                 CLEAR A
          BR        CONTU
          .
          .
```



After execution:
A=0 NO BORROW
A=1 BORROW

1. BCD CORRECTION NOT NECESSARY

	X	Y	A	M(XY)	S	BRANCH
	0	2	5	6	1	
SAMAN	0	2	1	6	1	
BR	0	2	1	6	1	YES
TAMZA	0	2	1	1	1	
	0	2	0	1	1	
BR	0	2	0	1	1	YES

2. BCD CORRECTION NECESSARY

	X	Y	A	M(XY)	S	BRANCH
	2	5	7	3	1	
SAMAN	2	5	C	3	1	
BR	2	5	C	3	0	NO
A10AAC	2	5	6	3	1	
TAMZA	2	5	6	6	1	
	2	5	0	6	1	
IA	2	5	1	6	1	

4-5 ARITHMETIC COMPARE INSTRUCTIONS.

Arithmetic compares are performed by the adder using two's complement addition. The contents of the accumulator are subtracted from the value it is being compared to. The carry bit is transferred to status. The only condition that will generate a carry is the less than or equal condition. No data is destroyed by the compare instructions.

4-5.1 IF ACCUMULATOR IS LESS THAN OR EQUAL TO MEMORY, ONE TO STATUS.

0	1	2	3	4	5	6	7
0	0	1	0	1	0	0	1

MNEMONIC: ALEM

STATUS: Carry into status

FORMAT: IV

ACTION: $A \leq M(X,Y)$?
$1 \rightarrow S$ if $A \leq M(X,Y)$
$0 \rightarrow S$ if $A > M(X,Y)$

DESCRIPTION: The value from the accumulator is subtracted from the contents of the memory location, addressed by the X and Y registers, using two's complement addition. Resulting carry information is transferred into status. Status equal to ONE indicates that the accumulator is less than or equal to the memory word. Memory and accumulator contents are unaltered.

MICROINSTRUCTIONS: MTP, NATN, CIN, C8

EXAMPLE: Assume accumulator contains a 5 and M(X,Y) contains a 6.

	0 1 1 0	M(X,Y)
	1 0 1 0	$+\overline{A}$ } $-\lvert A\rvert = -5$
	1	+1 }
CARRY = 1	0 0 0 1	RESULTS NOT STORED

4-5.2 IF ACCUMULATOR IS LESS THAN OR EQUAL TO CONSTANT, ONE TO STATUS.

0	1	2	3	4	5	6	7
0	1	1	1	C			
				LSB			MSB

MNEMONIC: ALEC

STATUS: Carry into status

FORMAT: II

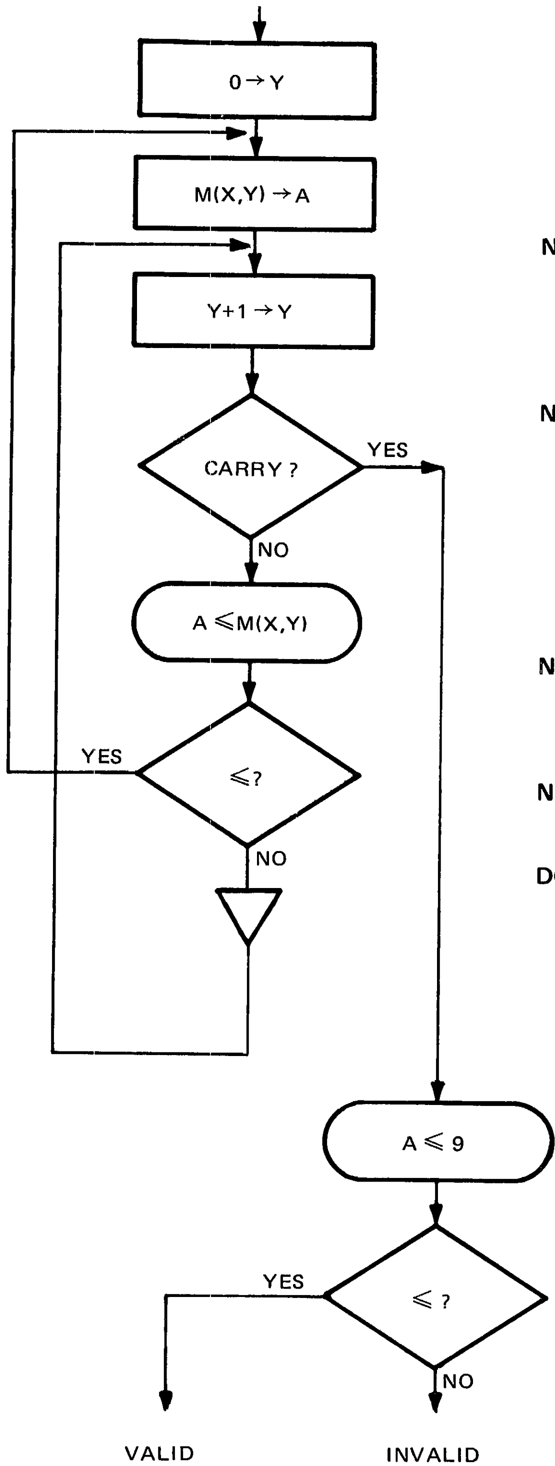
OPERAND: Constant value $0 \leq I(C) \leq 15$

ACTION:	A ≤ I(C)? 1 → S if A ≤ I(C) 0 → S if A > I(C)
PURPOSE:	To arithmetically compare accumulator contents to a constant value.
DESCRIPTION:	The accumulator value is subtracted from the constant (in the C field of the instruction) using two's complement addition. Resulting carry information is transferred into status. Status is set if the accumulator is less than or equal to the constant. The accumulator data is unaltered.
MICROINSTRUCTIONS:	CKP, NATN, CIN, C8

4-5.3 ARITHMETIC COMPARE EXAMPLE. The following example illustrates the arithmetic compare instructions.

This example performs a search of one complete RAM file, searching the 16 words of the file for the largest value. The current maximum value is maintained in the accumulator. At the end of the search, the maximum value found is tested for being a valid BCD number in the range of 0 to 9.

LABEL	OP CODE	OPERAND	COMMENT
	TCY	0	START AT Y = O
NEWMAX	TMA		LOAD NEW HIGH VALUE
NOCHNG	IYC		INCREMENT TO NEXT COL.
	BR	DONE	DONE OR CARRY
	ALEM		CURRENT MAX L.E. MEMORY DIGIT?
	BR	NEWMAX	YES, GO SET CURRENT AS MAX
	BR	NOCHNG	NO, GO CHECK NEXT DIGIT
DONE	ALEC	9	IS MAX L.E. 9?
	BR	VALID	YES
	BR	INVALID	NO



		X	Y	A	M(X,Y)	S	BR
		1	5	6	9	1	
	TCY 0	1	0	6	1	1	
	TMA	1	0	1	1	1	
NOCHNG	IYC	1	1	1	9	1	
	BR DONE	1	1	1	9	0	NO
	ALEM	1	1	1	9	1	
	BR NEWMAX	1	1	1	9	1	YES
NEWMAX	TMA	1	1	9	9	1	
	IYC	1	2	9	5	1	
	BR DONE	1	2	9	5	0	NO
	ALEM	1	2	9	5	1	
	BR NEWMAX	1	2	9	5	0	NO
	BR NOCHNG	1	2	9	5	1	YES
NOCHNG	IYC	1	3	9	4	1	
	*	*	CONTINUE			*	*
	BR NOCHNG	1	15	9	3	1	YES
NOCHNG	IYC	1	0	9	1	1	
	BR DONE	1	0	9	1	1	YES
DONE	ALEC 9	1	0	9	1	1	
	BR VALID	1	0	9	1	1	YES

4-6 LOGICAL COMPARE INSTRUCTIONS.

Logical compare instructions allow two values to be compared for equality. Operands may be register values, constants or memory words. The ALU compares P-input to the N-input. If equal, the ALU transmits a ZERO to status. The status may then be tested by a conditional instruction immediately following the compare. No data is destroyed by the logical compare.

4-6.1 IF MEMORY IS NOT EQUAL TO ZERO, ONE TO STATUS.

0	1	2	3	4	5	6	7
0	0	1	0	0	1	1	0

MNEMONIC: MNEZ

STATUS: Comparison result into status

FORMAT: IV

ACTION:
$M(X,Y) \neq 0?$
$1 \rightarrow S$ if $M(X,Y) \neq 0$
$0 \rightarrow S$ if $M(X,Y) = 0$

PURPOSE: To compare a memory word to zero.

DESCRIPTION: The memory contents addressed by the X and Y register are logically compared to zero. Comparison information is transferred into status. Inequality between memory value and zero will set status.

MICROINSTRUCTIONS: MTP, NE.

4-6.2 IF Y REGISTER IS NOT EQUAL TO ACCUMULATOR, ONE TO STATUS.

0	1	2	3	4	5	6	7
0	0	0	0	0	0	1	0

MNEMONIC: YNEA

STATUS: Comparison result into status

FORMAT: IV

ACTION:
$Y \neq A?$
$1 \rightarrow S$ and $1 \rightarrow SL$ if $Y \neq A$
$0 \rightarrow S$ and $0 \rightarrow SL$ if $Y = A$

PURPOSE: To compare the contents of the Y register and the accumulator for inequality, and to preset the status latch for buffering to the O-output register.

DESCRIPTION: The contents of the Y register are logically compared to the contents of the accumulator. Comparison information is transferred into status. Inequality will set status. Status also transfers into the status latch to be made available for a future data output instruction (TDO).

MICROINSTRUCTIONS: YTP, ATN, NE, STSL.

4-6.3 IF Y REGISTER IS NOT EQUAL TO A CONSTANT, ONE TO STATUS.

0	1	2	3	4	5	6	7
0	1	0	1	C			
				LSB			MSB

MNEMONIC: YNEC

STATUS: Comparison result into status

FORMAT: II

OPERAND: Constant, $0 \leqslant I(C) \leqslant 15$

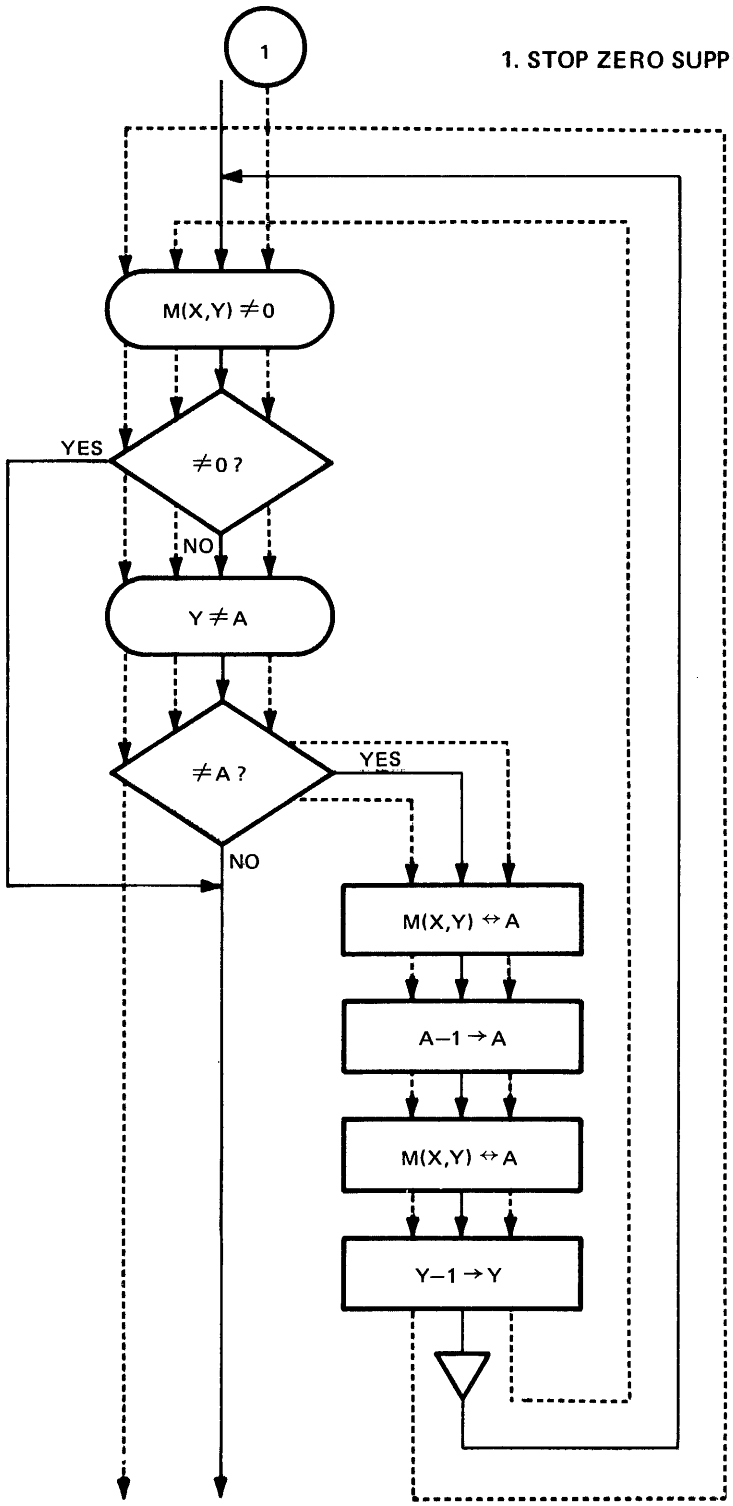
ACTION: $Y \neq I(C)$?
$1 \rightarrow S$ if $Y \neq C$
$0 \rightarrow S$ if $Y = C$

DESCRIPTION: The contents of the Y register are logically compared to the four-bit value from the C field of the instruction. Compare result is transferred into status. Inequality between the operands causes status to be set (ONE).

MICROINSTRUCTIONS: YTP, CKN, NE

4-6.4 LOGICAL COMPARE EXAMPLE. The following example illustrates the logical compare instructions. This example shows formatting the display of a floating point multidigit BCD number stored in RAM memory. The LSD is at Y address 0. The Y register sequentially addresses a 16 word file, starting with the most significant digit position (the MSD is at Y address 1 to 15). The accumulator contains the position of the implied decimal point. Zero suppression stops when the first non-zero digit is found or when the decimal point position is reached. Zeros are suppressed by replacing them with a blank code digit (F_{16}) which is obtained by subtracting one from the zero.

```
LABEL    OP CODE   OPERAND   COMMENT

LOOP     MNEZ                DIGIT NOT EQUAL TO ZERO?
         BR        DONE      YES, EXIT TO DONE
         YNEA                Y INDEX NOT EQUAL TO A (DECIMAL POINT)?
         BR        SUPRES    NO, CONTINUE TO SUPPRESSION
DONE     .                   YES, DONE
         .
         .
SUPRES   XMA                 EXCHANGE MEMORY AND A
         DAN                 DECREMENT A (PRODUCE F)
         XMA                 EXCHANGE M AND A
         DYN                 DECREMENT Y INDEX
         BR        LOOP      LOOP TO TEST NEXT DIGIT
```

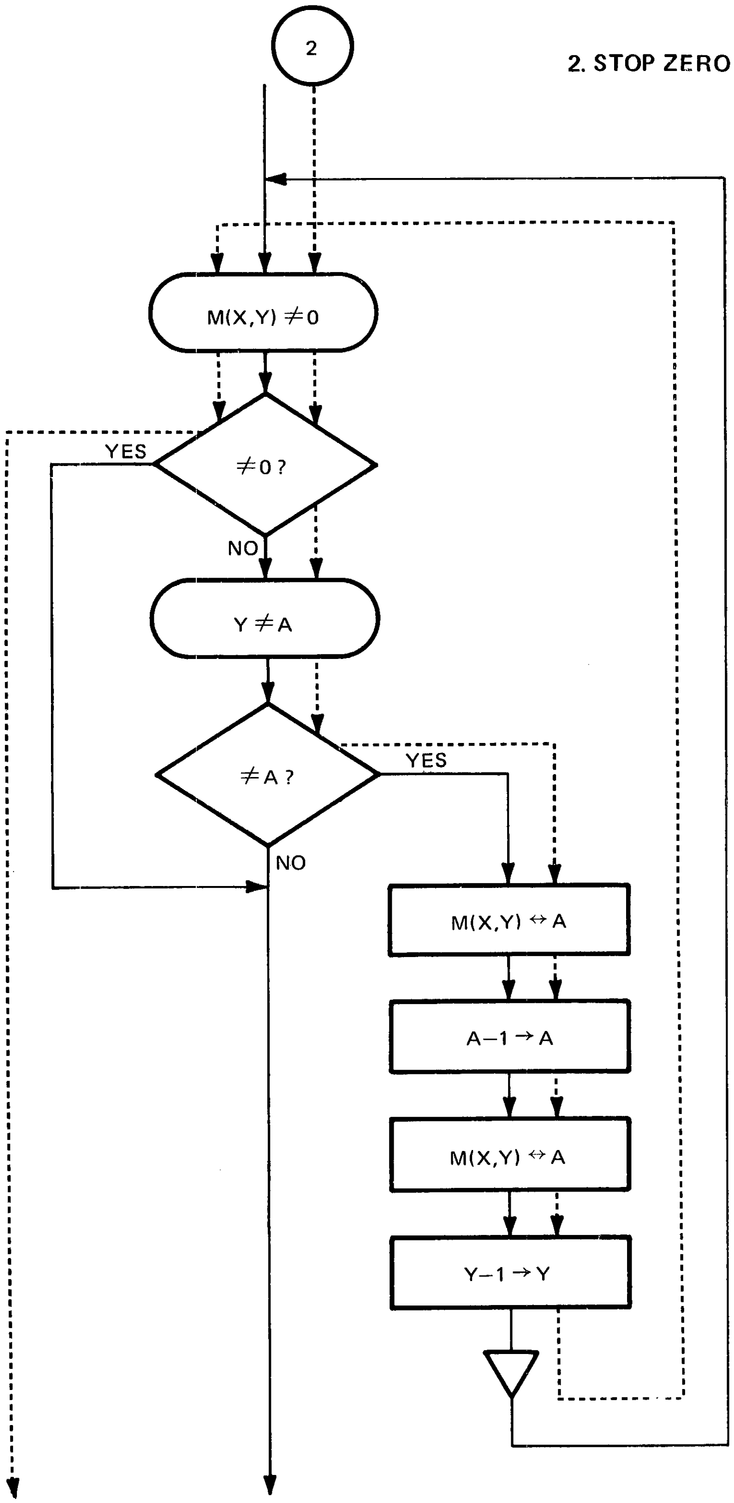


1. STOP ZERO SUPPRESSION BECAUSE OF A LEADING DECIMAL POINT

	X	Y	A	M(XY)	S	BRANCH
MNEZ	0	4	2	0	1	
BR	0	4	2	0	0	NO
YNEA	0	4	2	0	1	
BR	0	4	2	0	1	YES
XMA	0	4	0	2	1	
DAN	0	4	F	2	1	
XMA	0	4	2	F	0	
DYN	0	3	2	0	1	
BR	0	3	2	0	1	YES
MNEZ	0	3	2	0	1	
BR	0	3	2	0	0	NO
YNEA	0	3	2	0	1	
BR	0	3	2	0	1	YES
XMA	0	3	0	2	1	
DAN	0	3	F	2	1	
XMA	0	3	2	F	0	
DYN	0	2	2	0	1	
BR	0	2	2	0	1	YES
MNEZ	0	2	2	0	1	
BR	0	2	2	0	0	NO
YNEA	0	2	2	0	1	
BR	0	2	2	0	0	NO

CONTENTS SUMMARY:

X	0	0	0	0	0
Y	4	3	2	1	0
M(X,Y) BEFORE	0	0	0	1	2
M(X,Y) AFTER	F	F	0	1	2



2. STOP ZERO SUPPRESSION BECAUSE OF A NON-ZERO INTEGER

	X	Y	A	M(X,Y)	S	BRANCH
MNEZ	0	4	2	0	1	
BR	0	4	2	0	0	NO
YNEA	0	4	2	0	1	
BR	0	4	2	0	1	YES
XMA	0	4	0	2	1	
DAN	0	4	F	2	1	
XMA	0	4	2	F	0	
DYN	0	3	2	8	1	
BR	0	3	2	8	1	YES
MNEZ	0	3	2	8	1	
BR	0	3	2	8	1	YES

CONTENTS SUMMARY:

X	0	0	0	0	0
Y	4	3	2	1	0
M(X,Y) BEFORE	0	8	9 .	7	6
M(X,Y) AFTER	F	8	9 .	7	6

4-7 BIT MANIPULATION IN MEMORY (RAM) INSTRUCTIONS.

The bit instructions operate on an individual bit in the RAM. The selected bit may be set, reset or tested. These instructions allow a program to use bits as "flags" to maintain the on or off state of an event and to test the current state of the flag bit. The bit addresses are defined as follows:

BIT ADDRESS	B-FIELD I(6)	B-FIELD I(7)	RAM WORD MSB			RAM WORD LSB
0	0	0				X
1	1	0			X	
2	0	1		X		
3	1	1	X			

4-7.1 SET MEMORY (RAM) BIT.

0	1	2	3	4	5	6	7
0	0	1	1	0	0	B	
						LSB	MSB

MNEMONIC: SBIT

STATUS: Set

FORMAT: III

OPERAND: Bit address, $0 \leqslant I(B) \leqslant 3$

ACTION: $1 \rightarrow M(X,Y,B)$

DESCRIPTION: One of the four bits, as selected by the B-field of the operand, is set to a logic ONE in the RAM memory word addressed by the contents of the X and Y registers.

FIXED MICROINSTRUCTION: SBIT

4-7.2 RESET MEMORY (RAM) BIT.

0	1	2	3	4	5	6	7
0	0	1	1	0	1	B	
						LSB	MSB

MNEMONIC: RBIT

STATUS: Set

FORMAT: III

OPERAND: Bit address, $0 \leqslant I(B) \leqslant 3$

ACTION: $0 \rightarrow M(X,Y,B)$

DESCRIPTION: One of the four bits, as selected by the B-field of the instruction, is reset to a logic ZERO in the RAM memory word addressed by the contents of the X and Y registers.

FIXED MICROINSTRUCTION: RBIT

4-7.3 TEST MEMORY (RAM) BIT FOR ONE.

0	1	2	3	4	5	6	7
0	0	1	1	1	0	B	
						LSB	MSB

MNEMONIC: TBIT1

STATUS: Comparison result into status

FORMAT: III

OPERAND: Bit Address, $0 \leqslant I(B) \leqslant 3$

ACTION: $M(X,Y,B) = 1?$
$1 \rightarrow S$ if $M(X,Y,B) = 1$
$0 \rightarrow S$ if $M(X,Y,B) = 0$

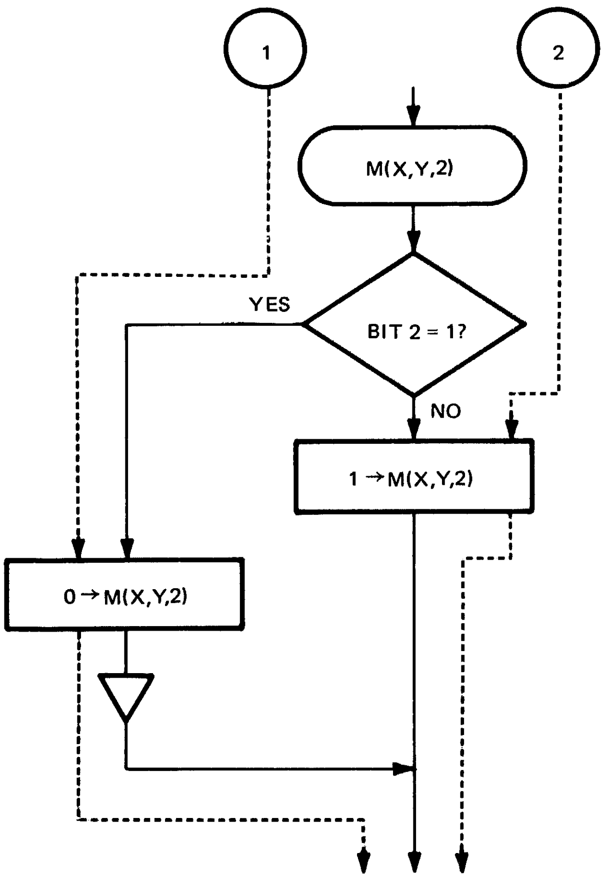
PURPOSE: To test a selected memory bit for a logic ONE and set status accordingly.

DESCRIPTION: The CKI logic generates a four-bit mask which goes to both P and N adder inputs (CKP and CKN microinstructions). The bit mask has a ZERO (opening) selected by the B field of the instruction word (shown in Table 2-7.1). The RAM word, selected by registers X and Y, is logically ORed with the bit mask word by the MTP microinstruction combined with the CKP microinstruction. The NE microinstruction sends to status the comparison information caused by logically comparing the unmasked bit from RAM with a ZERO (the opening in the bit mask input to the N side of the ALU).

MICROINSTRUCTIONS: CKP, CKN, MTP, NE

4-7.4 BIT MANIPULATION EXAMPLE. The following example illustrates the three memory bit instructions. This routine will "flip" the state of bit 2 in RAM word M(X,Y).

```
LABEL      OP CODE    OPERAND    COMMENTS

           .
           .
           TBIT1      2          IS BIT 2 ON ?
           BR         SETOFF     YES, BRANCH
           SBIT       2          NO, SET IT ON
           BR         CONTU
SETOFF     RBIT       2          BIT OFF
           .
```



1. BIT 2 = 1

	X	Y	A	M(X,Y)	S	BR
	2	F	3	7	1	
TBIT1 2	2	F	3	[7]	1	
BR	2	F	3	7	1	YES
RBIT 2	2	F	3	3	1	
BR	2	F	3	3	1	YES

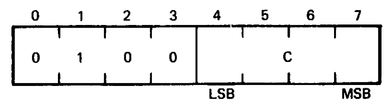
2. BIT 2 = 0

	X	Y	A	M(X,Y)	S	BR
	3	1	5	9	1	
TBIT1 2	3	1	5	[9]	1	
BR	3	1	5	9	0	NO
SBIT 2	3	1	5	D	1	

4-8 CONSTANT TRANSFER INSTRUCTIONS.

Most programs need constant values to preset counters for loop control, to set RAM constants, or to set a register to a RAM address. For the following instructions, constants from the C-field of the instruction are transferred into memory or the registers.

4-8.1 TRANSFER CONSTANT TO Y REGISTER.



MNEMONIC: TCY

STATUS: Set

FORMAT: II

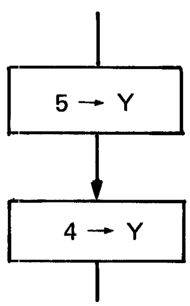
OPERAND: Constant, $0 \leqslant I(C) \leqslant 15$

ACTION: $I(C) \rightarrow Y$

PURPOSE: To load the Y register with a constant. Common uses are to set Y to a particular RAM word address, address a selected R(Y) output line or to initialize Y for loop control.

DESCRIPTION: The four-bit value from the C-field of the instruction is transferred into the Y register.

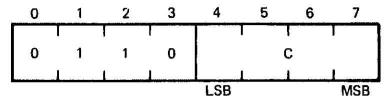
MICROINSTRUCTIONS: CKP, AUTY



	X	Y	A	M(X,Y)
	0	F	3	A
TCY 5	0	5	3	2
TCY 4	0	4	3	B

Note: M(X,Y) appears to change because the pointer (Y) is changed.

4-8.2 TRANSFER CONSTANT TO MEMORY AND INCREMENT Y REGISTER.



MNEMONIC: TCMIY

STATUS: Set

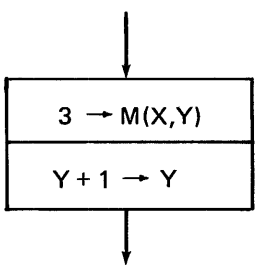
FORMAT: II

OPERAND: Constant, $0 \leqslant I(C) \leqslant 15$

ACTIONS: $I(C) \rightarrow M(X,Y)$
$Y + 1 \rightarrow Y$

DESCRIPTION: The four-bit value from the C-field of the instruction is stored in the memory location addressed by the X and Y registers. The Y register contents are then incremented by one.

MICROINSTRUCTIONS: CKM, YTP, CIN, AUTY.



		X	Y	A	M(X,Y)
		3	A	6	2
TCMIY	3	3	A	6	3
		3	B	6	9

4-9 INPUT INSTRUCTIONS.

The input instructions are used to bring external data from the four input (K) lines into the microcomputer. This data transfers into the registers or memory for manipulation or storage. No operands are used.

4-9.1 IF K INPUTS ARE NOT EQUAL TO ZERO, SET STATUS.

0	1	2	3	4	5	6	7
0	0	0	0	1	0	0	1

MNEMONIC: KNEZ

STATUS: Comparison result into status

FORMAT: IV

ACTION: $K_{8, 4, 2, 1} \neq 0$?
$1 \rightarrow S$ if $K \neq 0$
$0 \rightarrow S$ if $K = 0$

PURPOSE: To test the four K-input lines for a non-ZERO state. This instruction is useful for monitoring a keyboard for a "key down" condition.

DESCRIPTION: Data on the four external K-input lines are compared to zero. Comparison information is transferred into status. Non-ZERO data inputs cause status to be set (to ONE).

MICROINSTRUCTIONS: CKP, NE.

4-9.2 TRANSFER K-INPUTS TO ACCUMULATOR.

0	1	2	3	4	5	6	7
0	0	0	0	1	0	0	0

MNEMONIC: TKA

STATUS: Set

FORMAT: IV

ACTION: $K_{8, 4, 2, 1} \rightarrow A$

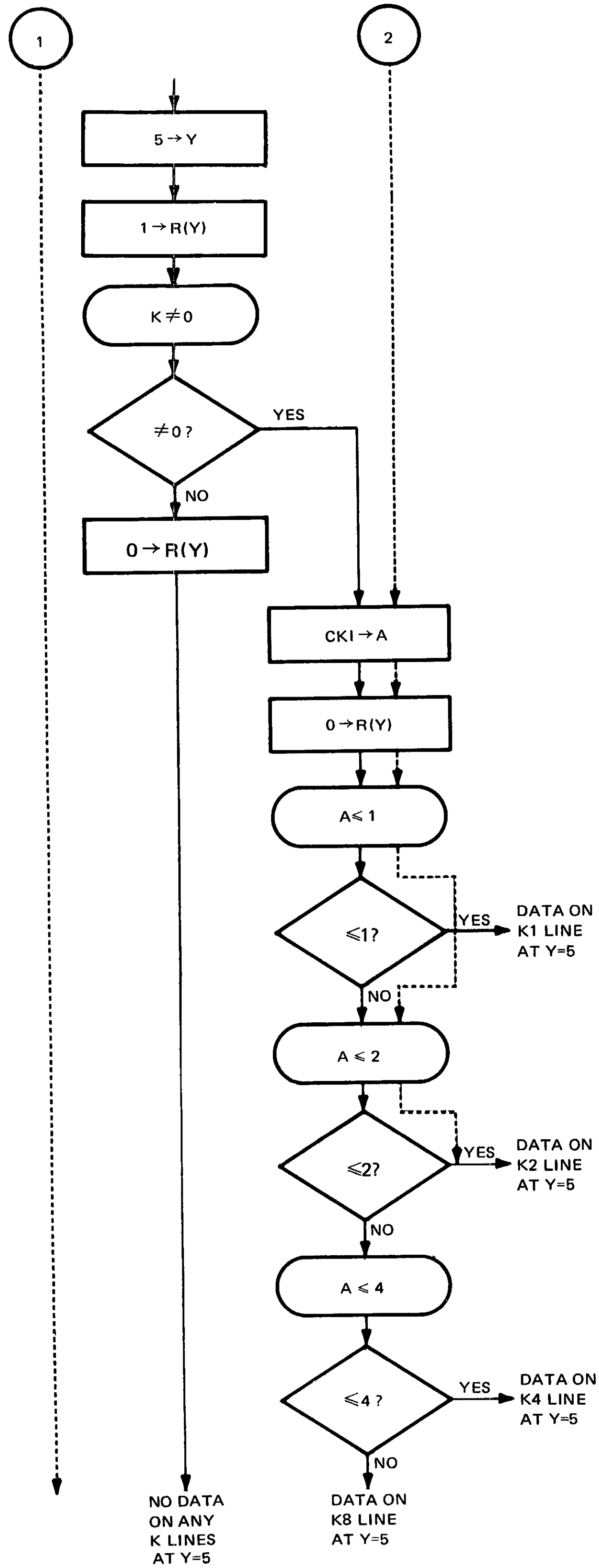
PURPOSE: To transfer the external input data into the accumulator for processing.

DESCRIPTION: Data present on the four external K-input lines is transferred into the accumulator.

MICROINSTRUCTIONS: CKP, AUTA

4-9.3 INPUT EXAMPLE. The following example illustrates the input instructions. This example handles input from a keyboard. The keys must be sampled one row at a time. The particular row selected is determined by which R-output line being set on. This example shows sampling on row five only, and determines which of four keys on row five are depressed. If all four K inputs are zero, no key is currently depressed. For simplicity no key-debounce logic has been included.

```
LABEL       OP CODE     OPERAND     COMMENT

            TCY         5           SET ROW 5
            SETR                    ENABLE ROW 5
            KNEZ                    TEST K INPUTS FOR NON-ZERO
            BR          INPUT       YES, GO TO INPUT
* NO DATA PRESENT ON INPUT LINES
            RSTR                    DISABLE ROW 5
            BR          CONTU       EXIT
*
* NOW STORE THE DATA FROM THE K LINES.
*
INPUT       TKA                     INPUT K LINES TO A
            RSTR                    DISABLE ROW 5
*
*
* NOW FIND WHICH KEY ON ROW 5.
*
            ALEC        1           KEY 1?
            BR          ONK1        YES
            ALEC        2           KEY 2?
            BR          ONK2        YES
            ALEC        4           KEY 4?
            BR          ONK4        YES
            BR          ONK8        MUST BE ON K8.
```



1. K = 0

	X	Y	A	S	R(Y)	BRANCH
	0	B	2	1	0	
TCY 5	0	5	2	1	0	
SETR	0	5	2	1	1	
KNEZ	0	5	2	1	1	
BR	0	5	2	0	1	NO
RSTR	0	5	2	1	0	

2. K ≠ 0

	X	Y	A	S	R(Y)	BRANCH
	1	7	4	1	0	
TCY 5	0	5	4	1	0	
SETR	0	5	4	1	1	
BR	0	5	4	1	1	
KNEZ	0	5	4	1	1	
TKA	0	5	2	1	1	
RSTR	0	5	2	1	0	
ALEC 1	0	5	2	1	0	
BR	0	5	2	0	0	NO
ALEC 2	0	5	2	1	0	
BR	0	5	2	1	0	YES

4-10 OUTPUT INSTRUCTIONS.

Output instructions make internal data available to external devices. Two types of output exist, individual or group.

The 13 R-output lines are controlled individually. The O-outputs go out as an eight-bit group. The R output lines are normally used to multiplex K input data, strobe O-output data, or to control individual output signals.

No operands are used.

4-10.1 SET R OUTPUT.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	0	1

MNEMONIC: SETR

STATUS: Set

FORMAT: IV

ACTION: $1 \rightarrow R(Y)$
For $0 \leqslant Y \leqslant 12$ TMS 1200
For $0 \leqslant Y \leqslant 10$ TMS 1000

PURPOSE: To set one R output line to a logic ONE.

DESCRIPTION: The contents of the Y register selects the proper R output. The content of the Y register is between 0 through 12 inclusive, to select the R output to be set. For values greater than 12 in the Y register, the instruction is a no-operation.

FIXED MICROINSTRUCTION: SETR

4-10.2 RESET R OUTPUT.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	0	0

MNEMONIC: RSTR

STATUS: Set

FORMAT: IV

ACTION: $0 \rightarrow R(Y)$
For $0 \leqslant Y \leqslant 12$ TMS 1200
For $0 \leqslant Y \leqslant 10$ TMS 1000

PURPOSE: To reset one R-output line to a logic ZERO.

DESCRIPTION: The contents of the Y register select the proper R output (R0 to R12). The contents of the Y register are between 0 and 12 inclusive to select the R output to be reset. For values greater than 12 in the Y register, the instruction is a no-operation.

FIXED MICROINSTRUCTION: RSTR

4-10.3 TRANSFER DATA FROM ACCUMULATOR AND STATUS LATCH TO O-OUTPUT REGISTER.

MNEMONIC: TDO

0	1	2	3	4	5	6	7
0	0	0	0	1	0	1	0

STATUS: Set

FORMAT: IV

ACTION: SL → O-output Register
A → O-output Register

DESCRIPTION: The contents of the accumulator and the status latch are transferred to the O-output register. The O-register data is decoded by the output PLA depending upon how the user programmed the output PLA. The output PLA translates the five-bit code into an eight-bit value present on the eight parallel lines.

FIXED MICROINSTRUCTION: TDO

4-10.4 CLEAR OUTPUT REGISTER.

MNEMONIC: CLO

0	1	2	3	4	5	6	7
0	0	0	0	1	0	1	1

STATUS: Set

FORMAT: IV

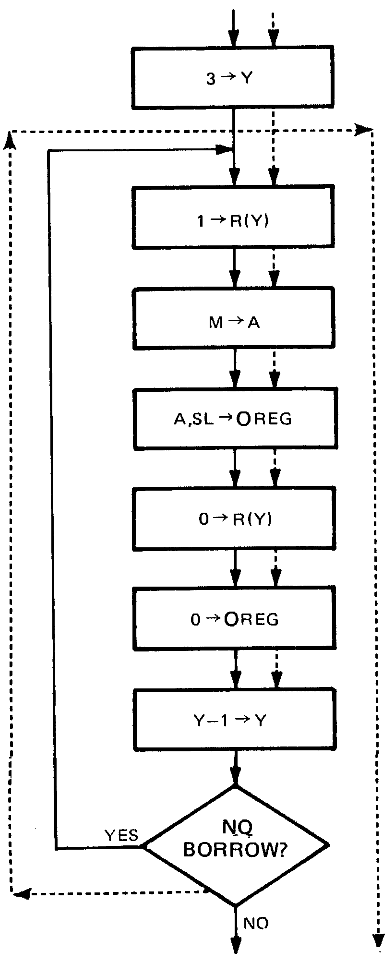
ACTION: 0 → O-Register

DESCRIPTION: The O-register contents are cleared to zero. It is important to remember that the output PLA may be defined by the user to translate this value into any eight-bit value desired.

FIXED MICROINSTRUCTION: CLO

4-10.5 OUTPUT SAMPLE. The following example illustrates the various output instructions. Four data words from memory, M(0,3) through M(0,0), go to the O-output register. The R-outputs are used to signal which word is presented. The O-register is cleared after each word has been presented. The example assumes that a previous YNEA instruction set the status latch to ZERO.

```
LABEL     OP CODE   OPERAND   COMMENT

          .
          .
          TCY       3         SET INDEX AND COUNTER
LOOP      SETR                SET R(Y) OUTPUT STROBE
          TMA                 LOAD DIGIT INTO A
          TDO                 LOAD OUTPUT FROM A AND SL
          RSTR                RESET R(Y) OUTPUT STROBE
          CLO                 CLEAR O OUTPUT REGISTER
          DYN                 DECREMENT Y REGISTER
          BR        LOOP      LOOP UNTIL Y BORROWS
          .
          .
```



		X	Y	A	M(X,Y)	S	O	R(Y)	BRANCH
	INITIAL	0	7	1	4	1	0	0	
	TCY 3	0	3	1	F	1	0	0	
LOOP	SETR	0	3	1	F	1	0	1	
	TMA	0	3	F	F	1	0	1	
	TDO	0	3	F	F	1	F	1	
	RSTR	0	3	F	F	1	F	0	
	CLO	0	3	F	F	1	0	0	
	DYN	0	2	F	A	1	0	0	
	BR LOOP	0	2	F	A	1	0	0	YES
LOOP	SETR	0	2	F	A	1	0	1	
	TMA	0	2	A	A	1	0	1	
	TDO	0	2	A	A	1	A	1	
	RSTR	0	2	A	A	1	A	0	
	CLO	0	2	A	A	1	0	0	
	DYN	0	1	A	5	1	0	0	
	BR LOOP	0	1	A	5	1	0	0	YES
LOOP	SETR	0	1	A	5	1	0	1	
	TMA	0	1	5	5	1	0	1	
	TDO	0	1	5	5	1	5	1	
	RSTR	0	1	5	5	1	5	0	
	CLO	0	1	5	5	1	0	0	
	DYN	0	0	5	9	1	0	0	
	BR LOOP	0	0	5	9	1	0	0	YES
LOOP	SETR	0	0	5	9	1	0	1	
	TMA	0	0	9	9	1	0	1	
	TDO	0	0	9	9	1	9	1	
	RSTR	0	0	9	9	1	9	0	
	CLO	0	0	9	9	1	0	0	
	DYN	0	F	9	B	1	0	0	
	BR LOOP	0	F	9	B	0	0	0	NO

4-11 RAM-X ADDRESSING INSTRUCTIONS.

Two instructions are provided to control the RAM-X file addressing. The instructions can load an absolute address in the X register or can complement the value in the X register to flip between RAM file addresses.

4-11.1 LOAD X REGISTER WITH A CONSTANT.

0	1	2	3	4	5	6	7
0	0	1	1	1	1	B	
						LSB	MSB

MNEMONIC: LDX

STATUS: Set

FORMAT: III

OPERAND: X file address; $0 \leqslant X \leqslant 3$

ACTION: $I(B) \rightarrow X$

DESCRIPTION: A constant value is loaded into the X-register. This is used to set the X register to the desired RAM file index. The two-bit B-field of the instruction is loaded into the X-register.

FIXED MICROINSTRUCTION: LDX

4-11.2 COMPLEMENT X REGISTER.

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

MNEMONIC: COMX

STATUS: Set

FORMAT: IV

ACTION: $\overline{X} \rightarrow X$

DESCRIPTION: The contents of the X register are logically complemented (one's complement):

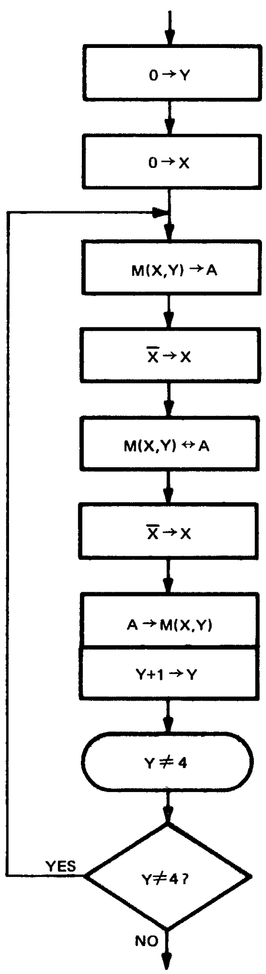
(0) 00 → 11 (3)
(1) 01 → 10 (2)
(2) 10 → 01 (1)
(3) 11 → 00 (0)

FIXED MICROINSTRUCTION: COMX

4-11.3 RAM-X ADDRESSING EXAMPLE. The following example illustrates using the LDX and COMX instructions that effect the X register.

This example exchanges the first four digits of RAM file 0 with RAM file 3. The exchange is performed using the complement X instruction.

LABEL	OP CODE	OPERAND	COMMENT
	TCY	0	SET RAM ADDRESS
	LDX	0	to M (0,0).
LOOP	TMA		FETCH FROM FILE 0
	COMX		COMPLEMENT X, FLIP TO FILE 3
	XMA		EXCHANGE M AND A
	COMX		FLIP TO FILE 0
	TAMIY		STORE ACCUMULATOR AND INCREMENT Y
	YNEC	4	DONE?
	BR	LOOP	LOOP UNTIL Y = 3



	X	Y	A	M(X,Y)	S	BRANCH
	2	5	8	0	1	
TCY 0	2	0	8	A	1	
LDX 0	0	0	8	3	1	
TMA	0	0	3	3	1	
COMX	3	0	3	6	1	
XMA	3	0	6	3	1	
COMX	0	0	6	3	1	
TAMIY	0	0	6	6	1	
	0	1	6	2	1	
YNEC 4	0	1	6	2	1	
BR	0	1	6	2	1	YES
TMA	0	1	2	2	1	
COMX	3	1	2	5	1	
XMA	3	1	5	2	1	
COMX	0	1	5	2	1	
TAMIY	0	1	5	5	1	
	0	2	5	1	1	
YNEC 4	0	2	5	1	1	
BR	0	2	5	1	1	YES
TMA	0	2	1	1	1	
COMX	3	2	1	4	1	
XMA	3	2	4	1	1	
COMX	0	2	4	1	1	
TAMIY	0	2	4	4	1	
	0	3	4	0	1	
YNEC 4	0	3	4	0	1	
BR	0	3	4	0	1	YES

ONE MORE LOOP UNTIL Y = 4.

RAM CONTENTS BEFORE EXAMPLE EXECUTION:

Y =	3 2 1 0
X = 0	0 1 2 3
X = 3	0 4 5 6

RAM CONTENTS AFTER EXAMPLE EXECUTION:

Y =	3 2 1 0
X = 0	0 4 5 6
X = 3	0 1 2 3

4-12 ROM ADDRESSING INSTRUCTIONS.

The following set of instructions controls the program execution sequence. The ROM program normally executes sequentially within a ROM page until altered by a ROM addressing instructions.

These instructions alter the contents of the program counter (PC), subroutine return register (SR), page address register (PA) or the page buffer registers (PB) as required to perform branching, subroutine calls, and returns.

Branch and call instructions are always conditional on status. Status performs the function of a switch. Status set (ONE) will enable, and status reset (ZERO) will disable the conditional jump of the instruction. The state of the status is dependent upon the results of the previously executed instruction. The normal state of status is to be a ONE (set), and it will always return to this state after one instruction cycle if the next instruction does not reset status (ZERO).

4-12.1 BRANCH, CONDITIONAL ON STATUS.

0	1	2	3	4	5	6	7
1	0	W					
		MSB					LSB

MNEMONIC: BR

STATUS: Conditional on status

FORMAT: I

OPERAND: Branch Address I(W)

ACTION:

If S = 1 and CL = 0:
I(W) → PC
PB → PA

If S = 1 and CL = 1:
I(W) → PC

If S = 0:
PC + 1 → PC
1 → S

NOTE

PC points to next address in a fixed sequence which is actually a pseudo random count.

PURPOSE: To allow the program to alter the normal sequential program execution. The branch will be conditional on the status results of the previously executed instruction.

DESCRIPTION: The branch instruction is always conditional upon the state of status. If status is reset (logic ZERO), then the branch is unsuccessfully executed and the next sequential instruction will be performed.

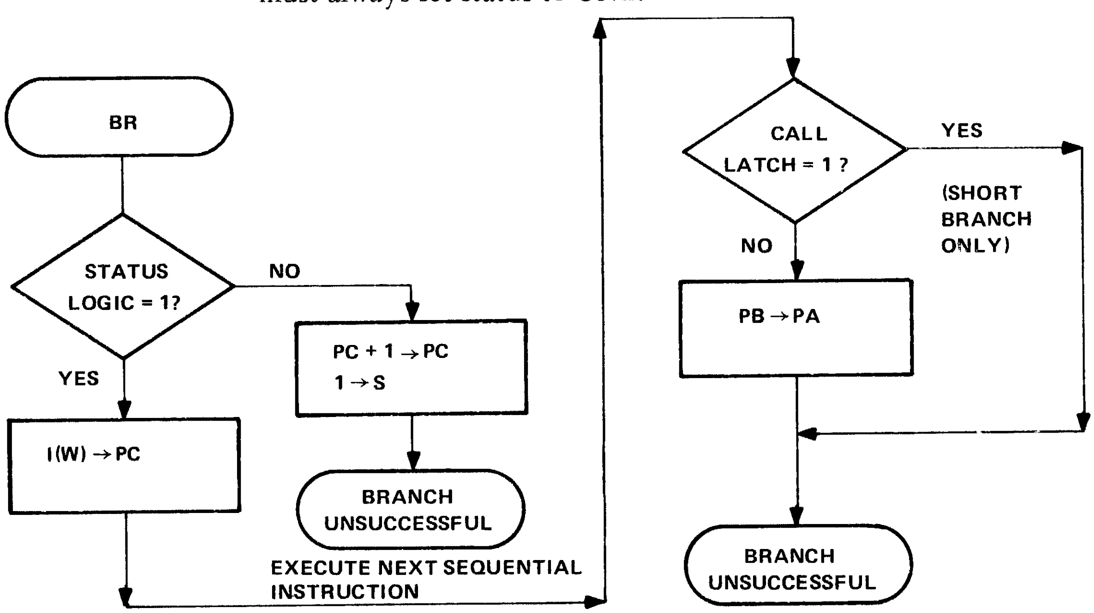
If the status is set (logic ONE), then the branch will occur by the following actions. The W-field of the instruction is loaded into the program counter (PC). The contents of the page buffer (PB) are transferred into the page address (PA) register (this transfer does not occur when the machine is in the call mode).

Branches may be of two types, short or long. Short branches address within the current page while long branches address into another ROM page. The type of branch performed is determined by the contents of the PB register. To execute a long branch, the contents of the PB register must be modified to the desired page address prior to the branch which is performed via the load PB-register (LDP) instruction. When a long branch is desired in the source program, a branch long, BL, directive causes the assembler to generate two instructions, LDP and BR.

FIXED MICROINSTRUCTION: BR

NOTE

To allow for conditional branching, the branch instruction must immediately follow the instruction that affected the status. Only that instruction immediately preceding the branch instruction determines if status is ZERO, causing the branch to be unsuccessful. If unconditional branching is desired, the preceding instruction must always set status to ONE.



SINGLE INSTRUCTION CYCLE FLOWCHART – BRANCH INSTRUCTION

4-12.2 CALL SUBROUTINE, CONDITIONAL ON STATUS.

0	1	2	3	4	5	6	7
1	1	W					
		MSB					LSB

MNEMONIC: CALL

STATUS: Conditional on status

FORMAT: I

OPERAND: Subroutine word address, I(W).

ACTION:

If $S = 1$ and $CL = 0$:
$PC + 1 \rightarrow SR$
$PB \leftrightarrow PA$
$I(W) \rightarrow PC$
$1 \rightarrow CL$

If $S = 1$ and $CL = 1$:
$I(W) \rightarrow PC$
$PA \rightarrow PB$

If $S = 0$:
$PC + 1 \rightarrow PC$
$1 \rightarrow S$

NOTE

PC actually has a pseudo random count to the next instruction.

PURPOSE: To allow the program to transfer control to a common subroutine. Because the call instruction saves the return address, subroutines may be called from various locations in a program, and the subroutine will return control back to the proper, saved calling address using the call-return instruction, RETN.

DESCRIPTION: Call is always conditional upon status. If status is reset, then the call is executed unsuccessfully. If the status is set, then the call is performed by the following operations.

The address of next instruction is saved in the subroutine return (SR) register. The contents of the page buffer and the page address registers are exchanged. This saves the return page address in PB and sets PA to the page address of the subroutine called. The PC is loaded with the contents of the W-field of the call instruction which is the address of the subroutine called.

The call latch (CL) is set to a logic ONE when in the call mode. This protects the return address in SR.

Long calls (call to another page) can be made by performing a LDP instruction prior to the CALL. Omitting the LDP instruction (and PA = PB) will result in a short call (call to the same page). When a long call is desired in the source program, the call long, CALLL, directive causes the assembler to generate two instructions, LDP and CALL.

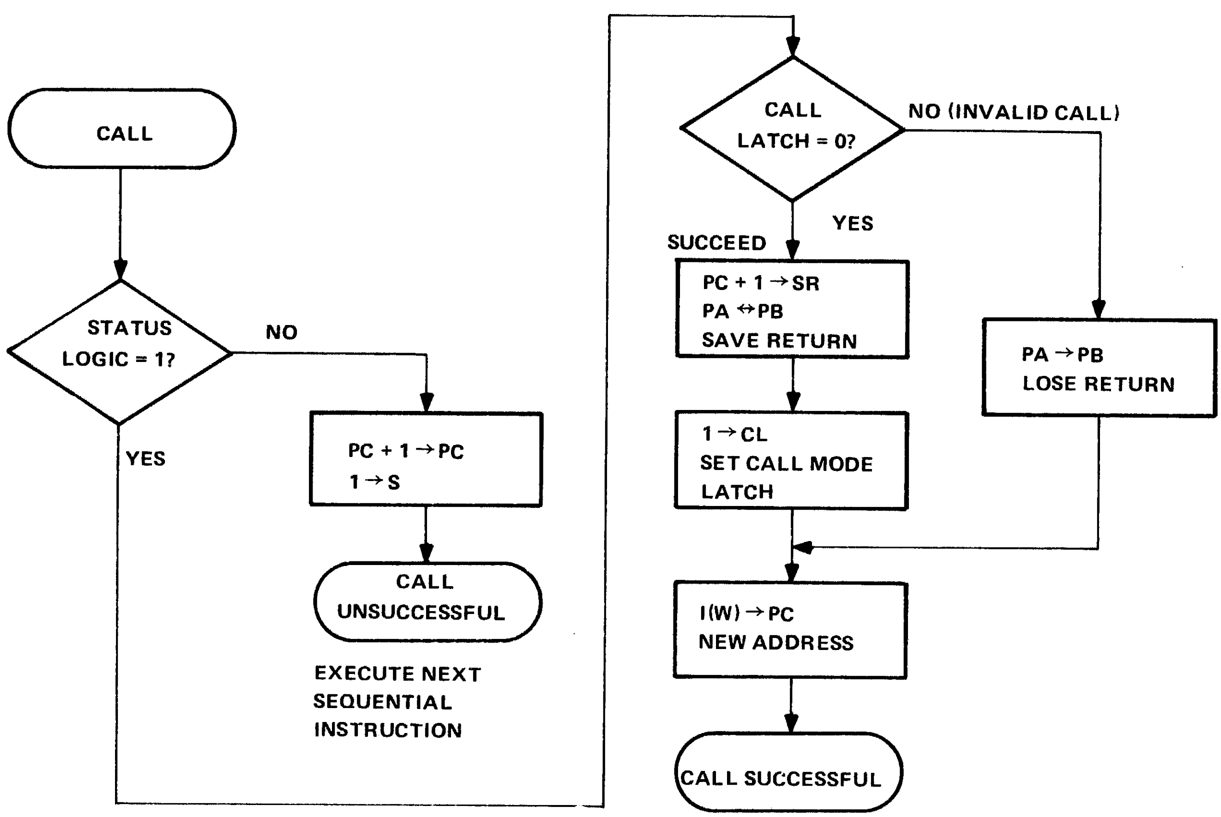
When the machine is in the call mode, it is not possible to perform long branches because the PB to PA transfer is locked out while in the call mode. A short branch is possible in the call mode.

Because only one level of return addresses may be saved, a subroutine may not call another subroutine.

NOTE

Executing a call instruction while the machine is in the call mode, CL = 1, will have the same effect as a short branch except the PA transfers to PB. The W-field of the instruction is transferred to the PC. The page address register is not altered. See paragraph 2-4 for a more detailed explanation.

FIXED MICROINSTRUCTION: CALL



SINGLE INSTRUCTION CYCLE FLOWCHART – CALL INSTRUCTION

4-12.3 RETURN FROM SUBROUTINE.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	1	1

MNEMONIC: RETN

STATUS: Set

FORMAT: IV

ACTION:

If CL = 1:
SR → PC
PB → PA
0 → CL

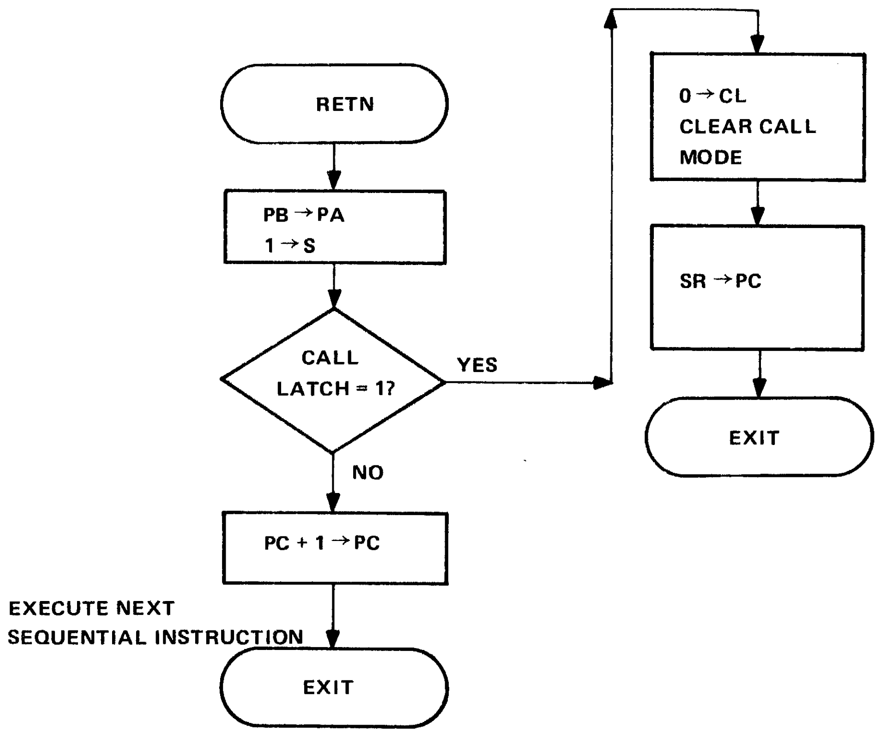
If CL = 0:
PC + 1 → PC
PB → PA

PURPOSE: To return control from a called subroutine back to the calling program.

DESCRIPTION: The return address is restored. The subroutine return (SR) register contents are transferred to the PC. Simultaneously, the contents of the PB register are transferred into the PA register. The call latch (CL) is reset, placing the machine in the normal non-call mode.

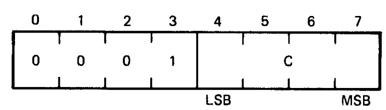
NOTE

When a return instruction is executed in the non-call mode (CL = 0), two different results may occur depending upon the contents of the page buffer (PB). If PA equals PB, (i.e., PB has not been modified by a LDP instruction), then the instruction will be a no-operation. If the PB has been altered, then control will be passed to the new page whose address is in PB. Note that when CL = 0, the PC is only incremented by one.



SINGLE INSTRUCTION CYCLE FLOWCHART – RETURN INSTRUCTION

4-12.4 LOAD PAGE BUFFER WITH A CONSTANT.



MNEMONIC: LDP

STATUS: Set

FORMAT: II

OPERAND: ROM page address: $0 \leqslant C \leqslant 15$

ACTION: $I(C) \rightarrow PB$

PURPOSE: To load the page buffer (PB) register with a new ROM page address. This is necessary for performing long branch or call instructions.

DESCRIPTION: The PB register is loaded with the four-bit value from the C-field of the instruction.

FIXED MICROINSTRUCTION: LDP

4-12.5 PROGRAM CONTROL EXAMPLE 1. The following example illustrates the usage of the program control instructions BR, CALL, RETN and LDP.

This example illustrates using a control loop that calls a subroutine to perform a specific function. The control loop continues to call the subroutine until certain conditions are met, then control is passed to another portion of the main program in a different ROM page. This particular example calls a 'shift left' routine to shift a five word string left one word address at a time. The shift routine is called until a non-zero word is found in position M(0,3). Because the subroutine is in another page, a long call is performed by setting a new page address in the page buffer (PB) before the call.

Page 3

```
LABEL       OP CODE    OPERAND    COMMENT

            LDX        0          SET RAM ADDRESS
LOOP        TCY        3            to M(0, 3)
            MNEZ                  M(0, 3) ≠ 0;
            BR         DONE       BRANCH IF NOT EQUAL, DONE
*
* SET UP TO CALL SHIFT LEFT ROUTINE
            LDP        5          SLRTN IS IN PAGE 5
            CALL       SLRTN      CALL SLRTN
            BR         LOOP       RETURN HERE, BRANCH TO LOOP
*
DONE        LDP        4          GO TO PAGE 4
            BR         MORE       PERFORM LONG BRANCH
```

Page 5

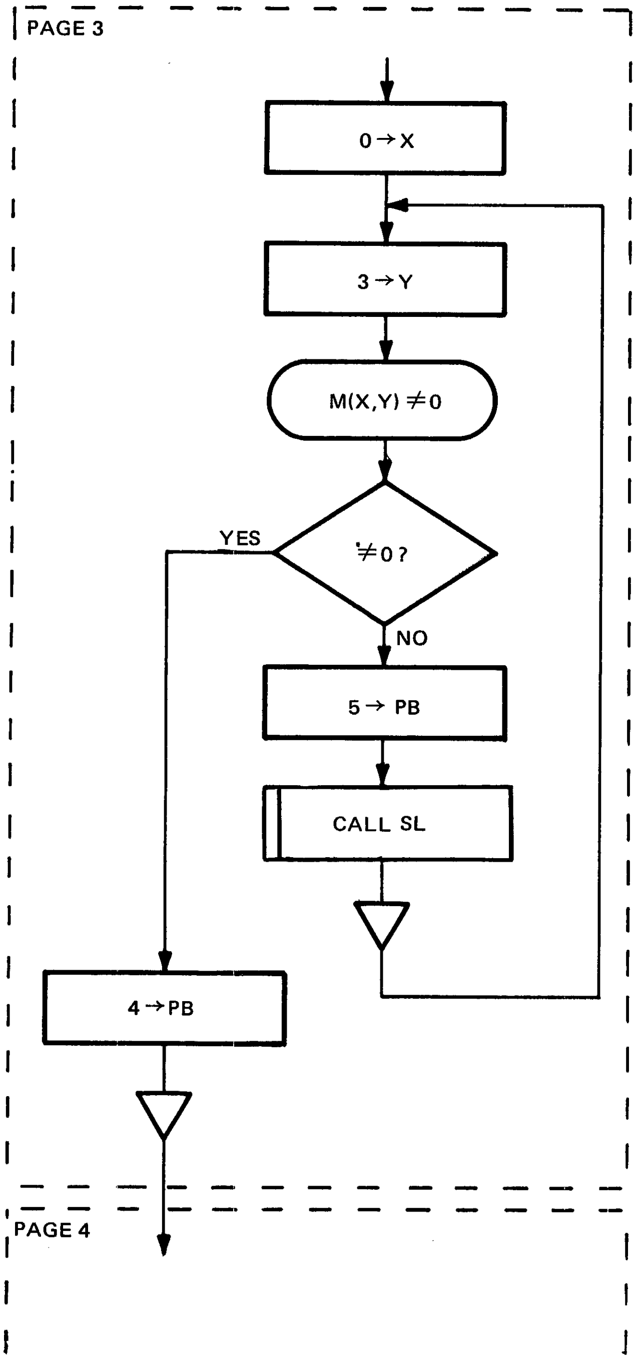
```
* COMMON SUBROUTINE, SLRTN, SHIFT LEFT.
SLRTN       TCY        0          CLEAR Y INDEX
            CLA                   CLEAR A
SWITCH      XMA                   EXCHANGE MEMORY & ACCUMULATOR
            IYC                   INCREMENT Y INDEX
            YNEC       4          Y = 4? (END OF STRING)
            BR         SWITCH     CONTINUE IF NOT EQUAL
            RETN                  RETURN TO CALL
```

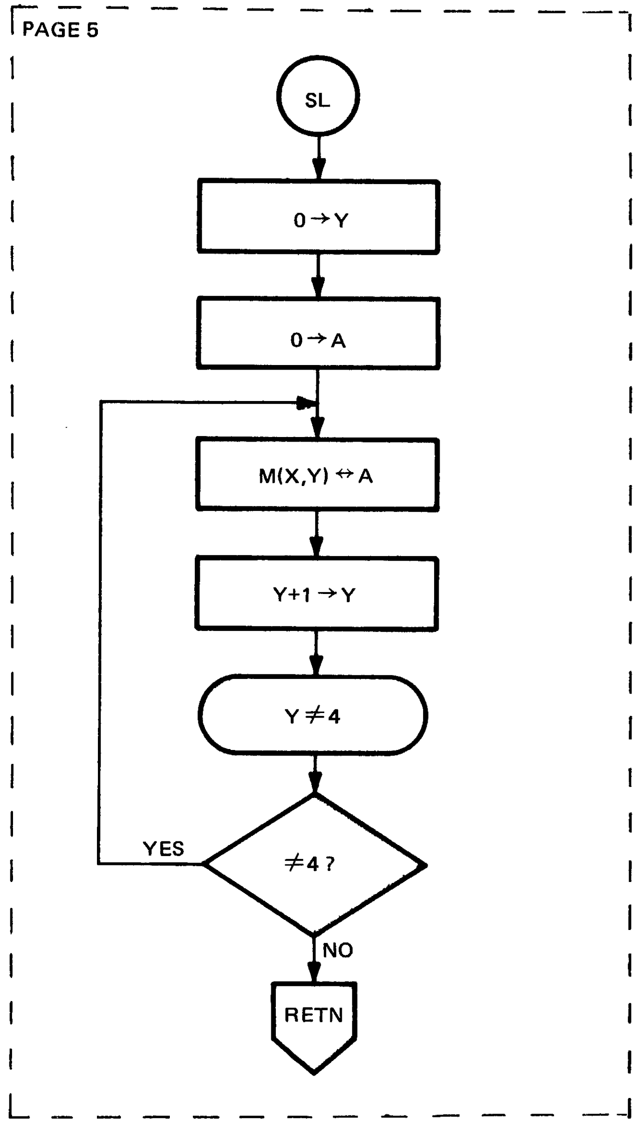
Shift left until M(0,3) ≠ 0, then branch to page 4.

Before					
X =	0				
Y	4	3	2	1	0
M(X,Y)	0	0	1	8	3

		X	Y	A	M(X,Y)	S	PA	PB	CL	BRANCH
		3	9	2	5	1	3	3	0	
	LDX 0	0	9	2	7	1	3	3	0	
	TCY 3	0	3	2	0	1	3	3	0	
	MNEZ	0	3	2	0	1	3	3	0	
	BR DONE	0	3	2	0	0	3	3	0	NO
	LDP 5	0	3	2	0	1	3	5	0	
	CALL SLRTN	0	3	2	0	1	5	3	1	
SLRTN	TCY 0	0	0	2	3	1	5	3	1	
	CLA	0	0	0	3	1	5	3	1	
SWITCH	XMA	0	0	3	0	1	5	3	1	
	IYC	0	1	3	8	1	5	3	1	
	YNEC 4	0	1	3	8	0	5	3	1	
	BR SWITCH	0	1	3	8	1	5	3	1	YES
SWITCH	XMA	0	1	8	3	1	5	3	1	
	IYC	0	2	8	1	1	5	3	1	
	YNEC 4	0	2	8	1	0	5	3	1	
	BR SWITCH	0	2	8	1	1	5	3	1	YES
SWITCH	XMA	0	2	1	8	1	5	3	1	
	IYC	0	3	1	0	1	5	3	1	
	YNEC 4	0	3	1	0	0	5	3	1	
	BR SWITCH	0	3	1	0	1	5	3	1	YES
SWITCH	XMA	0	3	0	1	1	5	3	1	
	IYC	0	4	0	0	1	5	3	1	
	YNEC 4	0	4	0	0	0	5	3	1	
	BR SWITCH	0	4	0	0	0	5	3	1	NO
	RETN	0	4	0	0	1	3	3	0	
	BR LOOP	0	4	0	0	1	3	3	0	YES
LOOP	TCY 3	0	3	0	1	1	3	3	0	
	MNEZ	0	3	0	1	1	3	3	0	
	BR DONE	0	3	0	1	1	3	3	0	YES
DONE	LDP 4	0	3	0	1	1	3	4	0	
	BR MORE	0	3	0	1	1	4	4	0	YES

After					
X =	0				
Y	4	3	2	1	0
M(X,Y)	0	1	8	3	0


PAGE 3
0 → X
3 → Y
M(X,Y) ≠ 0
≠ 0 ?
YES
NO
5 → PB
CALL SL
4 → PB
PAGE 4


PAGE 5
SL
0 → Y
0 → A
M(X,Y) ↔ A
Y+1 → Y
Y ≠ 4
≠ 4 ?
YES
NO
RETN

4-12.6 PROGRAM CONTROL EXAMPLE 2. The following example illustrates using the call instruction to conditionally call a subroutine.

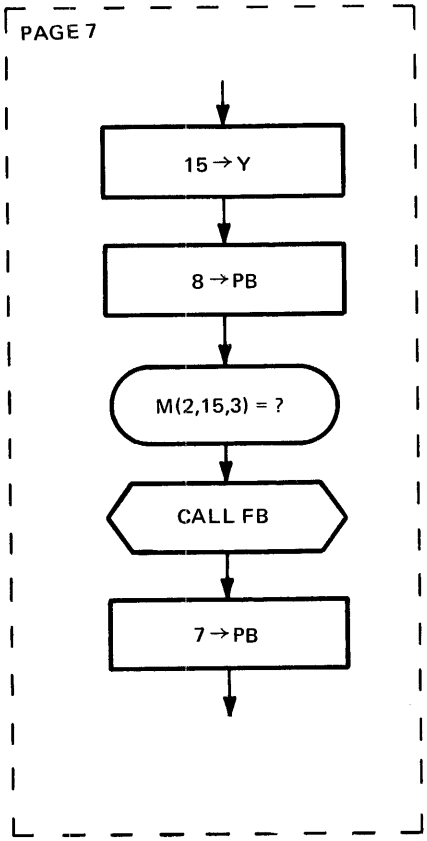
This example shows how to set up paging to a possible call before the instruction that sets the proper status. The subroutine is then conditionally called. Note that the current page must then be reset before any branching occurs. In this example, the subroutine will force bit 3 of the current memory word to a ZERO.

Page 7

```
LABEL     OP CODE   OPERAND   COMMENT

          LDX       2         SET RAM FILE ADDRESS
          TCY       15        ADDRESS M(2, 15)
          LDP       8         SET UP PB FOR CALL
          TBIT1     3         M(2, 15, 3) = 1?
          CALL      FB        CALL FB IF YES
          LDP       7         RESTORE PROPER PAGE ADDRESS
          .
          .
```

Page 8

```
* SUBROUTINE TO FLIP BIT 3 OFF
FB        RBIT      3         RESETS M(2, 15, 3) TO ZERO
          RETN                RETURN TO INSTRUCTION AFTER CALL
```

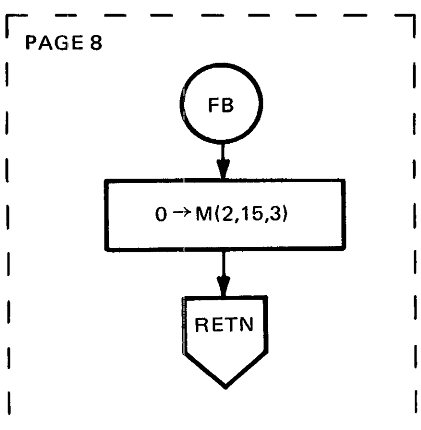


1. M(2,15,3) = 0

		X	Y	A	M(X,Y)	S	PA	PB	CL	CALL
LDX	2	2	6	0	4	1	7	7	0	
TCY	15	2	F	0	7	1	7	7	0	
LDP	8	2	F	0	7	1	7	8	0	
TBIT1	3	2	F	0	7	1	7	8	0	
CALL	FB	2	F	0	7	0	7	8	0	NO
LDP	7	2	F	0	7	1	7	7	0	

2. M(2,15,3) = 1

			X	Y	A	M(X,Y)	S	PA	PB	CL	CALL
FB	LDX	2	2	6	0	2	1	7	7	0	
	TCY	15	2	F	0	9	1	7	7	0	
	LDP	8	2	F	0	9	1	7	8	0	
	TBIT1	3	2	F	0	9	1	7	8	0	
	CALL	FB	2	F	0	9	1	8	7	1	YES
	RBIT	3	2	F	9	1	1	8	7	1	
	RETN		2	F	9	1	1	7	8	1	
	LDP	7	2	F	1	1	1	7	7	0	



SECTION V

TMS 1100/1300

5-1 INTRODUCTION.

The following features are contained in an expanded ROM and RAM version of the TMS 1000 series:

TMS 1100

- Pin-for-pin compatible with the TMS 1000
- 16K-bit ROM, 2048 eight-bit instructions
- 512-bit RAM, 128 four-bit data words.

TMS 1300

- 16K-bit ROM
- 512-bit RAM
- 16 individually-latched R-outputs, 40-pin package.

The next three sections of this manual detail the functional differences between the TMS 1000/1200 and the TMS 1100/1300. Thus, one should have read the previous four sections before continuing.

5-2 DESIGN SUPPORT.

To simulate the TMS 1100/1300 the user may access an assembler and simulator via one of several nationwide timeshare processing services. The assembler and simulator programs written by Texas Instruments are now capable of aiding design work on the new devices in the TMS 1000 series (see [1], [2], and [3] in Figure 5-2). In additio to software simulation, two methods of real-time algorithm verification are available. The HE-2, hardware emulator [4], is a prototyping system with a removeable memory board to debug the TMS 1000/1200 or a TMS 1100/1300 program. The HE-2 provides several debug aids such as memory inspection, a single-step, and breakpoint, and modification of all the programmable areas in the machine is done by a paper-tape reader.

For most applications, a System Evaluator-2 (TMS 1098/SE-2 [5]) is available for real-time algorithm verification. The SE-2 is a P-channel MOS/LSI microprocessor that is identical to the TMS 1100/1300 microcomputer with the ROM removed and the O-output register bits transferred out directly. The user replaces the ROM with external memory devices such as PROM, EPROM or a suitably loaded RAM. The program-counter, page-address, and chapter-address outputs select the instruction word stored in the program memory which feeds into an 8-bit parallel instruction bus. Then each instruction executes exactly like the TMS 1100/1300 standard-instruction-set descriptions found in Section VII and VIII. An external decoder may be necessary to convert the five-bit O-output code to the appropriate eight-bit code to completely simulate the O-output PLA.

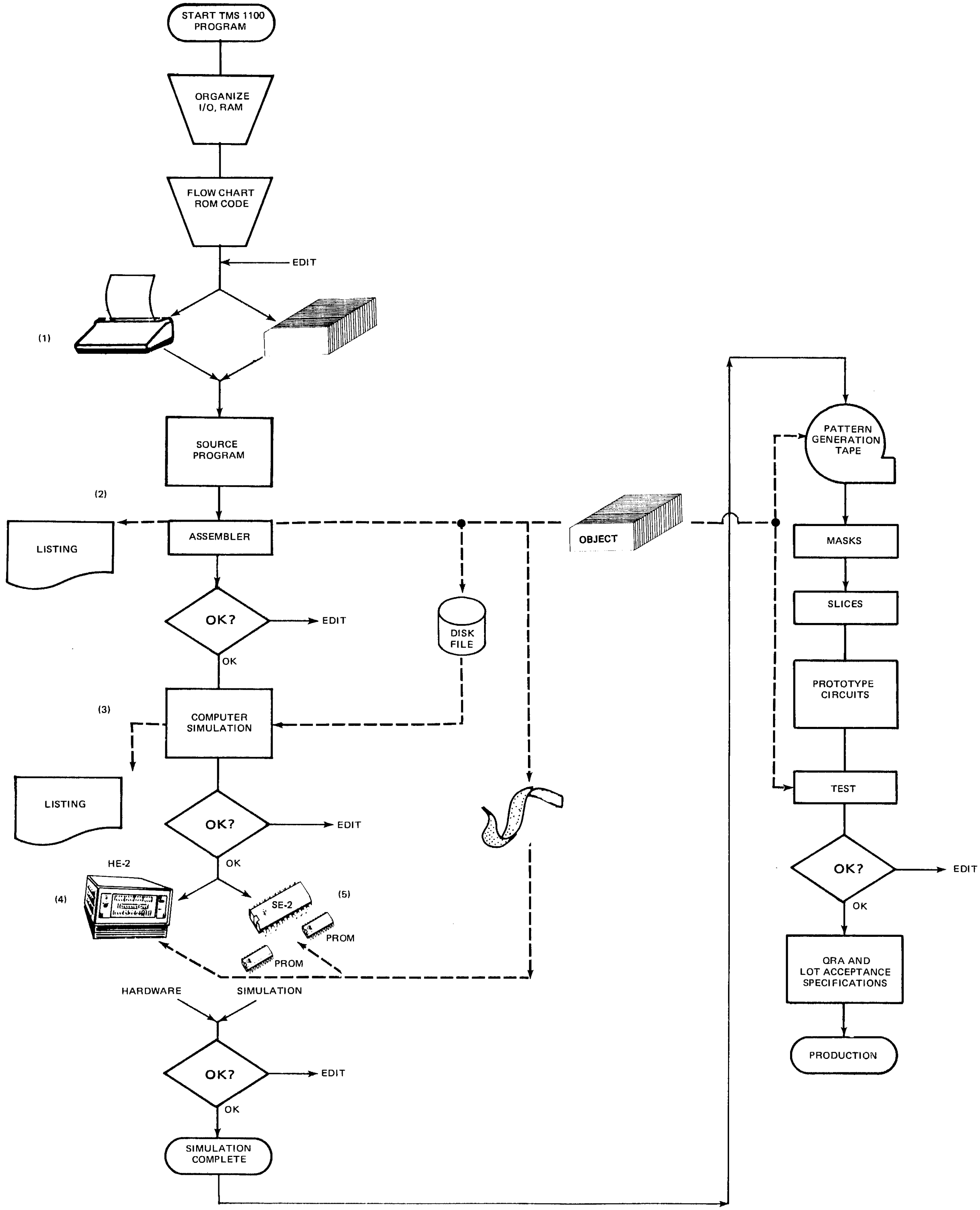


FIGURE 5-1 TMS 1100/1300 SERIES ALGORITHM DEVELOPMENT

SECTION VI

TMS 1100/1300 OPERATION

6-1 GENERAL.

The following sections concentrate on the differences between the TMS 1100/1300 and the TMS 1000/1200 features. The user should first understand the TMS 1000/1200 architecture and operation before reading the sections covering the following:

- Instruction Read-Only Memory
- Read/Write Random-Access-Memory
- Outputs
- Instruction PLA
- Fixed Instruction Decoder
- Standard Instruction Set.

The areas shaded in the TMS 1100/1300 functional block diagram (Figure 6-1.1) have been added or improved to change the basic TMS 1000/1200 architecture to the new TMS 1100/1300 design. The unshaded functional blocks operate per the previous TMS 1000/1200 descriptions unless stated otherwise.

6-2 ROM ADDRESSING.

The TMS 1100/1300 features 2048 eight-bit instruction words stored permanently in the ROM. Providing twice the TMS 1000/1200's ROM capacity requires one additional addressing bit, for a total of eleven bits. As previously described, the ROM is divided into 16 pages of 64 words each. The TMS 1100/1300 ROM has two chapters each containing 16 pages. The following latches control the chapter addressing:

1. CA – The chapter address latch stores the current chapter data.
2. CB – The chapter buffer latch stores the succeeding chapter data and transfers to the CA pending the successful execution of a subsequent branch or call instruction.
3. CS – The chapter subroutine latch stores the return address after successfully executing a call instruction (CA → CS). CS transfers data back to CA when the return from subroutine (RETN) instruction occurs.

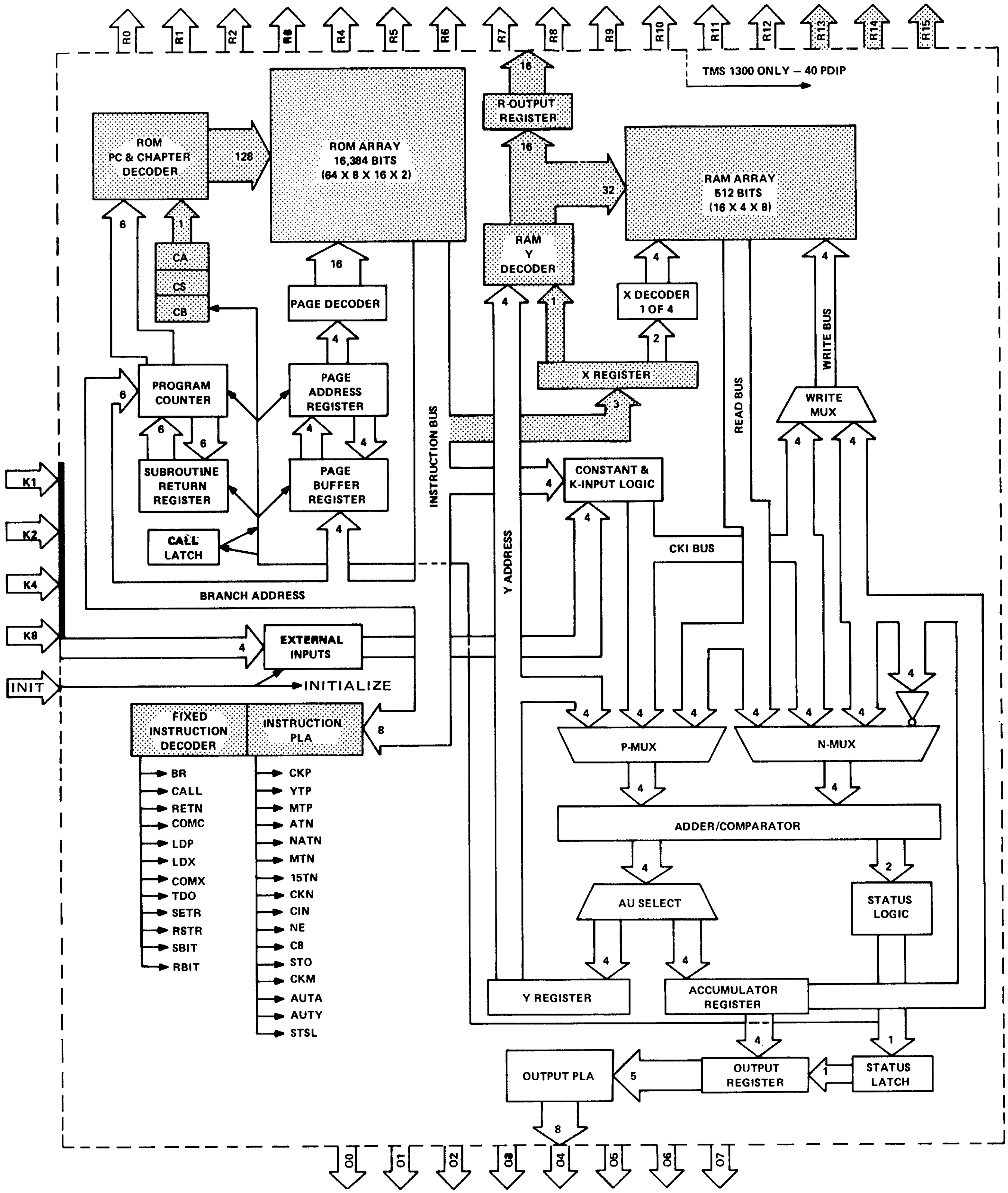


FIGURE 6-1.1 EXPANDED RAM/ROM TMS 1100 SERIES FUNCTIONAL BLOCK DIAGRAM

To begin with, the three latches are reset upon application of power to the circuit. When software control must shift to the other chapter, the complement chapter (COMC) instruction toggles the chapter-buffer bit. At any time after this point either branches to a routine or calls to a subroutine in chapter one is possible if status is equal to ONE. The following actions occur for the branch and call instructions.

	S = 1 CL = 0 ACTION	S = 1 CL = 1 ACTION	COMMENT
BR			
	1) CB → CA	1) CB → CA	Change Chapter
	2) PB → PA		Only if CL = 0, change page
	3) I(W) → PC	3) I(W) → PC	Change address on page
CALL			
	1) CA → CS		Store chapter return
	2) CB → CA	2) CB → CA	Change chapter
	3) PB ↔ PA	3) PA → PB	Exchange only if CL = 0
	4) PC+1 → SR		Store return address
	5) I(W) → PC	5) I(W) → PC	Change address on page
	6) 1 → CL		Store call mode

Note that if the call latch is set at the time of the branch, item 2) (S = 1) is skipped. To change the page buffer (by LDP) is improper if CL = 1 since PB holds the return address during the call mode. However, branches between the two chapters are permissible without executing the load-page-buffer (LDP) instruction during the call mode because actions 1) and 3) do occur. Thus, the size of the TMS 1100/1300 subroutine has increased to 128 instructions from 64 instructions maximum. As shown graphically in Figure 6-2.1, it is convenient to envision the ROM with adjacent pages that are equal in number.

Call instructions in a subroutine are invalid if status is ONE since the return address in PB is destroyed. Item 3) for the call instruction is PA → PB when the call latch is set.

If the status is ZERO due to the previous instructions effects, the branch or call is unsuccessfully executed (PC + 1 → PC), and status returns to ONE. If the status is always ONE due to the previous instruction (LDP or COMC for example), then the following branch or call is unconditional.

The return from subroutine instruction (RETN) causes the following actions.

RETN	1) SR → PC	CL = 1
	2) PB → PA	
	3) CS → CA	
	4) 0 → CL	

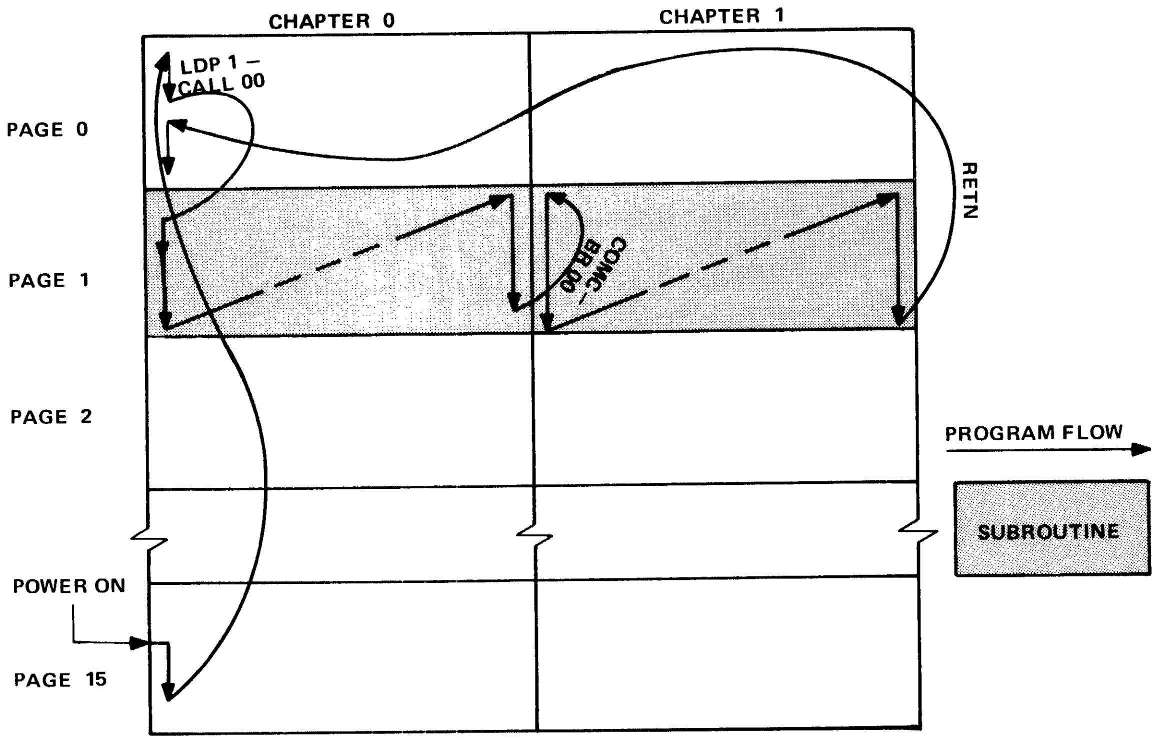


FIGURE 6-2.1 ROM ORGANIZATION

If the call latch was not set by a previous call instruction, the SR is equal to PC + 1, and CS is equal to CA. Therefore, the following summarizes the effect of the return instruction:

RETN	**1) PC + 1 → PC**	**CL = 0**
	2) PB → PA	

The return instruction is not dependent on the status logic and leaves status set at ONE after execution.

6-3 RAM ADDRESSING.

Since the TMS 1100/1300 RAM has twice the storage of a TMS 1000/1200 RAM, there is an additional bit of addressing for a total of seven. As before, the RAM is organized into four-bit words and each individual bit in a word can be set, reset, or tested once the X- and Y-address is fixed. To accommodate a larger RAM, the load-X-register instruction (LDX) has the following format:

I(0)	I(1)	I(2)	I(3)	I(4)	I(5) LSB	I(6)	I(7) MSB
0	0	1	0	1		F	

FORMAT V

LDX Action: I(5-7) → X
or
I(F) → X

The RAM is organized into eight files (addressed by X) each containing 16 four-bit words (addressed by Y). The most-significant bit of X is decoded to address two halves of the Y-decoder. The lower-order half of the Y-decoder is selected when $X_{MSB} = 0$ and only the low-order address lines enable the R-output register bits.

The 16 R-output-register-address lines are only available when the X-register contents are between zero and three, the most-significant bit being reset.

The TMS 1100/1300 complement X-register (COMX) causes only the most-significant bit to change state. Thus, complement X will change the X-address from file zero-to-four, one-to-five, two-to-six, three-to-seven, and vice-versa.

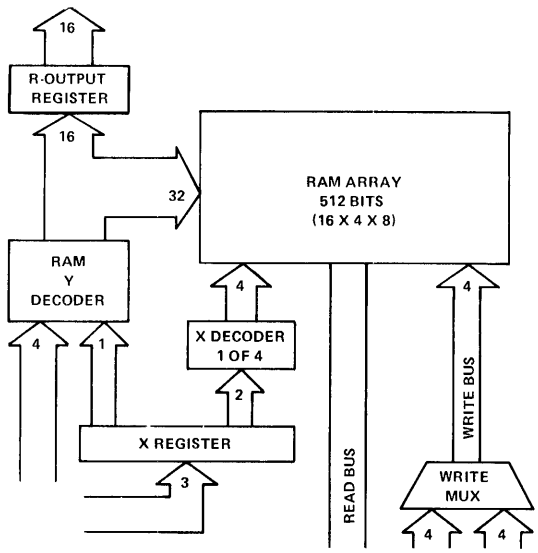


FIGURE 6-3.1 RAM ORGANIZATION

6-4 CONTROL AND DATA OUTPUTS.

6-4.1 R-OUTPUTS. In the TMS 1300 device 16 R-outputs are available and 11 are available in the TMS 1100. The maximum stand-alone keyboard matrix scanned by the TMS 1300 is shown on the next page.

The Y-register values select the appropriate bit for the SETR and RSTR instructions. Y-register must be less than or equal to ten in the TMS 1100, and the X-register must be less then or equal to three to address a R-output. The full Y-address range from zero-to-fifteen is usable by the TMS 1300.

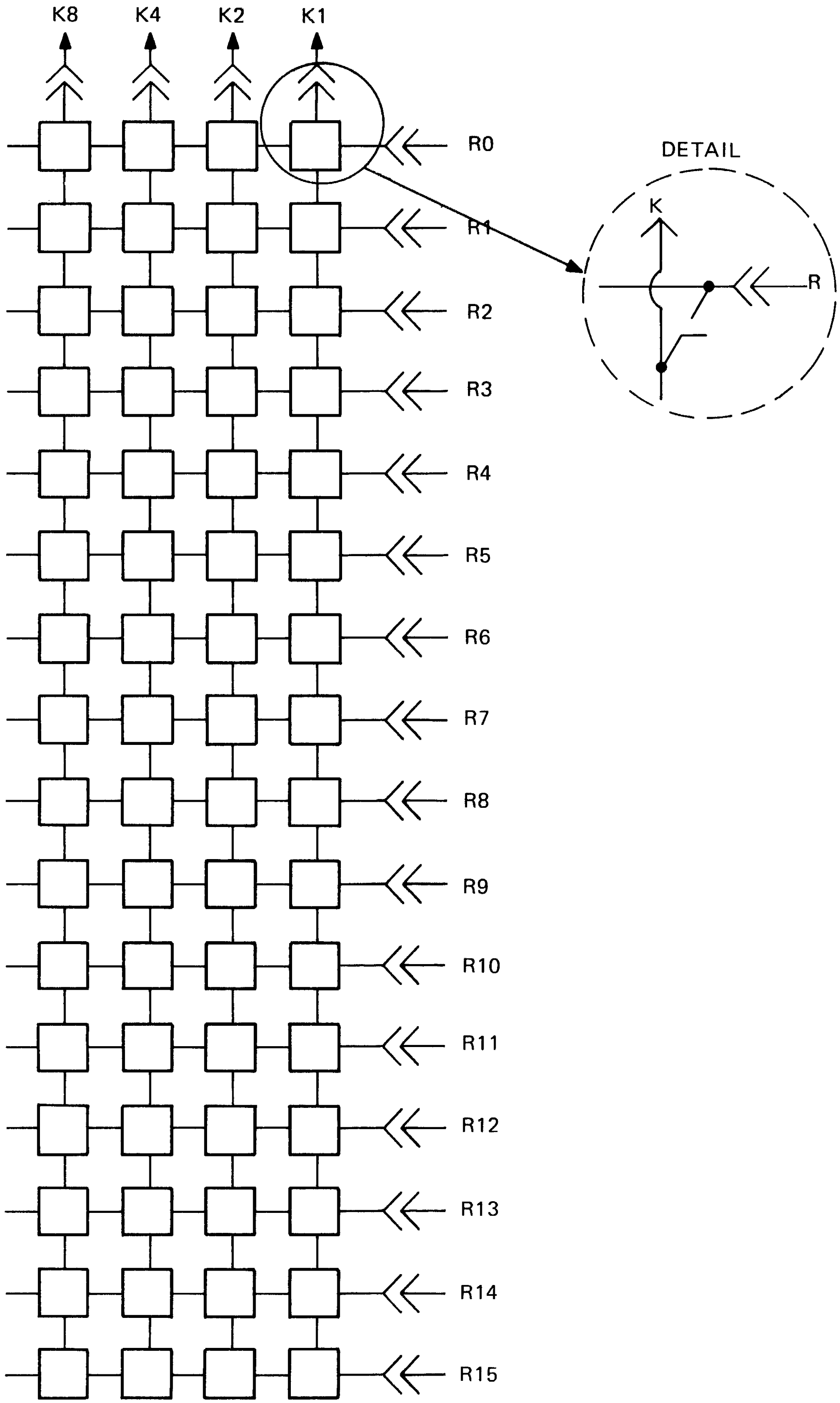


FIGURE 6-4.1 KEYBOARD MATRIX CONNECTIONS

6-4.2 O-OUTPUTS. The O-output configuration in the TMS 1100/1300 is identical to the TMS 1000/1200. However, the clear-output-register instruction (CLO) was replaced by the new complement-chapter (COMC) instruction. The effect of clearing the O-output register is obtained in the TMS 1100/1300 by loading zero in the accumulator and the status latch and then performing the transfer-data-out (TDO) instruction. In most cases, the above procedure is the normal sequence for transferring out all data anyway; hence, there is no disadvantage in deleting the CLO instruction.

6-5 INSTRUCTION DECODERS.

The 54 instructions decoded by the instruction PLA and the fixed-instruction decoder comprise the TMS 1100/1300 standard-instruction set. The 12 fixed instructions and 42 programmable instructions are listed in Table 6-5.1 with their corresponding fixed- and programmable-microinstructions. The TMS 1100/1300 standard-instruction set, which was designed for maximum flexibility, will be used by most programs. However, if timing or other considerations dictate an instruction's redefinition, contact the MOS Division in Houston, Texas. To aid users who need to microprogram the instruction set, Section IX contains helpful hints and guidelines for redefining instructions.

6-5.1 THE INSTRUCTION-PROGRAMMABLE-LOGIC ARRAY. The shaded functional blocks in Figure 6-5.1 are affected by the 16 programmable-microinstructions. The effect of enabling a given microinstruction is described in Table 6-5.2 and is identical to the TMS 1000/1200 operation. To provide a starting point for user algorithms, the standard coding for the instruction PLA should be used. The standard mnemonics and instruction definitions are resident in the Texas Instruments TMS 1100/1300 Assembler and Simulator Programs; so, users can begin algorithm designs readily and check-out may be accomplished with the SE-2. With the standard definitions for TMS 1100/1300 instructions, an automatic test generation program provides complete functional and parametric testing capability for every user's custom ROM and output PLA design.

The PLA schematic in Figure 6-5.2 shows the gate-mask coding for the standard instruction's microinstructions listed in Table 6-5.2. Figure 6-5.3 shows the SE-2 gate-placement option which is used to generate the PLA coding for the TMS 1100/1300 standard instructions (Table 6.5-1). This coding is the default **OPCPLA** description stored in the TMS 1100/1300 simulator (and can be punched in paper tape for loading the instruction definitions into the HE-2 instruction-PLA RAMs).

6-5.2 THE FIXED-INSTRUCTION DECODER. The 12 fixed instructions are decoded by fixed logic and cannot be changed. Every program must use the assigned opcode values as described by Table 6-5.1. The fixed microinstructions emanating from the bottom of the fixed-instruction decoder in Figure 6-5.4 fan out to the shaded logic blocks that they affect. The mnemonics have a one-to-one correspondence between the fixed instructions and their microinstructions because the standard instruction set uses one microinstruction (no programmable microinstruction) per fixed instruction (see Table 6-5.1).

TABLE 6-5.1 TMS 1100/1300 MICROINSTRUCTION INDEX

Mnemonic	Opcode								Microinstructions	
									Fixed	Programmable
A2AAC	0	1	1	1	1	0	0	0		CKP,ATN,CIN,C8,AUTA
A3AAC	0	1	1	1	0	1	0	0		CKP,ATN,CIN,C8,AUTA
A4AAC	0	1	1	1	1	1	0	0		CKP,ATN,CIN,C8,AUTA
A5AAC	0	1	1	1	0	0	1	0		CKP,ATN,CIN,C8,AUTA
A6AAC	0	1	1	1	1	0	1	0		CKP,ATN,CIN,C8,AUTA
A7AAC	0	1	1	1	0	1	1	0		CKP,ATN,CIN,C8,AUTA
A8AAC	0	1	1	1	1	1	1	0		CKP,ATN,CIN,C8,AUTA
A9AAC	0	1	1	1	0	0	0	1		CKP,ATN,CIN,C8,AUTA
A10AAC	0	1	1	1	1	0	0	1		CKP,ATN,CIN,C8,AUTA
A11AAC	0	1	1	1	0	1	0	1		CKP,ATN,CIN,C8,AUTA
A12AAC	0	1	1	1	1	1	0	1		CKP,ATN,CIN,C8,AUTA
A13AAC	0	1	1	1	0	0	1	1		CKP,ATN,CIN,C8,AUTA
A14AAC	0	1	1	1	1	0	1	1		CKP,ATN,CIN,C8,AUTA
ALEM	0	0	0	0	0	0	0	1		MTP,NATN,CIN,C8
AMAAC	0	0	0	0	0	1	1	0		ATN,MTP,C8,AUTA
BR	1	0			W				BR	
CALL	1	1			W				CALL	
CLA	0	1	1	1	1	1	1	1		CKP,CIN,C8,AUTA
COMC	0	0	0	0	1	0	1	1	COMC	
COMX	0	0	0	0	1	0	0	1	COMX	
CPAIZ	0	0	1	1	1	1	0	1		NATN,CIN,C8,AUTA
DAN	0	1	1	1	0	1	1	1		CKP,ATN,CIN,C8,AUTA
DMAN	0	0	0	0	0	1	1	1		MTP,15TN,C8,AUTA
DYN	0	0	0	0	0	1	0	0		YTP,15TN,C8,AUTY
IAC	0	1	1	1	0	0	0	0		CKP,ATN,CIN,C8,AUTA
IMAC	0	0	1	1	1	1	1	0		MTP,CIN,C8,AUTA
IYC	0	0	0	0	0	1	0	1		YTP,CIN,C8,AUTY
KNEZ	0	0	0	0	1	1	1	0		CKP,NE
LDP	0	0	0	1			C		LDP	
LDX	0	0	1	0	1		F		LDX	
MNEA	0	0	0	0	0	0	0	0		MTP,ATN,NE
MNEZ	0	0	1	1	1	1	1	1		MTP,NE
RBIT	0	0	1	1	0	1		B	RBIT	
RETN	0	0	0	0	1	1	1	1	RETN	
RSTR	0	0	0	0	1	1	0	0	RSTR	
SAMAN	0	0	1	1	1	1	0	0		MTP,NATN,CIN,C8,AUTA
SBIT	0	0	1	1	0	0		B	SBIT	
SETR	0	0	0	0	1	1	0	1	SETR	
TAM	0	0	1	0	0	1	1	1		STO
TAMDYN	0	0	1	0	0	1	0	0		STO,YTP,15TN,C8,AUTY
TAMIYC	0	0	1	0	0	1	0	1		STO,YTP,CIN,C8,AUTY
TAMZA	0	0	1	0	0	1	1	0		STO,AUTA
TAY	0	0	1	0	0	0	0	0		ATN,AUTY
TBIT1	0	0	1	1	1	0		B		CKP,CKN,MTP,NE
TCY	0	1	0	0			C			CKP,AUTY
TCMIY	0	1	1	0			C			CKM,YTP,CIN,AUTY
TDO	0	0	0	0	1	0	1	0	TDO	
TKA	0	0	0	0	1	0	0	0		CKP,AUTA
TMA	0	0	1	0	0	0	0	1		MTP,AUTA
TMY	0	0	1	0	0	0	1	0		MTP,AUTY
TYA	0	0	1	0	0	0	1	1		YTP,AUTA
XMA	0	0	0	0	0	0	1	1		MTP,STO,AUTA
YNEA	0	0	0	0	0	0	1	0		YTP,ATN,NE,STSL
YNEC	0	1	0	1			C			YTP,CKN,NE

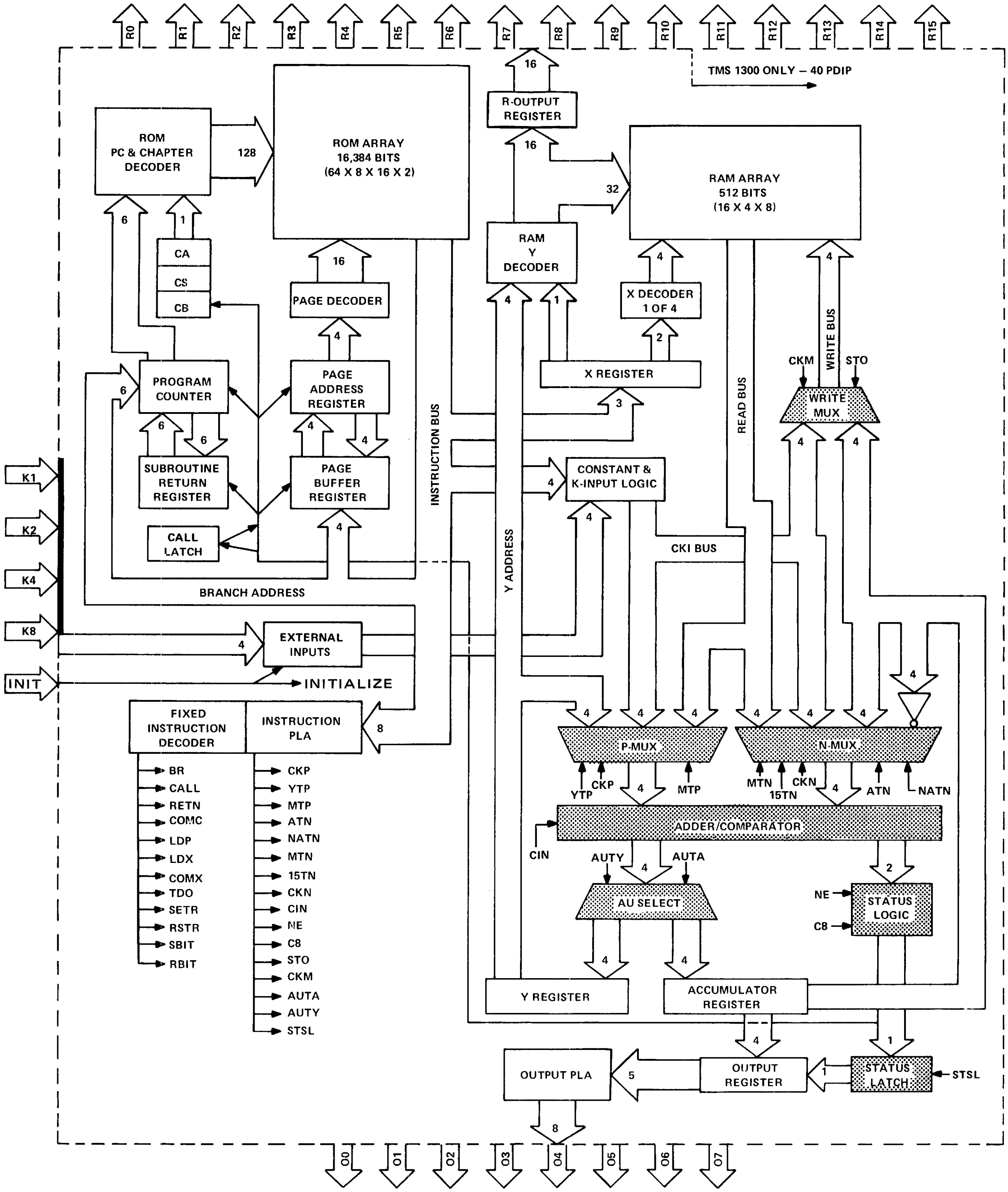


FIGURE 6-5.1 TMS 1100/1300 FUNCTIONAL BLOCKS AND PROGRAMMABLE MICROINSTRUCTIONS

TABLE 6-5.2 TMS 1000 SERIES PROGRAMMABLE MICROINSTRUCTIONS

Execution Sequence	Mnemonic	Affected Logic	Function
1	CKP	P-MUX	CKI to P-adder input
	YTP	P-MUX	Y-register to P-adder input
	MTP	P-MUX	Memory (X,Y) to P-adder input
1	ATN	N-MUX	Accumulator to N-adder input
	NATN	N-MUX	$\overline{\text{Accumulator}}$ to N-adder input
	MTN	N-MUX	Memory (X,Y) to N-adder input
	15TN	N-MUX	F_{16} to N-adder input
	CKN	N-MUX	CKI to N-adder input
1	CIN	Adder	One is added to sum of P plus N inputs (P+N+1)
	NE	Adder/Status	Adder compares P and N inputs. If they are identical, status is set to zero.
	C8	Adder/Status	Carry is sent to status (MSB only)
2	STO	Write MUX	Accumulator data to memory
	CKM	Write MUX	CKI to memory
3	AUTA	AU Select	Adder result stored into accumulator
	AUTY	AU Select	Adder result stored into Y-register
	STSL	Status Latch	Status is stored into status latch

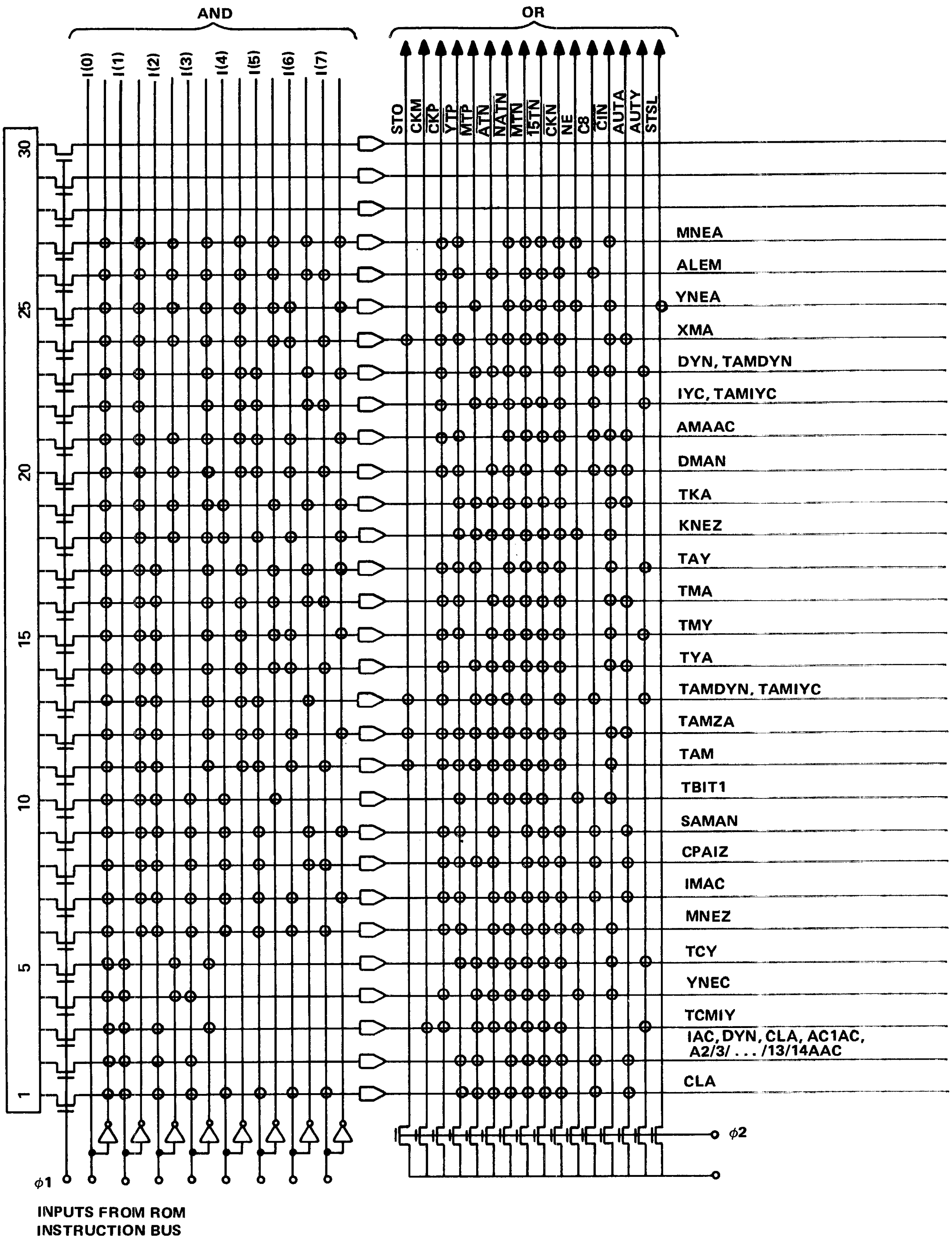


FIGURE 6-5.2 TMS 1100/1300 STANDARD INSTRUCTION PLA

OPCPLA	
OPX 00 = MTP, ATN, NE;	MNEA
OPX 01 = MTP, NATN, CIN, C8;	ALEM
OPX 02 = YTP, ATN, NE, STSL;	YNEA
OPX 03 = STO, MTP, AUTA;	XMA
OPB 00 – 00100 = YTP, 15TN, AUTY, C8;	DYN, TAMDYN
OPB 00 – 00101 = YTP, CIN, AUTY, C8;	IYC, TAMIYC
OPX 06 = MTP, ATN, AUTA, C8;	AMAAC
OPX 07 = MTP, 15TN, AUTA, C8;	DMAN
OPX 08 = CKP, AUTA;	TKA
OPX 0E = CKP, NE;	KNEZ
OPX 20 = ATN, AUTY;	TAY
OPX 21 = MTP, AUTA;	TMA
OPX 22 = MTP, AUTY;	TMY
OPX 23 = YTP, AUTA;	TYA
OPB 0010010 – = STO, YTP, 15TN, CIN, AUTY, C8;	TAMDYN, TAMIYC
OPX 26 = STO, AUTA;	TAMZA
OPX 27 = STO;	TAM
OPB 001110 – – = CKP, CKN, MTP, NE;	TBIT1
OPX 3C = MTP, NATN, CIN, AUTA, C8;	SAMAN
OPX 3D = NATN, CIN, AUTA, C8;	CPAIZ
OPX 3E = MTP, CIN, AUTA, C8;	IMAC
OPX 3F = MTP, NE;	MNEZ
OPB 0100 – – – – = CKP, AUTY;	TCY
OPB 0101 – – – – = YTP, CKN, NE;	YNEC
OPB 0110 – – – – = CKM, YTP, CIN, AUTY;	TCMIY
OPB 0111 – – – – = CKP, ATN, CIN, AUTA, C8;	IAC, DAN, A2/3/ . . . /13/14AAC, CLA, AC1AC
OPX 7F = CKP, CIN, AUTA, C8;	CLA

FIGURE 6-5.3 TMS 1100/1300 STANDARD INSTRUCTION PLA CODING

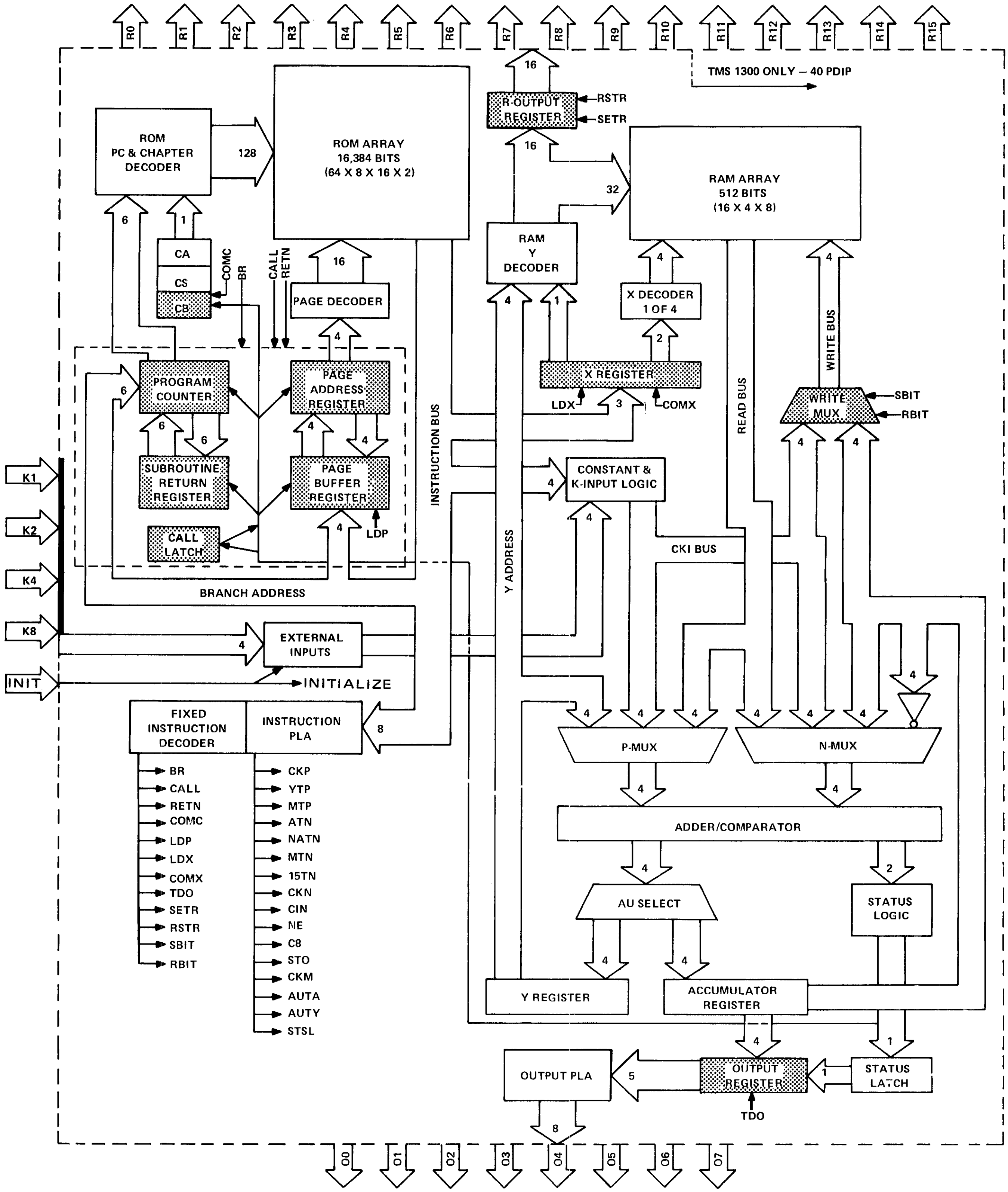


FIGURE 6-5.4 TMS 1100/1300 FUNCTIONAL BLOCKS AND FIXED MICROINSTRUCTIONS

SECTION VII

CROSS-REFERENCE TABLES TMS 1100/1300

This section provides the user with a quick-reference to the 40 TMS 1100/1300 instructions. Symbolic descriptions and paragraph references help rapid understanding of the standard instruction set.

- Table 7-1 lists the instructions by function
- Table 7-2 lists the instructions alphabetically
- Table 7-3 lists the microinstructions for each instruction
- Table 7-4 lists the instructions by binary machine code
- Figure 7-1 is a hexadecimal-instruction map.

TABLE 7-1 TMS 1100/1300 STANDARD INSTRUCTION SET

Function	Mnemonic	Status Effect C8	Status Effect NE	Description	Explained In Para.‡
Register-to-Register Transfer	TAY			Transfer accumulator to Y register	4-2.1
	TYA			Transfer Y register to accumulator	4-2.2
	CLA			Clear accumulator	4-2.3
Register to Memory	TAM			Transfer accumulator to memory	4-3.1
	TAMIYC	Y		Transfer accumulator to memory and increment Y register. If carry, one to status.	8-4.1
	TAMDYN	Y		Transfer accumulator to memory and decrement Y register. If no borrow, one to status.	8-4.2
	TAMZA			Transfer accumulator to memory and zero accumulator	4-3.3
Memory to Register	TMY			Transfer memory to Y register	4-3.4
	TMA			Transfer memory to accumulator	4-3.5
	XMA			Exchange memory and accumulator	4-3.6
Arithmetic	AMAAC	Y		Add memory to accumulator, results to accumulator. If carry, one to status	4-4.1
	SAMAN	Y		Subtract accumulator from memory, results to accumulator. If no borrow, one to status.	4-4.2
	IMAC	Y		Increment memory and load into accumulator. If carry, one to status	4-4.3
	DMAN	Y		Decrement memory and load into accumulator. If no borrow, one to status.	4-4.4
	IAC	Y		Increment accumulator. If no carry, one to status.	8-4
	DAN	Y		Decrement accumulator. If no borrow, one to status.	8-4
	A2AAC	Y		Add 2 to accumulator. Results to accumulator. If carry one to status.	8-4
	A3AAC	Y		Add 3 to accumulator. Results to accumulator. If carry one to status.	8-4
	A4AAC	Y		Add 4 to accumulator. Results to accumulator. If carry one to status.	8-4
	A5AAC	Y		Add 5 to accumulator. Results to accumulator. If carry one to status.	8-4
	A6AAC	Y		Add 6 to accumulator. Results to accumulator. If carry one to status.	8-4
	A7AAC	Y		Add 7 to accumulator. Results to accumulator. If carry one to status.	8-4
	A8AAC	Y		Add 8 to accumulator. Results to accumulator. If carry one to status.	8-4
	A9AAC	Y		Add 9 to accumulator. Results to accumulator. If carry one to status.	8-4
	A10AAC	Y		Add 10 to accumulator. Results to accumulator. If carry one to status.	8-4
	A11AAC	Y		Add 11 to accumulator. Results to accumulator. If carry one to status.	8-4
	A12AAC	Y		Add 12 to accumulator. Results to accumulator. If carry one to status.	8-4
	A13AAC	Y		Add 13 to accumulator. Results to accumulator. If carry one to status.	8-4
	A14AAC	Y		Add 14 to accumulator. Results to accumulator. If carry one to status.	8-4
	IYC	Y		Increment Y register. If carry, one to status.	4-4.6
	DYN	Y		Decrement Y register. If no borrow, one to status.	4-4.8
	CPAIZ	Y		Complement accumulator and increment. If then zero, one to status.	4-4.12
Arithmetic Compare	ALEM	Y		If accumulator less than or equal to memory, one to status.	4-5.1
Logical Compare	MNEA		Y	If memory is not equal to accumulator, one to status.	8-5
	MNEZ		Y	If memory not equal to zero, one to status.	4-6.1
	YNEA		Y	If Y register not equal to accumulator, one to status and status latch.	4-6.2
	YNEC		Y	If Y register not equal to a constant, one to status.	4-6.3
Bits in Memory	SBIT			Set memory bit.	4-7.1
	RBIT			Reset memory bit	4-7.2
	TBIT1		Y	Test memory bit. If equal to one, one to status.	4-6.3
Constants	TCY			Transfer constant to Y register	4-8.1
	TCMIY			Transfer constant to memory and increment Y	4-8.2
Input	KNEZ		Y	If K inputs not equal to zero, one to status	4-9.1
	TKA			Transfer K inputs to accumulator	4-9.2
Output	SETR			Set R output addressed by Y	8-6.1
	RSTR			Reset R output addressed by Y	8-6.2
	TDO			Transfer data from accumulator and status latch to O outputs	4-10.3
RAM X Addressing	LDX			Load X with file address	8-7.1
	COMX			Complement the MSB of X	8-7.2
ROM Addressing	BR			Branch on status = one	8-8.1
	CALL			Call subroutine on status = one	8-8.2
	RETN			Return from subroutine	8-8.3
	LDP			Load page buffer with constant	4-12.4
	COMC			Complement chapter buffer	8-8.4

‡The TMS 1100/1300 instruction values are different from the TMS 1000/1200 opcodes given in Section IV. The correct values are found in Tables 7-2 through 7-4.

TABLE 7-2 TMS 1100/1300 ALPHABETICAL MNEMONIC REFERENCE

Mnemonic	Opcode Binary	Opcode Hex	Action	Status C8	Status NE	Reference Paragraph
AC1AC	0 1 1 1 C	7 –	A + C + 1 → A	Y		8-4
ALEM	0 0 0 0 0 0 0 1	0 1	A ≤ M (X,Y)	Y		4-5.1
AMAAC	0 0 0 0 0 1 1 0	0 6	M(X,Y) + A → A	Y		4-4.1
BR	1 0 W	—	S = 1, CL = 0: CB → CA, PB → PA, I(W) → PC S = 1, CL = 1: CB → CA, I(W) → PC S = 0, CL = 1 OR 0: PC + 1 → PC, 1 → S			8-8.1
CALL	1 1 W	—	S = 1, CL = 0: CA → CS, CB → CA, PB ↔ PA, PC+1 → SR, I(W) → PC, 1 → CL S = 1, CL = 1: CB → CA, PA → PB, I(W) → PC S = 0, CL = 1 OR 0: PC+1 → PC, 1 → S			8-8.2
CLA	0 1 1 1 1 1 1 1	7 F	0 → A			4-2.3
COMC	0 0 0 0 1 0 1 1	0 B	$\overline{CB}$ → CB			8-8.4
COMX	0 0 0 0 1 0 0 1	0 9	$\overline{X}_{MSB}$ → X_{MSB}			8-7.2
CPAIZ	0 0 1 1 1 1 0 1	3 D	$\overline{A}$ + 1 → A	Y		4-4.12
DMAN	0 0 0 0 0 1 1 1	0 7	M(X,Y) – 1 → A	Y		4-4.4
DYN	0 0 0 0 0 1 0 0	0 4	Y – 1 → Y	Y		4-4.8
IMAC	0 0 1 1 1 1 1 0	3 E	M(X,Y) + 1 → A	Y		4-4.3
IYC	0 0 0 0 0 1 0 1	0 5	Y + 1 → Y	Y		4-4.6
KNEZ	0 0 0 0 1 1 1 0	0 E	$K_{8,4,2,1}$ ≠ 0		Y	4-4.1
LDP	0 0 0 1 C	1 –	C → PB			4-12.4
LDX	0 0 1 0 1 F	2 –	F → X			8-7.1
MNEA	0 0 0 0 0 0 0 0	0 0	M(X,Y) ≠ A		Y	8-5
MNEZ	0 0 1 1 1 1 1 1	3 F	M(X,Y) ≠ 0		Y	4-6.1
RBIT	0 0 1 1 0 1 B	3 –	0 → M(X, Y, B)			4-7.2
RETN	0 0 0 0 1 1 1 1	0 F	CL = 1: SR → PC, PB → PA, CS → CA, 0 → CL CL = 0: PC + 1 → PC, PB → PA			8-8.3
RSTR	0 0 0 0 1 1 0 0	0 C	0 → R(Y)			8-6.2
SAMAN	0 0 1 1 1 1 0 0	3 C	M(X,Y) – A → A	Y		4-4.2
SBIT	0 0 1 1 0 0 B	3 –	1 → M(X,Y,B)			4-7.1
SETR	0 0 0 0 1 1 0 1	0 D	1 → R(Y)			8-6.1
TAM	0 0 1 0 0 1 1 1	2 7	A → M(X,Y)			4-3.1
TAMDYN	0 0 1 0 0 1 0 0	2 4	A → M(X,Y) ; Y – 1 → Y	Y		8-3.2
TAMIYC	0 0 1 0 0 1 0 1	2 5	A → M(X,Y) ; Y + 1 → Y	Y		8-3.1
TAMZA	0 0 1 0 0 1 1 0	2 6	A → M(X,Y) ; 0 → A			4-3.3
TAY	0 0 1 0 0 0 0 0	2 0	A → Y			4-2.1
TBIT1	0 0 1 1 1 0 B	3 –	M(X,Y,B) = 1		Y	4-7.3
TCY	0 1 0 0 C	4 –	C → Y			4-8.1
TCMIY	0 1 1 0 C	6 –	C → M(X,Y) ; Y + 1 → Y			4-8.2
TDO	0 0 0 0 1 0 1 0	0 A	A, SL → O REGISTER			4-10.3
TKA	0 0 0 0 1 0 0 0	0 8	$K_{8,4,2,1}$ → A			4-9.2
TMA	0 0 1 0 0 0 0 1	2 1	M(X, Y) → A			4-3.5
TMY	0 0 1 0 0 0 1 0	2 2	M(X, Y) → Y			4-3.4
TYA	0 0 1 0 0 0 1 1	2 3	Y → A			4-2.2
XMA	0 0 0 0 0 0 1 1	0 3	M(X, Y) ↔ A			4-3.6
YNEA	0 0 0 0 0 0 1 0	0 2	Y ≠ A, S → SL		Y	4-6.2
YNEC	0 1 0 1 C	5 –	Y ≠ C		Y	4-6.3

* Opcodes 70 through 7E perform the instructions having the following mnemonics: IAC, DAN, A2AAC, A3AAC, A4AAC, A5AAC, A6AAC, A7AAC, A8AAC, A9AAC, A10AAC, A11AAC, A12AAC, A13AAC, A14AAC. See Figure 7-1.

TABLE 7-3 TMS 1100/1300 MICROINSTRUCTION INDEX

Mnemonic	Opcode	Microinstructions Fixed	Microinstructions Programmable
AC1AC	0 1 1 1 C		CKP,ATN,CIN,C8,AUTA
ALEM	0 0 0 0 0 0 0 1		MTP,NATN,CIN,C8
AMAAC	0 0 0 0 0 1 1 0		ATN,MTP,C8,AUTA
BR	1 0 W	BR	
CALL	1 1 W	CALL	
CLA	0 1 1 1 1 1 1 1		CKP,CIN,C8,AUTA
COMC	0 0 0 0 1 0 1 1	COMC	
COMX	0 0 0 0 1 0 0 1	COMX	
CPAIZ	0 0 1 1 1 1 0 1		NATN,CIN,C8,AUTA
DMAN	0 0 0 0 0 1 1 1		MTP,15TN,C8,AUTA
DYN	0 0 0 0 0 1 0 0		YTP,15TN,C8,AUTY
IMAC	0 0 1 1 1 1 1 0		MTP,CIN,C8,AUTA
IYC	0 0 0 0 0 1 0 1		YTP,CIN,C8,AUTY
KNEZ	0 0 0 0 1 1 1 0		CKP,NE
LDP	0 0 0 1 C	LDP	
LDX	0 0 1 0 1 F	LDX	
MNEA	0 0 0 0 0 0 0 0		MTP,ATN,NE
MNEZ	0 0 1 1 1 1 1 1		MTP,NE
RBIT	0 0 1 1 0 1 B	RBIT	
RETN	0 0 0 0 1 1 1 1	RETN	
RSTR	0 0 0 0 1 1 0 0	RSTR	
SAMAN	0 0 1 1 1 1 0 0		MTP,NATN,CIN,C8,AUTA
SBIT	0 0 1 1 0 0 B	SBIT	
SETR	0 0 0 0 1 1 0 1	SETR	
TAM	0 0 1 0 0 1 1 1		STO
TAMDYN	0 0 1 0 0 1 0 0		STO,YTP,15TN,C8,AUTY
TAMIYC	0 0 1 0 0 1 0 1		STO,YTP,CIN,C8,AUTY
TAMZA	0 0 1 0 0 1 1 0		STO,AUTA
TAY	0 0 1 0 0 0 0 0		ATN,AUTY
TBIT1	0 0 1 1 0 B		CKP,CKN,MTP,NE
TCY	0 1 0 0 C		CKP,AUTY
TCMIY	0 1 1 0 C		CKM,YTP,CIN,AUTY
TDO	0 0 0 0 1 0 1 0	TDO	
TKA	0 0 0 0 1 0 0 0		CKP,AUTA
TMA	0 0 1 0 0 0 0 1		MTP,AUTA
TMY	0 0 1 0 0 0 1 0		MTP,AUTY
TYA	0 0 1 0 0 0 1 1		YTP,AUTA
XMA	0 0 0 0 0 0 1 1		MTP,STO,AUTA
YNEA	0 0 0 0 0 0 1 0		YTP,ATN,NE,STSL
YNEC	0 1 0 1 C		YTP,CKN,NE

*The same programmable microinstructions perform the following instructions having opcodes 70 through 7E: IAC, DAN, A2AAC, A3AAC, A4AAC, A5AAC, A6AAC, A7AAC, A8AAC, A9AAC, A10AAC, A11AAC, A12AAC, A13AAC, A14AAC. Note the PLA Diagram, Figure 6-5.3.

TABLE 7-4 TMS 1100/1300 BINARY INSTRUCTION LIST

Opcode Binary List	Opcode Hex	Mnemonic	Action	Status C8	Status NE	Reference Paragraph
0 0 0 0 0 0 0 0	0 0	MNEA	M (X,Y) ≠ A		Y	8-5
0 0 0 0 0 0 0 1	0 1	ALEM	A ≤ M(X,Y)	Y		4-5.1
0 0 0 0 0 0 1 0	0 2	YNEA	Y ≠ A; S → SL		Y	4-6.2
0 0 0 0 0 0 1 1	0 3	XMA	M (X,Y) ↔ A	Y		4-3.6
0 0 0 0 0 1 0 0	0 4	DYN	Y − 1 → Y	Y		4-4.8
0 0 0 0 0 1 0 1	0 5	IYC	Y + 1 → Y			4-4.6
0 1 1 1 1 1 1 1	7 F	CLA	0 → A	Y		4-2.3
0 0 0 0 0 1 1 1	0 7	DMAN	M (X,Y) − 1 → A			4-4.4
0 0 0 0 1 0 0 0	0 8	TKA	$K_{8,4,2,1}$ → A			4-9.2
0 0 0 0 1 0 0 1	0 9	COMX	$\overline{X}_{MSB}$ → X_{MSB}			8-7.2
0 0 0 0 1 0 1 0	0 A	TDO	A, SL → O REGISTER			4-10.3
0 0 0 0 1 0 1 1	0 B	COMC	$\overline{CB}$ → CB			8-8.4
0 0 0 0 1 1 0 0	0 C	RSTR	0 → R(Y)			8-6.2
0 0 0 0 1 1 0 1	0 D	SETR	1 → R(Y)			8-6.1
0 0 0 0 1 1 1 0	0 E	KNEZ	$K_{8,4,2,1}$ ≠ 0		Y	4-9.1
0 0 0 0 1 1 1 1	0 F	RETN	CL = 1: SR → PC; PB → PA; CS → CA; 0 → CL CL = 0: PC + 1 → PC; PB → PA			8-8.3
0 0 0 1 C	1 –	LDP	C → PB			4-12.4
0 0 1 0 0 0 0 0	2 0	TAY	A → Y			4-2.1
0 0 1 0 0 0 0 1	2 1	TMA	M(X,Y) → A			4-3.5
0 0 1 0 0 0 1 0	2 2	TMY	M(X,Y) → Y			4-3.4
0 0 1 0 0 0 1 1	2 3	TYA	Y → A			4-2.2
0 0 1 0 0 1 0 0	2 4	TAMDYN	A → M(X,Y); Y − 1 → Y	Y		8-3.2
0 0 1 0 0 1 0 1	2 5	TAMIYC	A → M(X,Y); Y + 1 → Y	Y		8-3.1
0 0 1 0 0 1 1 0	2 6	TAMZA	A → M(X,Y); 0 → A			4-3.3
0 0 1 0 0 1 1 1	2 7	TAM	A → M(X,Y)			4-3.1
0 0 1 0 1 F	2 –	LDX	C → X			8-7.1
0 0 1 1 0 0 B	3 –	SBIT	1 → M(X,Y,B)			4-7.1
0 0 1 1 0 1 B	3 –	RBIT	0 → M(X, Y, B)			4-7.2
0 0 1 1 1 0 B	3 –	TBIT1	M(X,Y,B) = 1		Y	4-7.3
0 0 1 1 1 1 0 0	3 C	SAMAN	M(X,Y) −A → A	Y		4-4.2
0 0 1 1 1 1 0 1	3 D	CPAIZ	$\overline{A}$ + 1 → A	Y		4-4.12
0 0 1 1 1 1 1 0	3 E	IMAC	M(X,Y) + 1 → A	Y		4-4.3
0 0 1 1 1 1 1 1	3 F	MNEZ	M(X,Y) ≠ 0		Y	4-6.1
0 1 0 0 C	4 –	TCY	C → Y			4-8.1
0 1 0 1 C	5 –	YNEC	Y ≠ C		Y	4-6.3
0 1 1 0 C	6 –	TCMIY	C → M (X,Y); Y + 1 → Y			4-6.2
0 0 0 0 0 1 1 0	0 6	AMAAC	M(X,Y) + A → A	Y		4-4.1
0 1 1 1 C	7 –	AC1AC*	A + C + 1 → 1	Y		8-4
1 0 W	– –	BR	S = 1, CL = 0: CB → CA; PB → PA; I(W) → PC S = 1, CL = 1: CB → CA; I(W) → PC S = 0, CL = 1 OR 0: PC + 1 → PC; 1 → S			8-8.1
1 1 W	– –	CALL	S = 1, CL = 0: CA → CS; CB → CA; PB ↔ PA; PC + 1 → SR; I(W) → PC; 1 → CL S = 1, CL = 1: CB → CA; PA → PB; I(W) → PC $\overline{S}$=0, CL=1 OR 0: PC + 1 → PC; 1 → S			8-8.2

*Opcodes 70 through 7E perform IAC, DAN, A2AAC, A3AAC, A4AAC, A5AAC, A6AAC, A7AAC, A8AAC, A9AAC, A10AAC, A11AAC, A12AAC, A13AAC, and A14AAC.

MACHINE INSTRUCTION CODE

<table>
<tr><td colspan="4">A</td><td colspan="4">Z</td></tr>
<tr><td>I(0)</td><td>I(1)</td><td>I(2)</td><td>I(3)</td><td>I(4)</td><td>I(5)</td><td>I(6)</td><td>I(7)</td></tr>
<tr><td>MSB</td><td></td><td></td><td></td><td></td><td></td><td></td><td>LSB</td></tr>
</table>

<table>
<tr><th>A \ Z</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>A</th><th>B</th><th>C</th><th>D</th><th>E</th><th>F</th><th>*OPERAND</th></tr>
<tr><td>0</td><td>MNEA</td><td>ALEM</td><td>YNEA</td><td>XMA</td><td>DYN</td><td>IYC</td><td>AMAAC</td><td>DMAN</td><td>TKA</td><td>COMX</td><td>TDO</td><td>COMC</td><td>RSTR</td><td>SETR</td><td>KNEZ</td><td>RETN</td><td></td></tr>
<tr><td>1</td><td colspan="16">LDP</td><td>C</td></tr>
<tr><td>2</td><td>TAY</td><td>TMA</td><td>TMY</td><td>TYA</td><td>TAM-DYN</td><td>TAM-IYC</td><td>TAM-ZA</td><td>TAM</td><td colspan="8">LDX</td><td>F</td></tr>
<tr><td>3</td><td colspan="4">SBIT</td><td colspan="4">RBIT</td><td colspan="4">TBIT1</td><td>SAMAN</td><td>CPAIZ</td><td>IMAC</td><td>MNEZ</td><td>B</td></tr>
<tr><td>4</td><td colspan="16">TCY</td><td>C</td></tr>
<tr><td>5</td><td colspan="16">YNEC</td><td>C</td></tr>
<tr><td>6</td><td colspan="16">TCMIY</td><td>C</td></tr>
<tr><td>7</td><td>IAC</td><td>A9AAC</td><td>A5AAC</td><td>A13AAC</td><td>A3AAC</td><td>A11AAC</td><td>A7AAC</td><td>DAN</td><td>A2AAC</td><td>A10AAC</td><td>A6AAC</td><td>A14AAC</td><td>A4AAC</td><td>A12AAC</td><td>A8AAC</td><td>CLA</td><td></td></tr>
<tr><td>8</td><td colspan="16" rowspan="4">BR</td><td rowspan="4">W</td></tr>
<tr><td>9</td></tr>
<tr><td>A</td></tr>
<tr><td>B</td></tr>
<tr><td>C</td><td colspan="16" rowspan="4">CALL</td><td rowspan="4">W</td></tr>
<tr><td>D</td></tr>
<tr><td>E</td></tr>
<tr><td>F</td></tr>
</table>

*C = constant; B = bit address; W = memory address; F = file address

FIGURE 7-1 TMS 1100/1300 STANDARD INSTRUCTION MAP

SECTION VIII

TMS 1100/1300 STANDARD-INSTRUCTION DESCRIPTION

8-1 GENERAL.

There are 40 basic TMS 1100/1300 instructions in the standard-instruction set. Tables 7-1 through 7-4 summarized the instruction set in various ways. The instruction values are different from the TMS 1000/1200 and there are new instructions added.

8-2 TMS 1000/1200 vs TMS 1000/1300 INSTRUCTIONS.

8-2.1 DIFFERENCES IN DEFINITION. The load-X-register instruction (LDX) has a larger operand field creating a new format (V) for that instruction. COMX affects the MSB only in the TMS 1100/1300 (in the TMS 1000/1200 COMX complemented the entire X-register contents).

The complement-chapter instruction (COMC) displaces the TMS 1000/1200's CLO instruction.

The add-constant plus-one-to accumulator (with carry → status), AC1AC_, will perform the A6/8/10AAC, IA, and DAN TMS 1000/1200 instructions. The increment-accumulator instruction (IA) in the TMS 1000/1200 is now performed by AC1AC 0 which sends a zero to status if there is no carry. Decrement accumulator (DAN) is replaced by AC1AC 14. The ALEC instruction is not included in the TMS 1100/1300 instruction set; therefore, the AC1AC must be used. In this case the operand contains the two's complement of the ALEC operand and subtraction is the effect. Note, however, that each time AC1AC is performed, the results are stored in the accumulator. For this reason instructions saving the accumulator temporarily and restoring the contents may be needed.

The TAMIYC replaces the TAMIY instruction, allowing loop control with the carry information.

The MNEA and TAMDYN are new instructions in the TMS 1100/1300 repertoire.

Table 8-2.1 shows a cross-reference for the TMS 1100/1300 and TMS 1000/1200 instructions. The shaded areas indicate either changes in opcodes, mnemonics, or actions.

TABLE 8-2.1 TMS 1000 SERIES INSTRUCTION CROSS REFERENCE

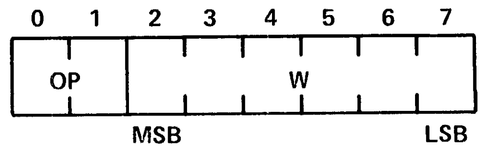
TMS 1100/1300				TMS 1000/1200	
Mnemonic	Opcode Binary	Opcode Hex	Action	Mnemonic	Opcode Hex
AC1AC*	0 1 1 1 C	7 –	A + C + 1 → A	ALEC*	7 –
ALEM	0 0 0 0 0 0 0 1	0 1	A ≤ M(X,Y)	ALEM	2 9
AMAAC	0 0 0 0 0 1 1 0	0 6	M(X,Y) + A → A	AMAAC	2 5
BR	1 0 W	—	S = 1, CL = 0: CB → CA; PB → PA; I(W) → PC S = 1, CL = 1: CB → CA; I(W) → PC **S = 0, CL = 1 OR 0**: PC + 1 → PC; 1 → S	BR	—
CALL	1 1 W	—	S = 1, CL = 0: CA → CS; CB → CA; PB ↔ PA; PC + 1 → SR; I(W) → PC; 1 → CL S = 1, CL = 1: CB → CA; PA → PB; I(W) → PC **S = 0, CL = 1 OR 0**: PC + 1 → PC; 1 → S	CALL	—
CLA	0 1 1 1 1 1 1 1	7 F	0 → A	CLA	2 F
COMC	0 0 0 0 1 0 1 1	0 B	$\overline{CB}$ → CB	CLO**	0 B
COMX	0 0 0 0 1 0 0 1	0 9	$\overline{X}_{MSB} \rightarrow X_{MSB}$	COMX	0 0
CPAIZ	0 0 1 1 1 1 0 1	3 D	$\overline{A}$ + 1 → A	CPAIZ	2 D
DMAN	0 0 0 0 0 1 1 1	0 7	M(X,Y) − 1 → A	DMAN	2 A
DYN	0 0 0 0 0 1 0 0	0 4	Y − 1 → Y	DYN	2 C
IMAC	0 0 1 1 1 1 1 0	3 E	M(X,Y) + 1 → A	IMAC	2 8
IYC	0 0 0 0 0 1 0 1	0 5	Y + 1 → Y	IYC	2 B
KNEZ	0 0 0 0 1 1 1 0	0 E	$K_{8,4,2,1} \neq 0$	KNEZ	0 9
LDP	0 0 0 1 C	1 –	C → PB	LDP	1 –
LDX	0 0 1 0 1 F	2 –	F → X	LDX	3 –
MNEA	0 0 0 0 0 0 0 0	0 0	M(X,Y) ≠ A		
MNEZ	0 0 1 1 1 1 1 1	3 F	M(X,Y) ≠ 0	MNEZ	2 6
RBIT	0 0 1 1 0 1 B	3 –	0 → M(X,Y,B)	RBIT	3 –
RETN	0 0 0 0 1 1 1 1	0 F	CL = 1: SR → PC; PB → PA; CS → CA; 0 → CL CL = 0: PC + 1 → PC; PB → PA	RETN	0 F
RSTR	0 0 0 0 1 1 0 0	0 C	0 → R(Y)	RSTR	0 C
SAMAN	0 0 1 1 1 1 0 0	3 C	M(X,Y) − A → A	SAMAN	2 7
SBIT	0 0 1 1 0 0 B	3 –	1 → M(X,Y,B)	SBIT	3 –
SETR	0 0 0 0 1 1 0 1	0 D	1 → R(Y)	SETR	0 D
TAM	0 0 1 0 0 1 1 1	2 7	A → M(X,Y)	TAM	0 3
TAMDYN	0 0 1 0 0 1 0 0	2 4	A → M(X,Y); Y − 1 → Y		
TAMIYC	0 0 1 0 0 1 0 1	2 5	A → M(X,Y); Y + 1 → Y	TAMIY	2 0
TAMZA	0 0 1 0 0 1 1 0	2 6	A → M(X,Y); 0 → A	TAMZA	0 4
TAY	0 0 1 0 0 0 0 0	2 0	A → Y	TAY	2 4
TBIT1	0 0 1 1 1 0 B	3 –	M(X,Y,B) = 1	TBIT1	3 –
TCY	0 1 0 0 C	4 –	C → Y	TCY	4 –
TCMIY	0 1 1 0 C	6 –	C → M(X,Y); Y + 1 → Y	TCMIY	6 –
TDO	0 0 0 0 1 0 1 0	0 A	A, SL → O REGISTER	TDO	0 A
TKA	0 0 0 0 1 0 0 0	0 8	$K_{8,4,2,1}$ → A	TKA	0 8
TMA	0 0 1 0 0 0 0 1	2 1	M(X,Y) → A	TMA	2 1
TMY	0 0 1 0 0 0 1 0	2 2	M(X,Y) → Y	TMY	2 2
TYA	0 0 1 0 0 0 1 1	2 3	Y → A	TYA	2 3
XMA	0 0 0 0 0 0 1 1	0 3	M(X,Y) ↔ A	XMA	2 E
YNEA	0 0 0 0 0 0 1 0	0 2	Y ≠ A, S → SL	YNEA	0 2
YNEC	0 1 0 1 C	5 –	Y ≠ C	YNEC	5 –

*ALEC replaced by AC1AC. AC1AC is synonymous with DAN, IAC, A6AAC, A8/10AAC and both are accepted by the TMS 1100 and 1300 assembler.

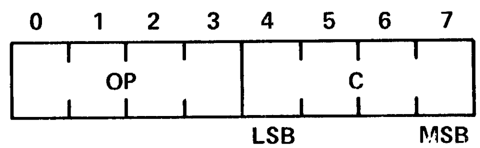
**CLO replaced by COMC.

8-2.2 **INSTRUCTION FORMATS.** Format V is a new TMS 1100/1300 format. Format V, used for LDX, has a three-bit operand because the X-register contains three bits (see paragraph 8-5.1).

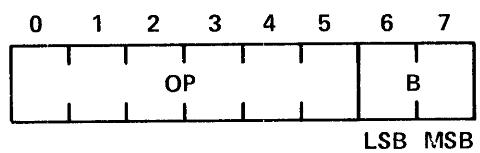
Instruction Format I:



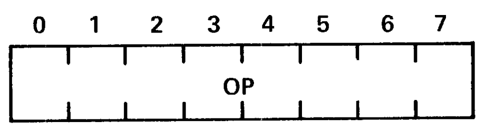
Instruction Format II:



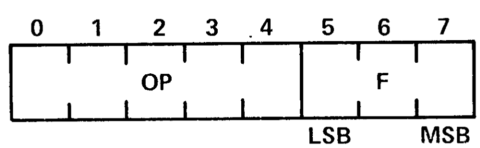
Instruction Format III:



Instruction Format IV:



Instruction Format V:



8-3 REGISTER-TO-MEMORY TRANSFER.

The register-to-memory transfer instructions that are unique to the TMS 1100/1300 are the TAMIYC and TAMDYN instructions.

8-3.1 TRANSFER ACCUMULATOR-TO-MEMORY AND INCREMENT Y REGISTER.

MNEMONIC: TAMIYC

0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	1

STATUS: Carry into status

FORMAT: IV

ACTION:

$A \rightarrow M(X,Y)$

$Y + 1 \rightarrow Y$

$1 \rightarrow S$ if $Y = 15$ } Initial conditions

$0 \rightarrow S$ if $Y < 15$ }

PURPOSE: The Y register sequentially addresses a file of 16 RAM words, and the addressed words are set to the accumulator value(s), during initialization for example.

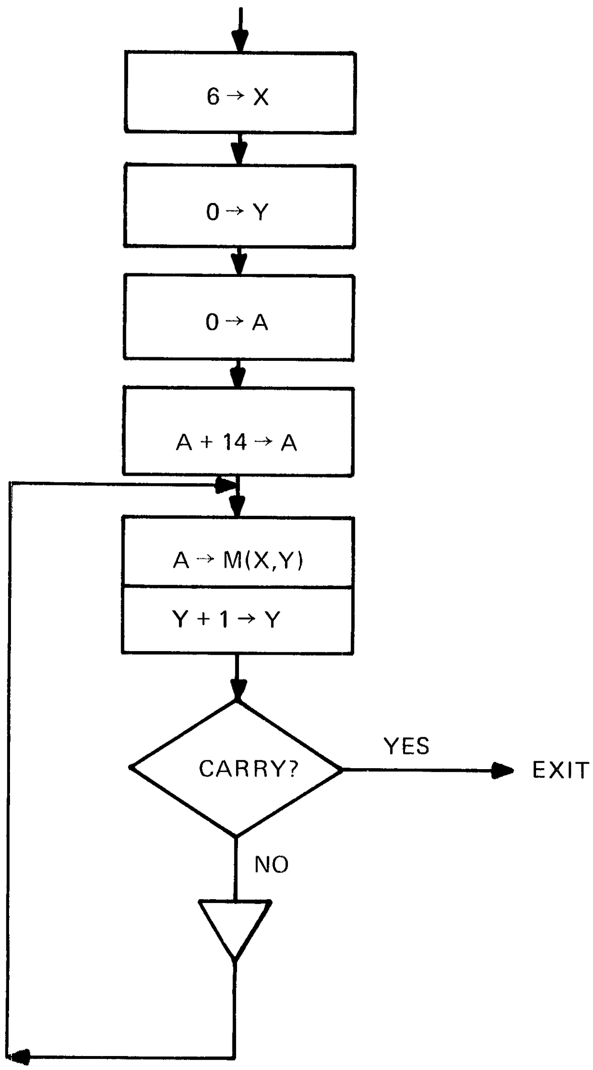
DESCRIPTION: The contents of the accumulator are stored in the memory location addressed by the X and Y registers. The contents of the accumulator are unaltered. Then the contents of the Y register are incremented by one. Carry information is transferred into status. If the result is greater than 15, status is set.

MICROINSTRUCTIONS: STO, YTP, CIN, C8, AUTY

EXAMPLE: The following routine transfers all E's to file six in the TMS 1100 RAM:

			X	Y	A	M(X,Y)	S	BRANCH
	LDX	6	6	2	1	A	1	
	TCY	0	6	0	1	A	1	
	CLA		6	0	0	A	1	
	A14AAC		6	0	E	A	1	
LOOP	TAMIYC		6	0	E	E	0	
			6	1	E	2	0	
	BR	EXIT	6	1	E	2	0	NO
	BR	LOOP	6	1	E	2	1	YES
			6	1	E	E	1	
			6	2	E	7	1	
			6	2	E	7	0	NO
			6	2	E	7	1	YES

⋮



8-3.2 TRANSFER ACCUMULATOR-TO-MEMORY AND DECREMENT Y REGISTER.

MNEMONIC: TAMDYN

0	1	2	3	4	5	6	7
0	0	1	0	0	1	0	0

STATUS: Carry into status

FORMAT: IV

ACTION:

$A \rightarrow M(X, Y)$
$Y - 1 \rightarrow Y$
$\left.\begin{array}{l} 1 \rightarrow S \text{ if } Y \geqslant 1 \\ 0 \rightarrow S \text{ if } Y = 0 \end{array}\right\}$ Initial conditions

PURPOSE: The Y register sequentially addresses a file of 16 RAM words, and the addressed words are set to the accumulator value(s), during initialization for example.

DESCRIPTION: The contents of the accumulator are stored in the memory location addressed by the X and Y registers. The contents of the accumulator are unaltered. Then, the contents of the Y register are decremented by one. Carry information is transferred to status. If the result is not equal to 15, status will be set indicating no borrow.

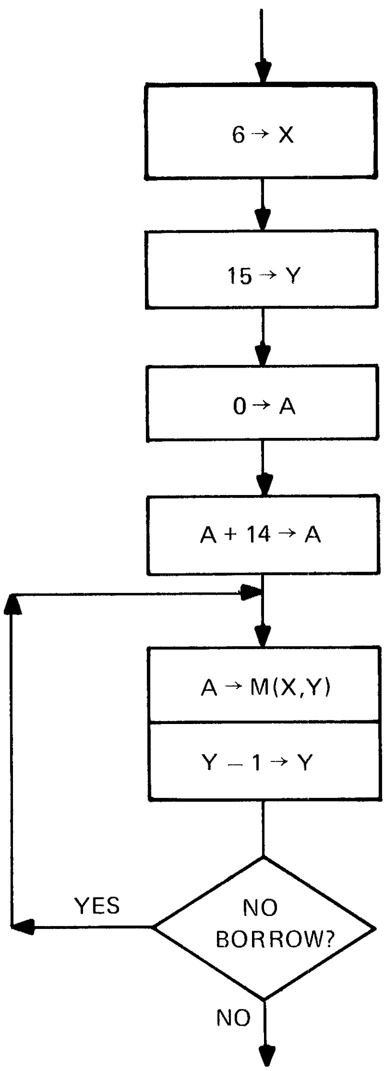
MICROINSTRUCTIONS: STO, YTP, 15TN, C8, AUTY

EXAMPLE: The following routine transfers all E's to file six with one less branch instruction than the previous example:

```
        LDX      6
        TCY      15
        CLA
        A14AAC
LOOP    TAMDYN
        BR       LOOP
```

X	Y	A	M(X,Y)	S	BRANCH
6	2	1	A	1	
6	F	1	3	1	
6	F	0	3	1	
6	F	E	3	1	
6	F	E	E	0	
6	E	E	2	0	
6	E	E	2	1	YES
6	E	E	E	1	
6	D	E	5	1	
6	D	E	5	1	YES

⋮



8-4 ARITHMETIC INSTRUCTIONS.

The A6/8/10AAC and DAN instructions from the TMS 1000/1200 standard-instruction set are included in the TMS 1100/1300 standard-instruction set. The IAC instruction replaces IA and sends carry out to status. A 2/3/4/5/7/9/11/12/13/14 AAC are new instructions for the TMS 1100/1300. All of the accumulator arithmetic instructions are format IV instructions. However, those instructions are replaceable by one format II instruction, AC1AC –. The AC1AC – instruction is used for convenience and is interchangeable with the format IV mnemonics in the source program. The assembler converts either format into the proper opcode.

0	1	2	3	4	5	6	7
0	1	1	1		C		
				LSB			MSB

*MNEMONIC: AC1AC

STATUS: Carry into status

FORMAT: II

OPERAND: Constant value $0 \leqslant I(C) \leqslant 14$

ACTION: $A + C + 1 \rightarrow A$
$1 \rightarrow S$ if sum > 15
$0 \rightarrow S$ if sum $\leqslant 15$

PURPOSE: Used as an alternate mnemonic for various add-immediate-value-to-the-accumulator instructions. See note below.

DESCRIPTION: The C field of the instruction word, I(7-4), is incremented by one and added to the accumulator contents. The result is placed back into the accumulator. The resulting carry information transfers to status. A result greater than 15 sets status.

MICROINSTRUCTIONS: CKP, ATN, CIN, C8, AUTA

*Note: The AC1AC instructions have the equivalent format IV mnemonics enabled by the default instruction definitions in the TMS 1100/1300 assembler:

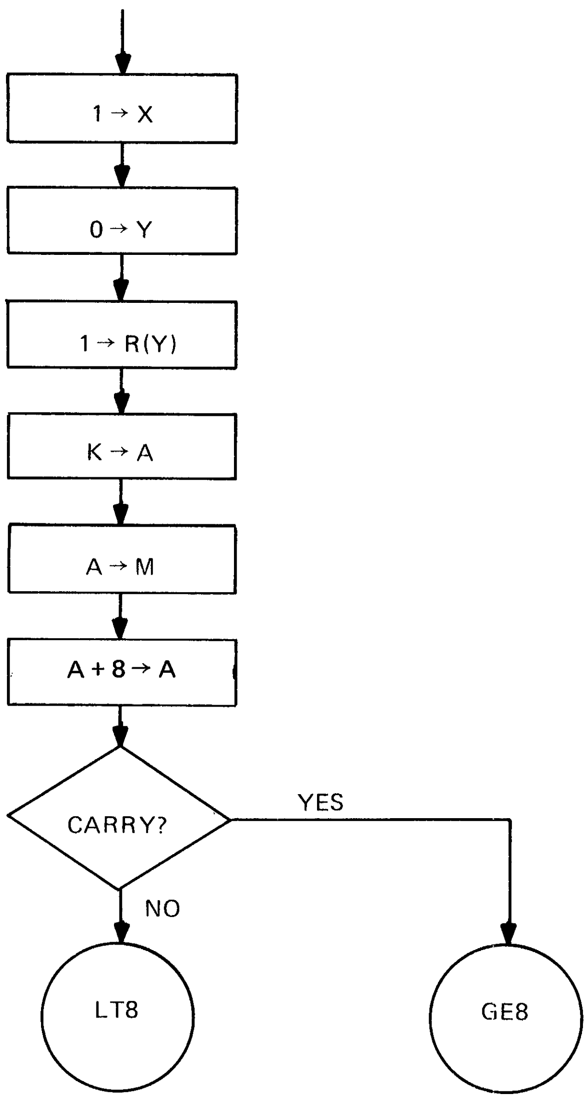
MNEMONICS			OP CODE
FORMAT II		FORMAT IV	
OP CODE	I(C)		HEXADECIMAL
AC1AC	0	IAC	70
AC1AC	1	A2AAC	78
AC1AC	2	A3AAC	74
AC1AC	3	A4AAC	7C
AC1AC	4	A5AAC	72
AC1AC	5	A6AAC	7A
AC1AC	6	A7AAC	76
AC1AC	7	A8AAC	7E
AC1AC	8	A9AAC	71
AC1AC	9	A10AAC	79
AC1AC	10	A11AAC	75
AC1AC	11	A12AAC	7D
AC1AC	12	A13AAC	73
AC1AC	13	A14AAC	7B
AC1AC	14	DAN	77
(ILLEGAL)	15	CLA	7F

EXAMPLES: Refer to paragraph 12-3 and 12-4 for addition and subtraction in BCD.

The following example shows a input program which tests the K-input data for a value greater than or equal to eight. The K-input data is valid when R0 is set and the data is loaded into M(1,0) for permanent storage.

		X	Y	A	M(X,Y)	S	BRANCH	K
LDX	1	1	A	0	4	1		0
TCY	0	1	0	0	2	1		0
SETR		1	0	0	2	1		C
TKA		1	0	C	2	1		C
TAM		1	0	C	C	1		C
A8AAC		1	0	4	C	1		C
BR	GE8	1	0	4	C	1	YES	C

LT8



8-5 LOGICAL COMPARE.

A new logical-compare instruction allows the user to test the RAM contents against the accumulator data.

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

MNEMONIC: MNEA

STATUS: Comparison result into status

FORMAT: IV

ACTION:
$M(X,Y) \neq A$?
$1 \rightarrow S$ if $M(X,Y) \neq A$
$0 \rightarrow S$ if $M(X,Y) = A$

PURPOSE: To compare the RAM data with the accumulator contents.

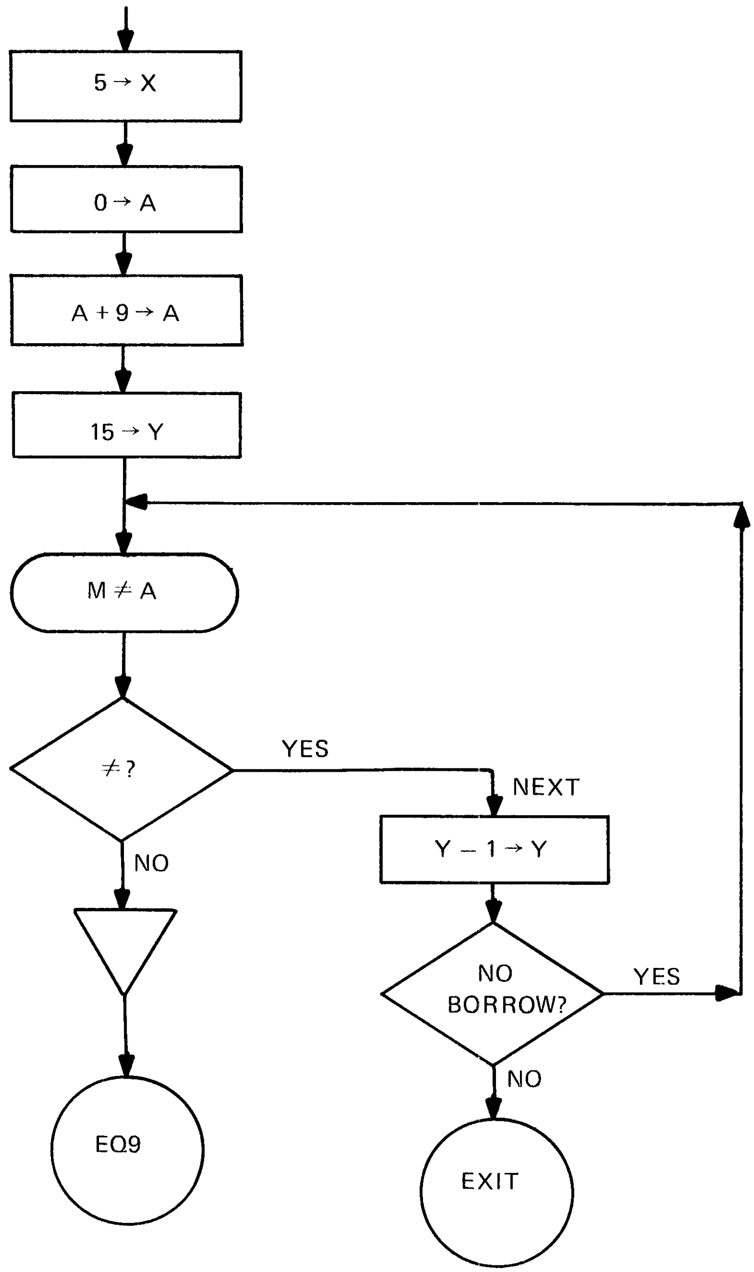
DESCRIPTION: The contents of the memory addressed by the X and Y registers are logically compared to the accumulator contents. The comparison information is transferred to status. Inequality will set status.

MICROINSTRUCTIONS: MTP, ATN, NE

EXAMPLE: The following routine searches file five for the first word equal to nine.

```
        LDX     5
        CLA
        A9AAC
        TCY     15
LOOP    MNEA
        BR      NEXT
        BR      EQ9
NEXT    DYN
        BR      LOOP
EXIT
```

X	Y	A	M(X,Y)	S	BRANCH
5	0	F	3	1	
5	0	0	3	1	
5	0	9	3	1	
5	F	9	9	1	
5	F	9	9	1	
5	F	9	9	0	NO
5	F	9	9	1	YES



8-6 OUTPUT INSTRUCTIONS.

The TMS 1100/1300 has one less output instruction (CLO) than the TMS 1000/1200. The operation of the SETR and RSTR commands is possible for $0 \leq Y \leq 15$, $0 \leq X \leq 3$.

8-6.1 SET R-OUTPUT.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	0	1

MNEMONIC: SETR

STATUS: SET

FORMAT: IV

ACTION: $1 \rightarrow R(Y)$

For $0 \leq X \leq 3$;
$0 \leq Y \leq 10$, TMS 1100
$0 \leq Y \leq 15$, TMS 1300

PURPOSE: To set one selected R-output latch to a logic ONE.

DESCRIPTION: The contents of the Y register selects the proper R output. The X register must be less than or equal to three. The Y-register contents is between zero and ten for TMS 1100 applications, and for values greater than ten, the instruction is a no-operation. The full range, $0 \leq Y \leq 15$, can be used in the 40-pin package version, TMS 1300.

FIXED MICROINSTRUCTION: SETR

EXAMPLE: See paragraphs 4-10.5 and 13-3.2.

8-6.2 RESET R-OUTPUT.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	0	0

MNEMONIC: RSTR

STATUS: SET

FORMAT: IV

ACTION: $0 \rightarrow R(Y)$

For $0 \leq X \leq 3$;
$0 \leq Y \leq 10$, TMS 1100
$0 \leq Y \leq 15$, TMS 1300

PURPOSE: To reset one selected R-output latch to a logic ZERO.

DESCRIPTION: The contents of the Y register selects the proper R output. The X register must be less than or equal to three. The Y-register contents is between zero and ten for TMS 1100 applications, and for values greater than ten, the instruction is a no-operation. The full range, $0 \leqslant Y \leqslant 15$, can be used in the 40-pin package version, TMS 1300.

FIXED MICROINSTRUCTION: RSTR

EXAMPLE: See paragraphs 4-10.5 and 13-3.2.

8-7 RAM X ADDRESSING.

The RAM addressing is modified to reflect the additional four files in the TMS 1100/1300. Also, the COMX instruction operation affects only the MSB of the X register.

8-7.1 LOAD X REGISTER.

0	1	2	3	4	5	6	7
0	0	1	0	1		F	
					LSB		MSB

MNEMONIC: LDX

STATUS: SET

FORMAT: V

OPERAND: X-file address; $0 \leqslant X \leqslant 7$

ACTION: $I(F) \rightarrow X$

DESCRIPTION: A constant value, I(7-5), is loaded into the X register. This is used to set the X register to the desired RAM file index. The three-bit F-field of the instruction is loaded into the X register.

FIXED MICROINSTRUCTION: LDX

EXAMPLES: See paragraphs 8-3 to 8-5.

8-7.2 COMPLEMENT THE MSB OF X-REGISTER.

0	1	2	3	4	5	6	7
0	0	0	0	1	0	0	1

MNEMONIC: COMX

STATUS: SET

FORMAT: IV

ACTION: $\overline{X}_{MSB} \rightarrow X_{MSB}$

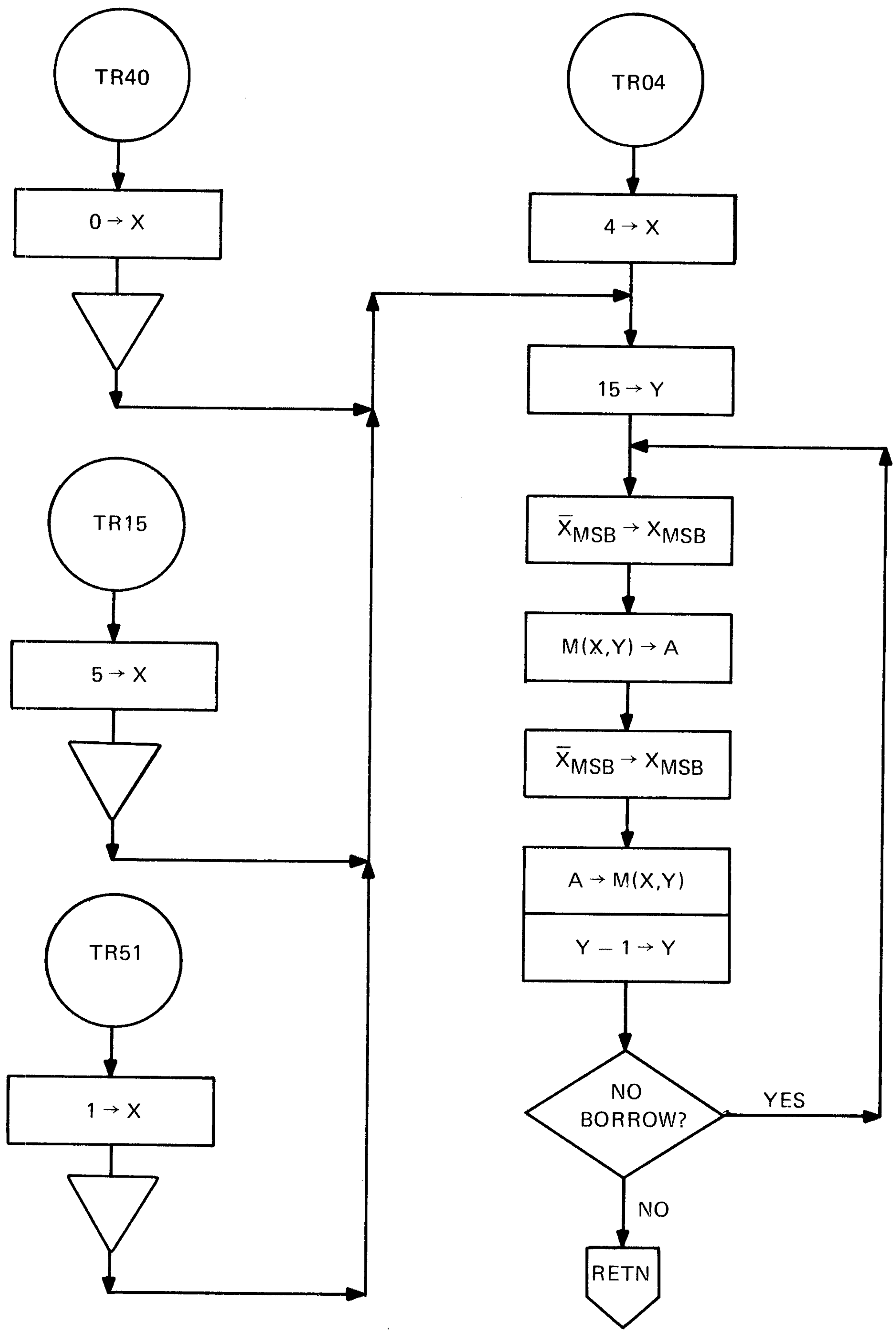
DESCRIPTION: The MSB (most-significant-bit) of the X register is logically complemented.

(0)	000	→	100	(4)
(1)	001	→	101	(5)
(2)	010	→	110	(6)
(3)	011	→	111	(7)
(4)	100	→	000	(0)
(5)	101	→	001	(1)
(6)	110	→	010	(2)
(7)	111	→	011	(3)

FIXED MICROINSTRUCTION: COMX

EXAMPLE: The next example illustrates the power of the COMX instruction (when used with the appropriate LDX instructions). A subroutine has four entry points. Each entry point has a different starting condition for the X register. Four different transfers of multi-precision data are accomplished by the complement-X-register instruction (COMX) which is the only X-addressing within the base subroutine. Since a user often requires data transfer to and from various files, this technique saves ROM instructions by not having to write a different routine for each transfer.

```
LABEL   OPCODE   OPERAND   COMMENT

TR04    LDX      4         ENTRY POINT TO TRANSFER
                           FROM FILE ZERO TO FOUR.

TRANS   TCY      15        INITIALIZE Y TO 15

LOOP    COMX               FLIP THE MSB OF X
        TMA                TRANSFER M(X,Y) TO ACCUMULATOR
        COMX               FLIP THE MSB OF X AGAIN
        TAMDYN             COMPLETE THE TRANSFER OF ONE WORD
        BR       LOOP      BRANCH IF NO BORROW
        RETN

TR40    LDX      0         ENTRY POINT TO TRANSFER
        BR       TRANS     FROM FILE FOUR TO ZERO

TR15    LDX      5         ENTRY POINT TO TRANSFER
        BR       TRANS     FROM FILE ONE TO FIVE

TR51    LDX      1         ENTRY POINT TO TRANSFER
        BR       TRANS     FROM FILE FIVE TO ONE.
```


TR40
0 → X
TR15
5 → X
TR51
1 → X
TR04
4 → X
15 → Y
$\bar{X}_{MSB} \rightarrow X_{MSB}$
M(X,Y) → A
$\bar{X}_{MSB} \rightarrow X_{MSB}$
A → M(X,Y)
Y − 1 → Y
NO BORROW?
YES
NO
RETN

8-8 ROM ADDRESSING.

An additional fixed instruction, COMC, is necessary to change chapters in the TMS 1100/1300. The remaining ROM addressing instructions have appropriate actions to accommodate the additional chapter addressing.

8-8.1 BRANCH, CONDITIONAL ON STATUS.

0	1	2	3	4	5	6	7
1	0			W			

MNEMONIC: BR

STATUS: Condition on status

FORMAT: I

OPERAND: Branch address, I(W)

ACTION:

If S = 1, CL = 0
CB → CA
PB → PA
I(W) → PC

If S = 1, CL = 1
CB → CA
I(W) → PC

If S = 0, CL = 1 or 0
PC + 1 → PC
1 → S

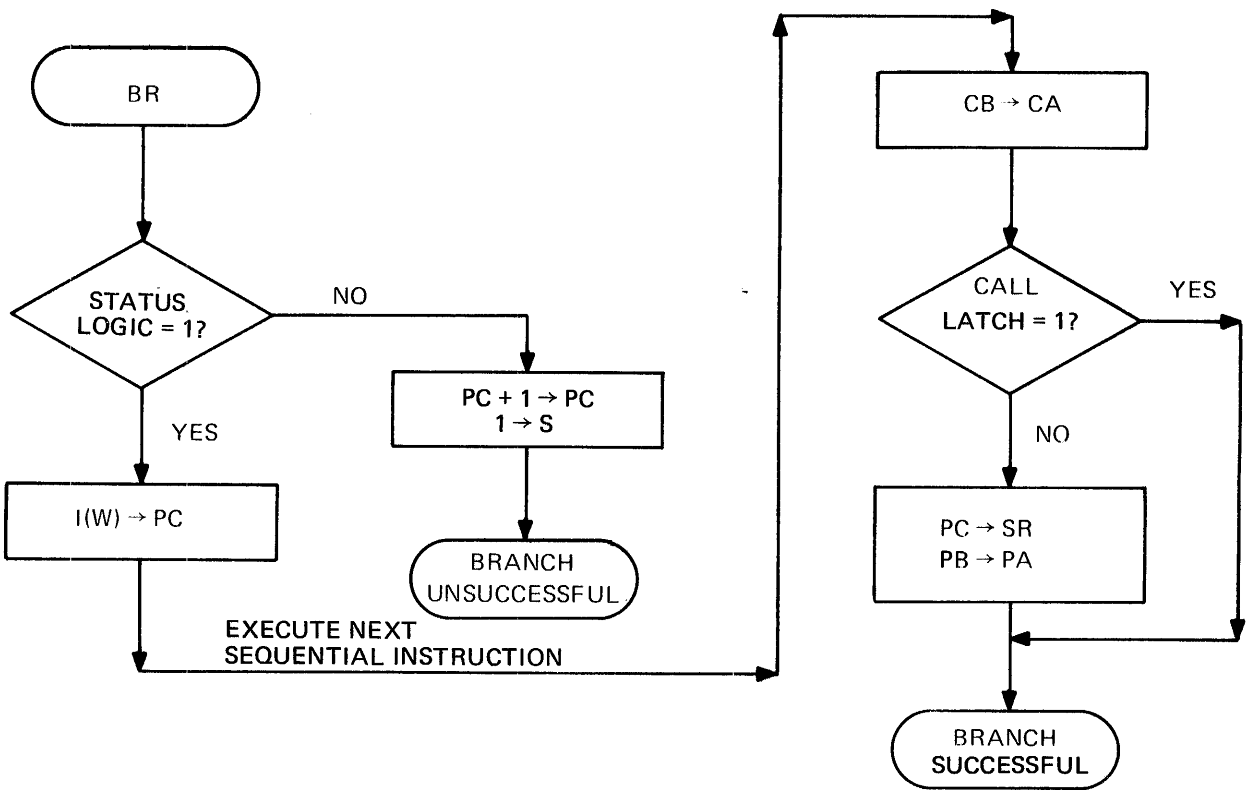
PURPOSE: To alter the normal sequential program execution by the conditional results of the previous instruction.

DESCRIPTION: The branch instruction is always conditional upon the state of status. If status is reset (logical ZERO), then the branch is unsuccessfully executed and the next sequential instruction will be performed. If the status is set (logic ONE), then the branch performs the following actions: the chapter-buffer (CB) bit goes to the chapter-address (CA) latch. The page-buffer (PB) contents transfers into the page-address (PA) register (unless CL = 1). The branch address, W-field, loads into the program counter.

The following standard-symbol flowchart describes the machine operation for the branch instructions that result in the actions listed above.

FIXED MICROINSTRUCTION: BR

EXAMPLE: See paragraphs 4-12 and 8-8.4.



SINGLE INSTRUCTION CYCLE FLOWCHART – BRANCH INSTRUCTION

8-8.2 CALL, CONDITIONAL ON STATUS.

0	1	2	3	4	5	6	7
1	1			W			

MNEMONIC: CALL

STATUS: Conditional on status

FORMAT: I

OPERAND: Branch Address, I(W)

ACTION:

If S = 1, CL = 0
$CA \rightarrow CS$
$CB \rightarrow CA$
$PB \leftrightarrow PA$
$PC + 1 \rightarrow SR$
$I(W) \rightarrow PC$
$1 \rightarrow CL$

If S = 1, CL = 1
$CB \rightarrow CA$
$PA \rightarrow PB$
$I(W) \rightarrow PC$

If S = 0, CL = 1 or 0
$PC + 1 \rightarrow PC$
$1 \rightarrow S$

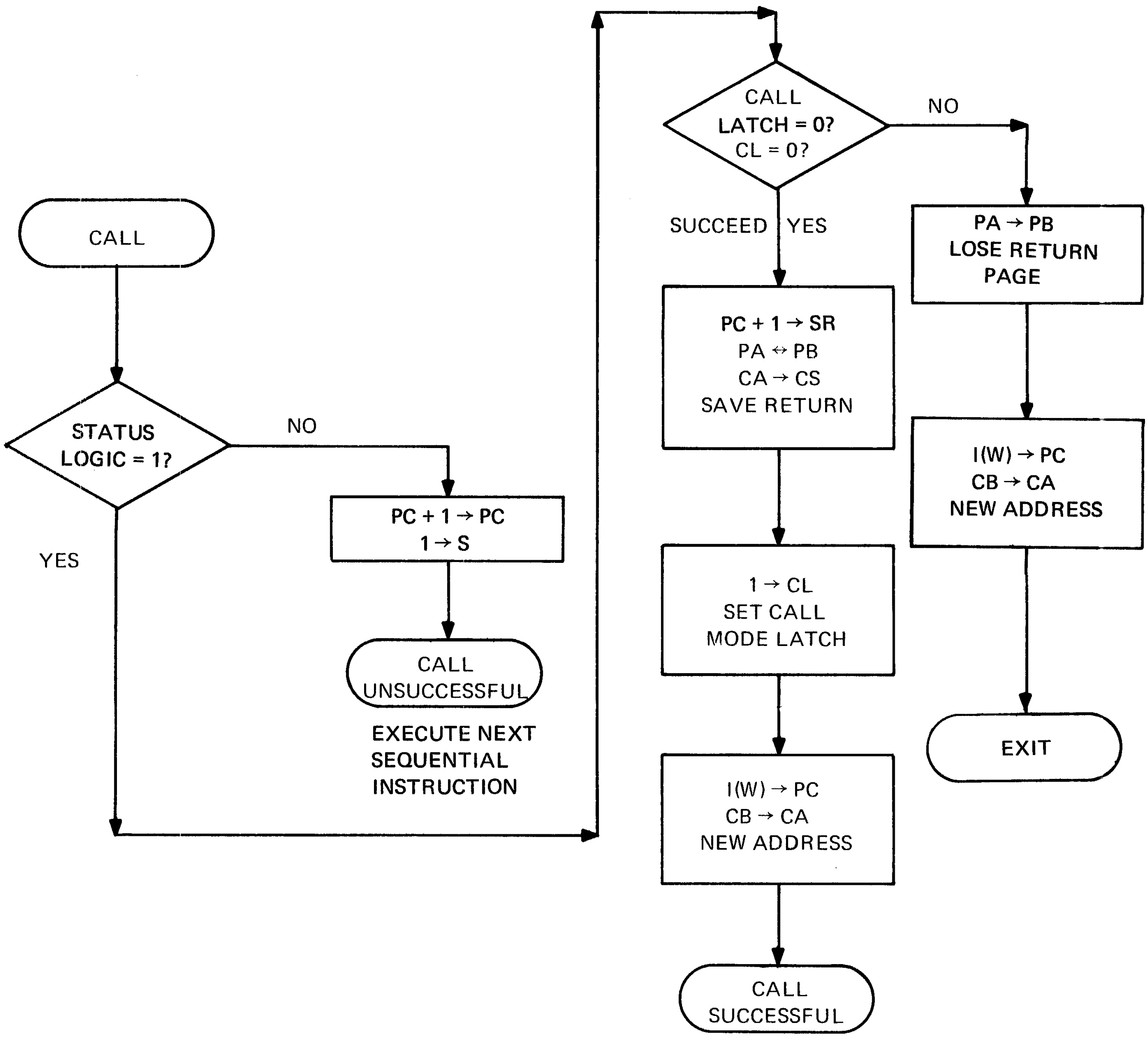
PURPOSE: To allow the program to transfer control to a common subroutine. Because the call instruction saves the return address, subroutines may be called from various locations in a program. A RETN, call return, instruction will restore control back to the instruction after the call that instituted the subroutine.

DESCRIPTION: Call is always conditional upon status. If status is reset, then the call is unsuccessfully executed. If status is set, the following operations occur:

The address of the next instruction is saved in the subroutine-return register (SR). The contents of the page-buffer (PB) and the page-address (PA) registers are exchanged. The chapter-address (CA) bit transfers to the chapter-subroutine (CS) latch. The chapter-buffer (CB) bit goes to the chapter-address (CA) latch. The branch address, W-field, loads into the program counter. The call latch (CL) is set. If the call latch was set by a previous call instruction, the PB-to-PA transfer does not occur; instead, the PA transfers to PB changing the saved page address. Also, the CA will not transfer to CS. The following standard-symbol flowchart describes the machine operations for the call instructions that result in the actions listed above.

FIXED MICROINSTRUCTION: CALL

EXAMPLE: See paragraphs 4-12 and 8-8.4.



SINGLE INSTRUCTION CYCLE FLOWCHART – CALL INSTRUCTION

8-8.3 RETURN FROM SUBROUTINE.

0	1	2	3	4	5	6	7
0	0	0	0	1	1	1	1

MNEMONIC: RETN

STATUS: SET

FORMAT: IV

ACTION:

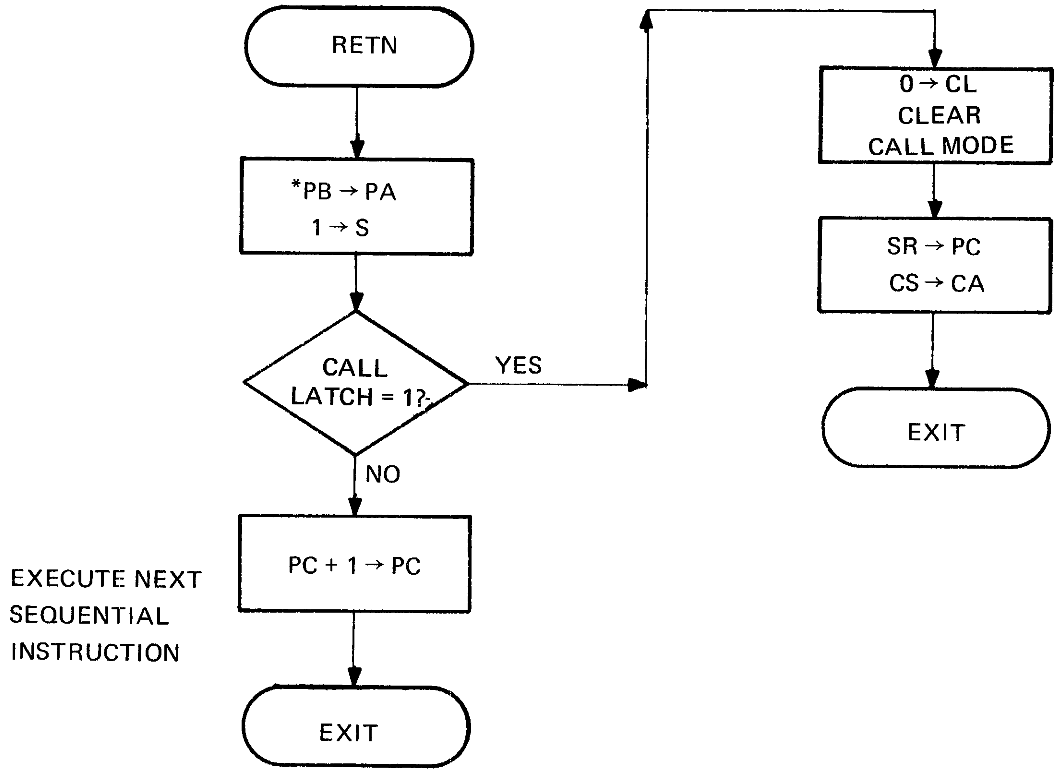
If CL = 1
- SR → PC
- PB → PA
- CS → CA
- 0 → CL

If CL = 0
- PC + 1 → PC
- PB → PA

PURPOSE: To return control from a called subroutine back to the calling program.

DESCRIPTION: The return address is restored. The subroutine-return (SR) register contents transfer to the PC. Simultaneously, the contents of the page-buffer (PB) transfer into the page-address (PA) register. If the call-latch (CL) is ONE, the chapter subroutine (CS) content loads into the chapter address (CA). If the call latch is ZERO, the CS-to-CA transfer does not occur as summarized from the following standard-symbol flowchart.

FIXED MICROINSTRUCTION: RETN



SINGLE INSTRUCTION CYCLE FLOWCHART – RETURN INSTRUCTION

*Note: The page buffer (PB) may contain data other than the return address if a LDP (see paragraph 4-12.4) instruction was executed during the call mode equal to ONE. Also, when not in the call mode the page buffer contents go to the page-address (PA) register, changing the page address as well as updating the program counter.

8-8.4 COMPLEMENT-CHAPTER BUFFER.

0	1	2	3	4	5	6	7
0	0	0	0	1	0	1	1

MNEMONIC: COMC

STATUS: SET

FORMAT: IV

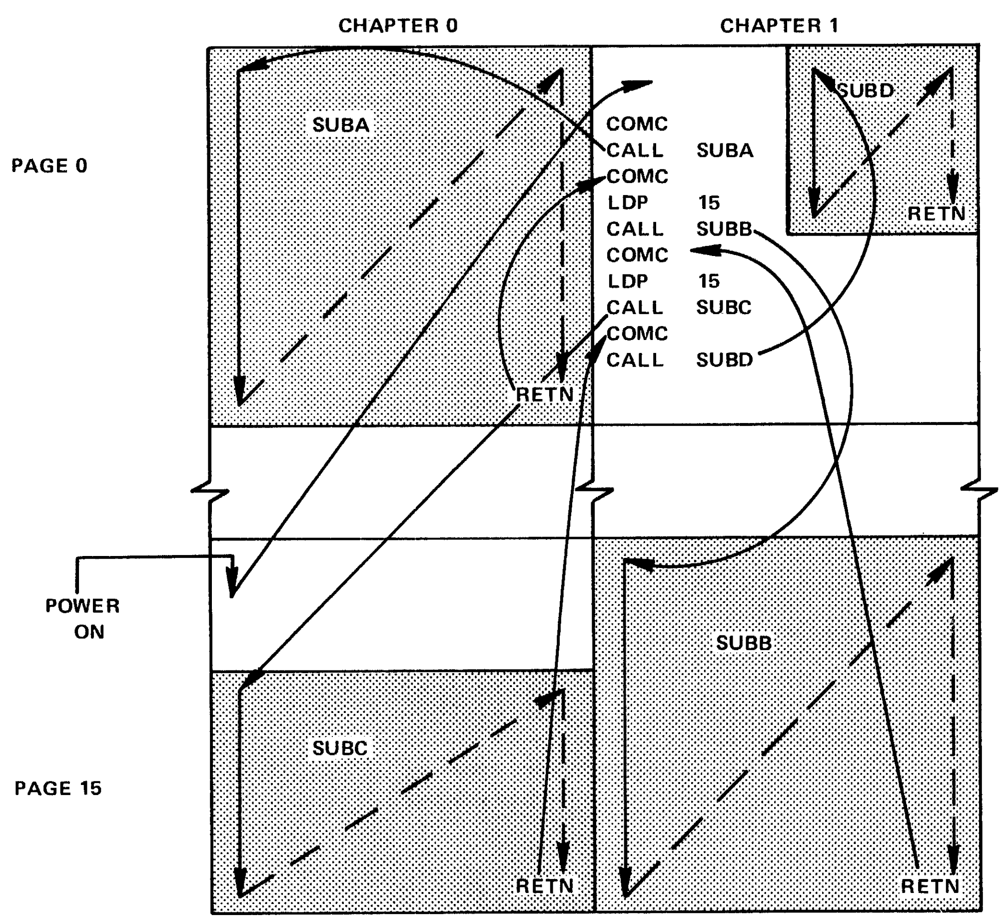
ACTION: $\overline{CB} \rightarrow CB$

PURPOSE: To set up a branch or call to the opposite chapter's ROM address.

DESCRIPTION: The chapter-buffer (CB) bit is complemented logically. Note that the chapter-buffer bit, chapter-address (CA), and chapter-subroutine (CS) bits are reset to ZERO upon application of power.

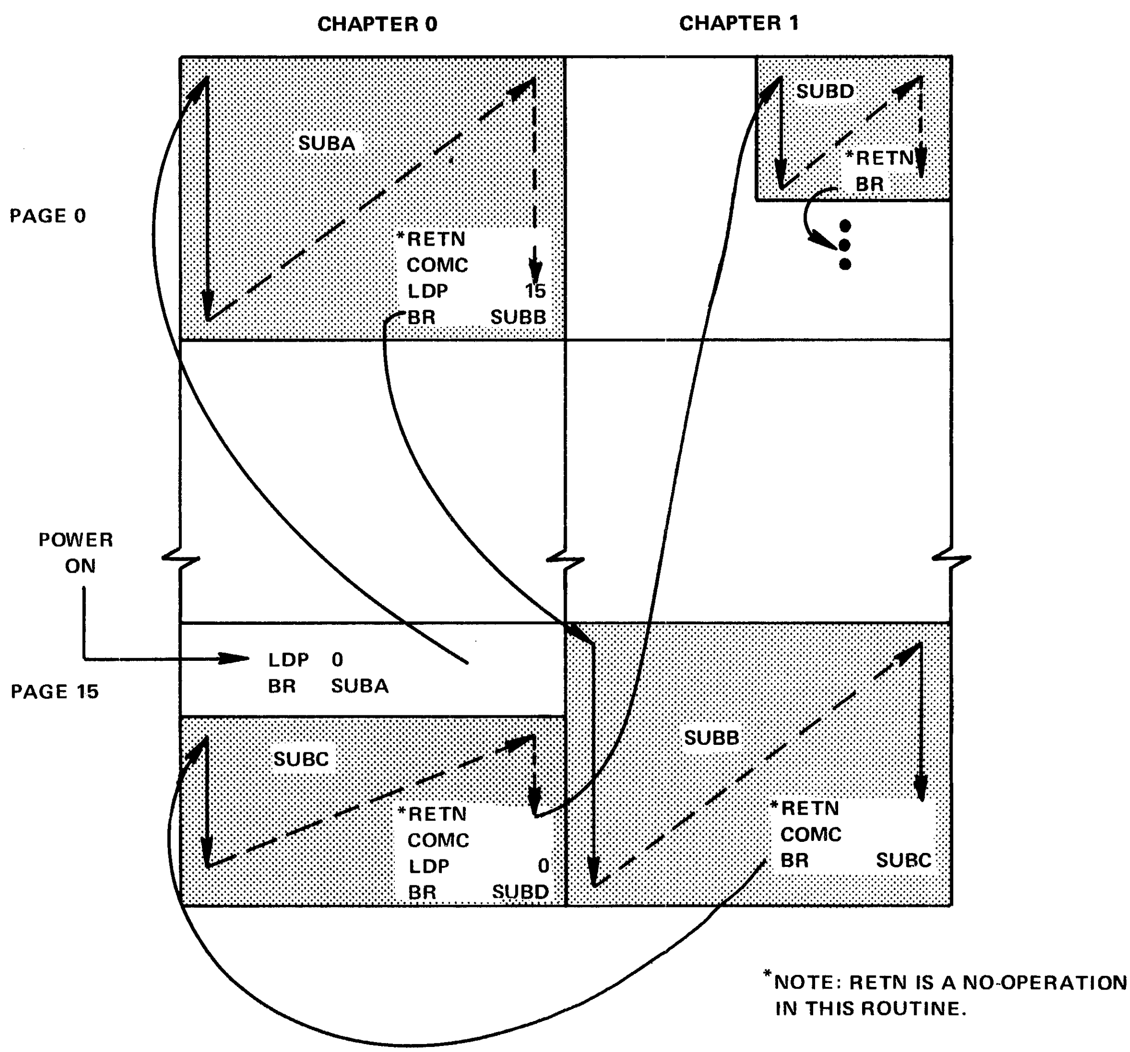
EXAMPLE: A subroutine and a routine are on page one, chapter one. Four subroutines are located at various ROM (see map below) addresses: SUBA, at PA = 0, CA = 0; SUBB, at PA = 15, CA = 1; SUBC, at PA = 15, CA = 0; and SUBD, at PA = 0, CA = 1. The following routine calls the four subroutines in order:

```
PA = 0, CA = 1
  COMC
  CALL     SUBA
  COMC
  LDP      15
  CALL     SUBB
  COMC
  LDP      15
  CALL     SUBC
  COMC
  CALL     SUBD
```



A second routine can call the subrontines conditionally perhaps in a different order. Also, the subroutines can be accessed easily by another main program that branches to the subroutines rather than calling them. In that case, a branch after the RETN location is executed because the RETN acts as a no-operation if CL is ZERO. Thus, the control transfer is also accomplished with branch instructions in conjunction with the any-desired COMC or LDP combination.

FIXED MICROINSTRUCTION: COMC



SECTION IX

MICROPROGRAMMING

9-1 GENERAL.

The smallest quantum of control with which programmers can exercise a computer is the way to describe a microinstruction. As shown in Table 9-1, an instruction is generally considered to consist of one or more microinstructions. A similar definition for a macroinstruction is that one or more instructions constitute a single macroinstruction. A normal programming task requires generation of a ROM-instruction sequence that forms routines or subroutines. An unusual occurence in computer programming tasks is when the instruction set itself is changed; especially since this requires a hardware change for non-microprogrammable machines.

TABLE 9-1 DEFINITION OF TERMS

MACROINSTRUCTION	ONE OR MORE INSTRUCTIONS
INSTRUCTION	SINGLE ROM WORD CONSISTING OF ONE OR MORE MICROINSTRUCTIONS
MICROINSTRUCTIONS	SMALLEST UNIT OF CONTROL OVER SPECIFIC LOGIC BLOCKS
MICROPROGRAMMING	REDEFINING INSTRUCTIONS – CHANGING THE COMBINATIONS OF MICROINSTRUCTIONS USED

In any case, the smallest element of control is derived from a second ROM or PLA in the TMS 1000 series that decodes instructions rather than addresses. Each instruction enables a number of sequenced control lines that affect logic blocks throughout the device. The programmable microinstructions (control lines) which fan out over the device are explained in paragraphs 2-17 and 6-5. Before continuing with this section on microprogramming, paragraphs 2-14 through 2-17 and 6-5 should be read and understood.

The following diagram, Figure 9-1, indicates the procedure for microprogramming. In the case of the assembler, the entire instruction set must be defined. The simulator requires the instruction PLA definition reflecting the desired changes. Software and hardware simulations are available and a new instruction set may be verified by either method. The manufacturing inputs are the ROM-instruction code and the entire PLA definitions. Although the same gate-level mask that fixes the ROM and output PLA also encodes the instruction PLA, the test program (see paragraph 9-3.7) may need modification and any desired microprogramming requires evaluation.

Contact Texas Instruments, Houston, if microprogramming assistance is needed or when significant changes to the instruction set are encountered.

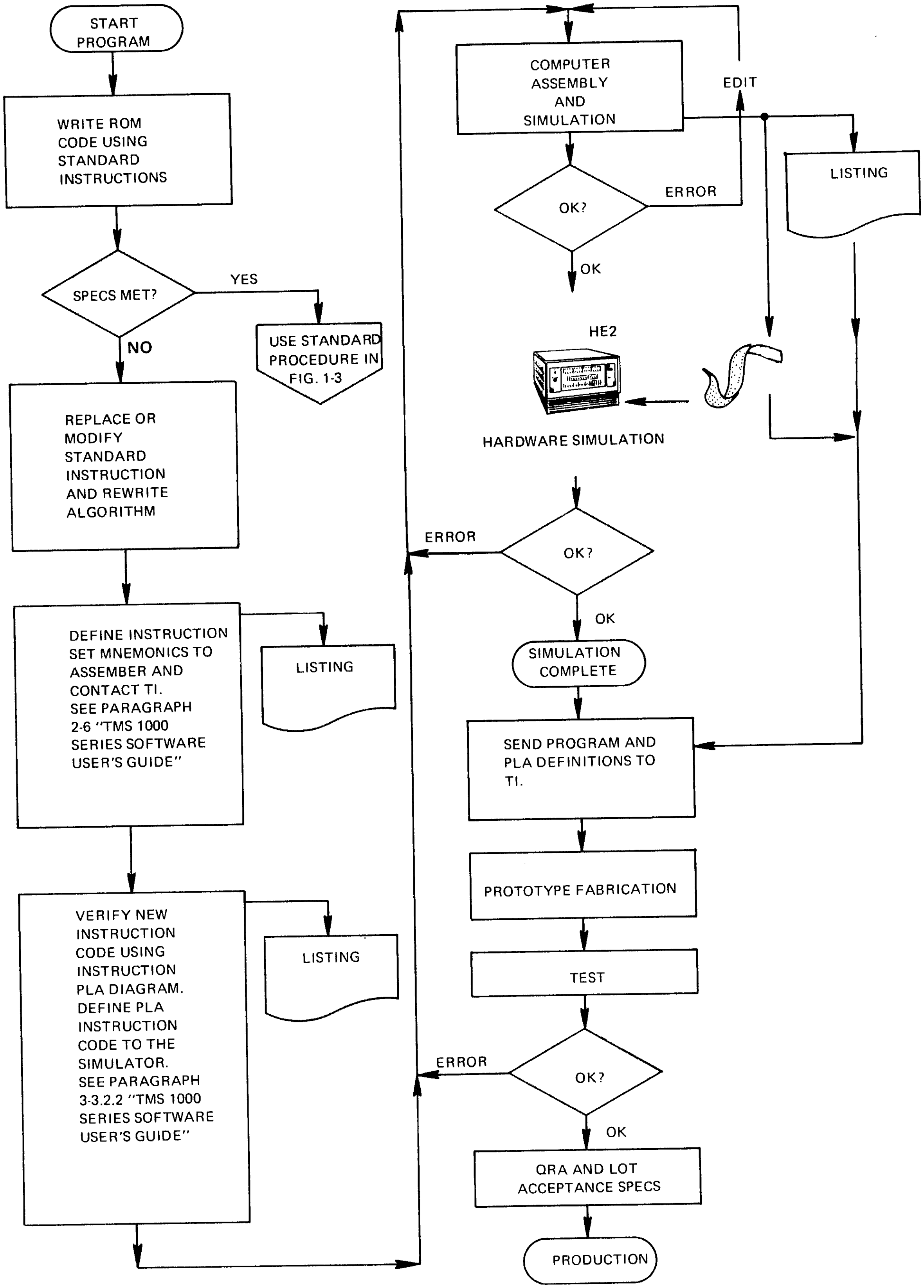


FIGURE 9-1 MICROPROGRAMMING STEPS

9-2 THE INSTRUCTION-PROGRAMMABLE-LOGIC ARRAY.

A ROM generally has a fixed-address decode and programmable-output data is encoded when a single address is enabled. In contrast, a PLA has both programmable decode and encode. Thus, a great deal of versatility exists in a physically limited area. The instruction-PLA inputs come from the instruction bus. Thirty terms, F_i (i = 1 to 30), decode the instruction word and enable some combination of the 16 microinstructions. The following equation describes the possible expressions to be satisified by the instruction-word data.

$$F_i = \prod_{\ell=0}^{7}{}^{*}\ I(\ell), \qquad I(\ell) = 1, 0, X.$$

$$\text{(i.e., MNEZ} = F6 = \overline{I(0)} \cdot \overline{I(1)} \cdot I(2) \cdot I(3) \cdot I(4) \cdot I(5) \cdot I(6) \cdot I(7))$$

In the equation above, the programmer chooses the input variables (1, 0, or X) that describe the desired instruction opcode(s) I(0-7) that will enable a given term, F_i. The 1, 0, or X (don't care) is programmed by a PLA term to decode corresponding data from the instruction memory. A one-to-one correspondence (ignoring the don't-care bits) causes a PLA term to be true. A true term causes the selected combination of microinstructions to go out.

Note that ALU-input controls have negative-true outputs. If a negative-true microinstruction is selected by the programmer, the logical complement (no gate) is placed in the OR-matrix array. Given that Q_j is a microinstruction output, then all selections (gate placement indicated by circles) for the output are logically ORed together:

$$Q_j = \sum_{i=1}^{30}{}^{*}\ F_i \qquad \text{where } j = 1 \text{ to } 16.$$

$$\text{(i.e., STO} = Q_j = F_{11} + F_{12} + F_{13} + F_{24})$$

In most cases only one decoded F_i term is true at a time. So, Q (1-16) corresponds exactly to what is selected by the OR matrix for that term.

If two or more decode terms, F_i, are enabled simultaneously for any given instruction opcode, note the equation above for the negative-true microinstructions; all F_i terms must be ZERO to enable those negative-true outputs (logical OR).

For example, the PLA definitions shown for the TMS 1100/1300 in Figure 9-2.1 and 9-2.2 have two terms enabled for some instructions; this illustrates reducing the number of terms required to describe a particular instruction set. Suppose that the TAMIYC, opcode 25_{16}, instruction is addressed by the program counter. Both terms F_{13} and F_{22} are enabled by opcode 25_{16}. The STO microinstruction is on because F_{13} enables it. The C8 and AUTY microinstructions are enabled by both terms. $\overline{\text{YTP}}$ is ZERO for both terms, and $\overline{\text{CIN}}$ is ZERO for both terms. Thus, YTP, CIN, STO, C8, and AUTY all combine to activate the control necessary to perform TAMIYC. Note that $\overline{\text{15TN}}$ is ONE in term F_{22} and ZERO in term F_{13}:

$$\overline{\text{15TN}} = Q_j = F_{22} + F_{13}$$

$$F_{22} = 1 \qquad F_{13} = 0$$

$$\therefore Q_j = 1 = \overline{\text{15TN}}$$

$$\therefore \text{15TN} = 0$$

The 15TN output is therefore disabled for this instruction by term F_{22}.

---

*Π,Σ, are used in the Boolean sense here.

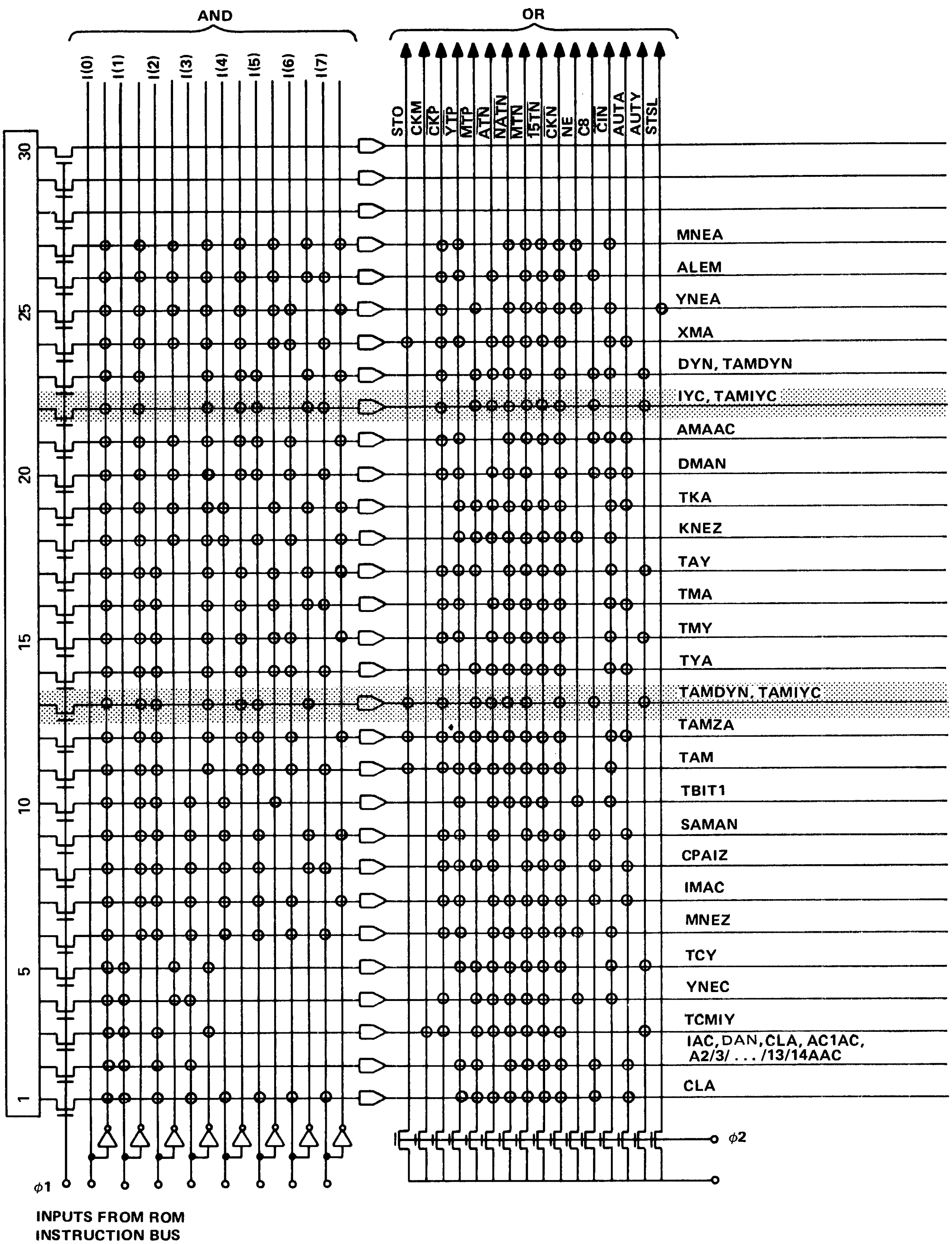


FIGURE 9-2.1 TMS 1100/1300 STANDARD INSTRUCTION PLA

```
OPCPLA
OPX 00=MTP,ATN,NE;                          MNEA
OPX 01=MTP,NATN,CIN,C8;                     ALEM
OPX 02=YTP,ATN,NE,STSL;                     YNEA
OPX 03=STO,MTP,AUTA;                        XMA
OPB 00-00100=YTP,15TN,AUTY,C8;              DYN,TAMDYN
OPB 00-00101=YTP,CIN,AUTY,C8;               IYC,TAMIYC
OPX 06=MTP,ATN,AUTA,C8;                     AMAAC
OPX 07=MTP,15TN,AUTA,C8;                    DMAN
OPX 08=CKP,AUTA;                            TKA
OPX 0E=CKP,NE;                              KNEZ
OPX 20=ATN,AUTY;                            TAY
OPX 21=MTP,AUTA;                            TMA
OPX 22=MTP,AUTY;                            TMY
OPX 23=YTP,AUTA;                            TYA
OPB 0010010-=STO,YTP,15TN,CIN,AUTY,C8       TAMDYN,TAMIYC
OPX 26=STO,AUTA;                            TAMZA
OPX 27=STO,                                 TAM
OPB 001110--=CKP,CKN,MTP,NE;                TBIT1
OPX 3C=MTP,NATN,CIN,AUTA,C8;                SAMAN
OPX 3D=NATN,CIN,AUTA,C8;                    CPAIZ
OPX 3E=MTP,CIN,AUTA,C8;                     IMAC
OPX 3F=MTP,NE;                              MNEZ
OPB 0100----=CKP,AUTY;                      TCY
OPB 0101----=YTP,CKN,NE;                    YNEC
OPB 0110----=CKM,YTP,CIN,AUTY;              TCMIY
OPB 0111----=CKP,ATN,CIN,AUTA,C8;           IAC,DAN,A2/3/. . ./13/14AAC,CLA,AC1AC
OPX 7F=CKP,CIN,AUTA,C8;                     CLA
```

FIGURE 9-2.2 TMS 1100/1300 STANDARD INSTRUCTION PLA CODING

If F_{13} is true and another PLA term enables 15TN and disables CIN, then a new instruction TAMDYN is defined. Term F_{23} does this when decoding opcode 24_{16}.

9-3 MICROPROGRAMMING GUIDELINES.

The next seven paragraphs describe the ground-rules for microprogramming. The information encapsulates ideas presented elsewhere in this manual and brings all the microprogramming facts together for clarity.

9-3.1 FIXED INSTRUCTIONS. The fixed instructions can not be deleted or have their values changed. The shaded areas in Figures 9-3.1 and 9-3.2 indicate the values for the fixed instructions of the TMS 1000/1200 and TMS 1100/1300 respectively. Nonetheless, fixed instructions are programmable in the sense that their operations may be augmented by adding programmable microinstructions to the existing fixed microinstruction. Paragraph 9-4 has two examples of this technique.

MACHINE INSTRUCTION CODE

A				Z			
I(0)	I(1)	I(2)	I(3)	I(4)	I(5)	I(6)	I(7)
MSB							LSB

A \ Z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	*OPERAND
0	COMX	A8AAC	YNEA	TAM	TAMZA	A10AAC	A6AAC	DAN	TKA	KNEZ	TDO	CLO	RSTR	SETR	IA	RETN	
1	LDP																C
2	TAMIY	TMA	TMY	TYA	TAY	AMAAC	MNEZ	SAMAN	IMAC	ALEM	DMAN	IYC	DYN	CPAIZ	XMA	CLA	
3	SBIT				RBIT				TBIT1				LDX				B
4	TCY																C
5	YNEC																C
6	TCMIY																C
7	ALEC																C
8	BR																W
9	BR																W
A	BR																W
B	BR																W
C	CALL																W
D	CALL																W
E	CALL																W
F	CALL																W

*C = constant, B = B field, W = memory address.

FIGURE 9-3.1 TMS 1000/1200 FIXED INSTRUCTION MAP

MACHINE INSTRUCTION CODE

A				Z			
I(0)	I(1)	I(2)	I(3)	I(4)	I(5)	I(6)	I(7)
MSB							LSB

A \ Z	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	*OPERAND
0	MNEA	ALEM	YNEA	XMA	DYN	IYC	AMAAC	DMAN	TKA	COMX	TDO	COMC	RSTR	SETR	KNEZ	RETN	
1	LDP																C
2	TAY	TMA	TMY	TYA	TAM-DYN	TAM-IYC	TAM-ZA	TAM	LDX								F
3	SBIT				RBIT				TBIT1				SAMAN	CPAIZ	IMAC	MNEZ	B
4	TCY																C
5	YNEC																C
6	TCMIY																C
7	IAC	A9AAC	A5AAC	A13AAC	A3AAC	A11AAC	A7AAC	DAN	A2AAC	A10AAC	A6AAC	A14AAC	A4AAC	A12AAC	A8AAC	CLA	
8–B	BR																W
C–F	CALL																W

*C = constant; B = bit address; W = memory address; F = file address

FIGURE 9-3.2 TMS 1100/1300 FIXED INSTRUCTION MAP

9-3.2 **TIMING.** The programmable instruction timing is fixed according to the order given in Table 9-3.1. The sequence for an instruction is as follows: the ALU inputs first, storage into memory next, and storage into registers and status latch last. This timing is also given in Section A5 in the Appendix. Both figures in A5 should be noted whenever a fixed microinstruction is combined with programmable macroinstructions. In particular, the R-output register addressing takes place at the same time as the RAM addressing. The ALU inputs are determined during $\phi 1$ while the RAM data is read out.

TABLE 9-3.1 TMS 1000 SERIES PROGRAMMABLE MICROINSTRUCTIONS

Execution Sequence	Mnemonic	Logic Affected	Function
1	CKP	P-MUX	CKI to P-adder input
	YTP	P-MUX	Y-register to P-adder input
	MTP	P-MUX	Memory (X,Y) to P-adder input
1	ATN	N-MUX	Accumulator to N-adder input
	NATN	N-MUX	$\overline{\text{Accumulator}}$ to N-adder input
	MTN	N-MUX	Memory (X,Y) to N-adder input
	15TN	N-MUX	F_{16} to N-adder input
	CKN	N-MUX	CKI to N-adder input
1	CIN	Adder	One is added to sum of P plus N inputs (P+N+1)
	NE	Adder/Status	Adder compares P and N inputs. If they are identical, status is set to zero
	C8	Adder/Status	Carry is sent to status (MSB only)
2	STO	Write MUX	Accumulator data to memory
	CKM	Write MUX	CKI to memory
3	AUTA	AU Select	Adder result stored into accumulator
	AUTY	AU Select	Adder result stored into Y-register
	STSL	Status Latch	Status is stored into status latch

9-3.3 **ALU OPERATION.** Since all of the programmable microinstructions affect the ALU, its operation deserves a thorough understanding. The guidelines are straightforward:

a) Multiple-simultaneous inputs to either one of the multiplexers are logically ORed. See the TBIT1 explanation in paragraph 4-7.3.

b) If an adder input has no microinstruction selecting data, that input to the adder is 0000 in binary (i.e., N input in the MNEZ instruction).

c) Any simultaneous output selections occur in tandem (i.e., C8 and AUTA are executed together).

d) No instruction may use two ALU operations in a single instruction cycle.

9-3.4 THE CONSTANT AND K-INPUT LOGIC. Table 9-3.2 summarizes the data available on the CKI bus for the instruction values listed. Whenever CKP, CKN, or CKM microinstructions are used, the data sent out by CKI is variable depending on the opcode chosen for the instruction. A decoder within the CKI logic block determines the actions, and this decoder cannot be modified (similar to fixed microinstructions). For information concerning how the TMS 1000/1200 instructions utilized the CKI logic, see Table 2-7.1.

TABLE 9-3.2

OPCODE (HEX)	CKI LOGIC OPERATION
00–07	CONSTANT OPERAND → CKI BUS
08–0F	K INPUT → CKI BUS
10–1F	N/A
20–2F	0 → CKI BUS
30–3F	BIT MASK → CKI BUS
40–7F	CONSTANT OPERAND → CKI BUS
80–FF	N/A

9-3.5 INSTRUCTION PROGRAMMABLE LOGIC ARRAY. The maximum number of PLA terms is 30. Any additional programming in the TMS 1000/1200 instruction set requires reduction of terms (explained in paragraphs 9-4.3 and 9-2). Three terms in the TMS 1100/1300 instruction PLA are available for microprogramming. Any further additions to the microprogrammable code requires a reduction of terms or the loss of a present standard instruction. A direct replacement of one instruction for another requires no additional PLA terms. The accompanying diagram, Figure 9-3.3, is convenient for checking trial coding.

9-3.6 SIMULATION. The HE-1 and HE-2 emulators verify microprogramming results through hardware. The SE-1 and SE-2 devices have the capability to use the standard-instruction sets. However, it is easy to verify any instruction set through software simply by using the TMS 1000 series simulator and assembler. Note that the simulator accepts the instruction PLA coding and the assembler accepts only the mnemonic definitions (see Figure 9-1).

9-3.7 TEST GENERATION. An automatic-test-generation program provides Texas Instruments a test pattern sequence for all codes using the standard-instruction sets. An additional NRE cost can be incurred for major deviations from the standard-instruction set. The cost is dependent on the amount of engineering required to devise a complete test for the unit. For some changes, however, there is no charge. Table 9-3.3 and 9-3.4 list the instructions needed by the test-generator algorithms. If those instructions listed are changed, Texas Instruments must hand program the test. All instructions on the list should be used at least once in the program or be placed in unused portions of the instruction memory. For instance, an unused instruction can be placed at the end of a page that contains only 63 instructions. If there are deviations from the standard instruction set, or if all the instructions are not somewhere in the ROM, please inform the MOS Division in Houston, Texas.

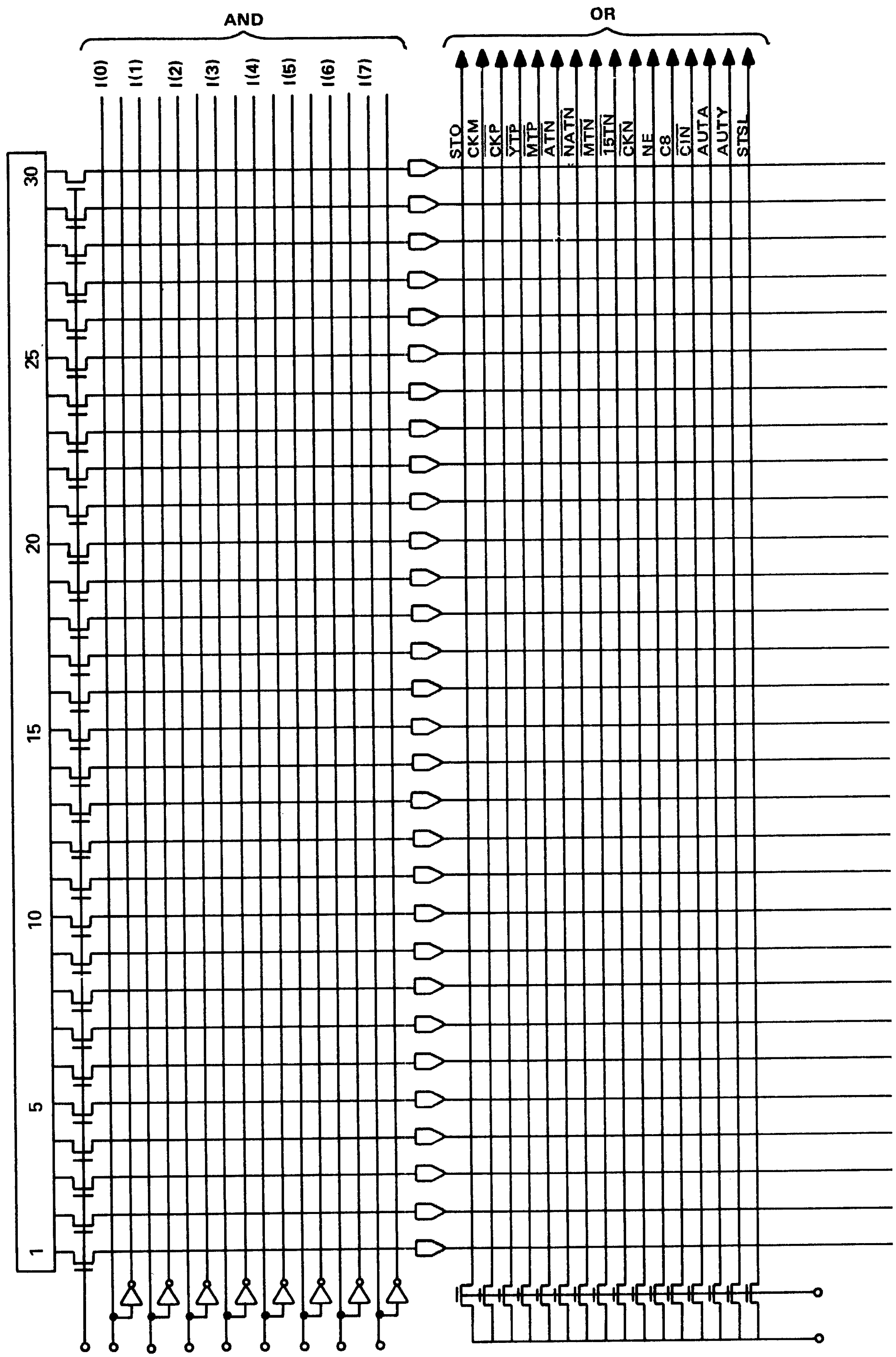


FIGURE 9-3.3 TMS 1000 INSTRUCTION PLA

TABLE 9-3.3 TMS 1000/1200 TEST ALGORITHM INSTRUCTIONS

MNEMONIC	OPCODE
ALEC	0 1 1 1 — — — —
AMAAC	0 0 1 0 0 1 0 1
BR	1 0 — — — — — —
CLA	0 0 1 0 1 1 1 1
COMX	0 0 0 0 0 0 0 0
DYN	0 0 1 0 1 1 0 0
IYC	0 0 1 0 1 0 1 1
LDP	0 0 0 1 — — — —
LDX	0 0 1 1 1 1 — —
RBIT	0 0 1 1 0 1 — —
SBIT	0 0 1 1 0 0 — —
SETR	0 0 0 0 1 1 0 1
TAM	0 0 1 0 0 0 0 0
TAMIY	0 0 0 0 0 0 1 1
TAY	0 0 1 0 0 1 0 0
TBIT1	0 0 1 1 1 0 — —
TCY	0 1 0 0 — — — —
TDO*	0 0 0 0 1 0 1 0
TKA	0 0 0 0 1 0 0 0
TMA	0 0 1 0 0 0 0 1
TYA	0 0 1 0 0 0 1 1
XMA	0 0 1 0 1 1 1 0
YNEA	0 0 0 0 0 0 1 0
YNEC	0 1 0 1 — — — —

*TDO can be programmed to decrement the Y register.

TABLE 9-3.4 TMS 1100/1300 TEST ALGORITHM INSTRUCTIONS

MNEMONIC	OPCODE
AMAAC	0 1 1 1 0 0 0 0
BR	1 0 — — — — — —
CLA	0 0 0 0 0 1 1 0
COMC	0 0 0 0 1 0 1 1
COMX	0 0 0 0 1 0 0 1
DYN	0 0 0 0 0 1 0 0
IYC	0 0 0 0 0 1 0 1
LDP	0 0 0 1 — — — —
LDX	0 0 1 0 1 — — —
RBIT	0 0 1 1 0 1 — —
SBIT	0 0 1 1 0 0 — —
SETR	0 0 0 0 1 1 0 1
TAM	0 0 1 0 0 1 1 1
TAMIY	0 0 1 0 0 1 0 1
TAY	0 0 1 0 0 0 0 0
TBIT1	0 0 1 1 1 0 — —
TCY	0 1 0 0 — — — —
TDO*	0 0 0 0 1 0 1 0
TKA	0 0 0 0 1 0 0 0
TMA	0 0 1 0 0 0 0 1
TYA	0 0 1 0 0 0 1 1
XMA	0 0 0 0 0 0 1 1
YNEA	0 0 0 0 0 0 1 0
YNEC	0 1 0 1 — — — —

*TDO can be programmed to decrement the Y register.

9-3.8 SUMMARY. To sum up, knowledge of the TMS 1000 series logic is all that one needs to start inventing instructions. The only hard step to make is deciding when an instruction redefinition is appropriate. The decision must weigh the schedules, the cost, and the simulation methods required. In some cases, concerning marginal feasibility, the instruction set will have to be modified and the effects on a program development are readily assessed. However, in the latter development stages a problem may arise unexpectedly. In such cases, it is suggested that the Applications Programming Staff in Houston be involved in obtaining a satisfactory solution to the problem. If the proper information can be provided, an alternative path (such as ROM code reduction) may solve such problems without resorting to microprogramming.

9-4 MICROPROGRAMMING HINTS.

Two of the following examples show how microinstructions define a more powerful instruction in the TMS 1100/1300 instruction PLA. A third example indicates how don't cares in PLA programming enable multiple instructions and make room in the TMS 1000/1200 for more instructions.

9-4.1 TDO EXAMPLE. Assuming that every time the user wants a transfer-data-out (TDO) instruction, he also needs a decrement-Y-register (DYN) instruction. If these two instructions combine into one instruction (six oscillator pulses long), then execution time is reduced and the ROM instruction count is shorter. The following steps proceed according to the guideline set up in paragraph 9-3:

1) The TDO opcode is $0A_{16}$. Since TDO is a fixed microinstruction, $0A_{16}$ is permanently assigned to that instruction. See paragraph 9-3.1.

2) The appropriate programmable microinstructions are checked for timing. The decrement-Y-register uses YTP, 15TN, C8, and AUTY. The Y-register contents plus 15 (−1) transfer back into the Y-register. The C8 microinstruction sends the carry-bit to the status logic. Since TDO is not dependent on the Y-register or status logic, the timing is appropriate. See paragraphs 9-3.2 and 9-3.3. Figure 9-4.1 shows the data flow involved in the new instruction.

3) The CKI logic is unused by the new instruction which will be defined as TDODYN. Check paragraph 9-3.4.

4) The TMS 1100/1300 instruction PLA has three available product terms. Term 28 receives gates to decode $0A_{16}$ as indicated in Figure 9-4.2. The microinstructions YTP, 15TN, C8, and AUTY are enabled by gates on the OR matrix on the right half of Figure 9-4.2.

5) The input that defines the new mnemonic to the assembler is shown in Figure 9-4.3. The entire instruction set description must be contained in the **INSTRUCTIONS LDP** section.

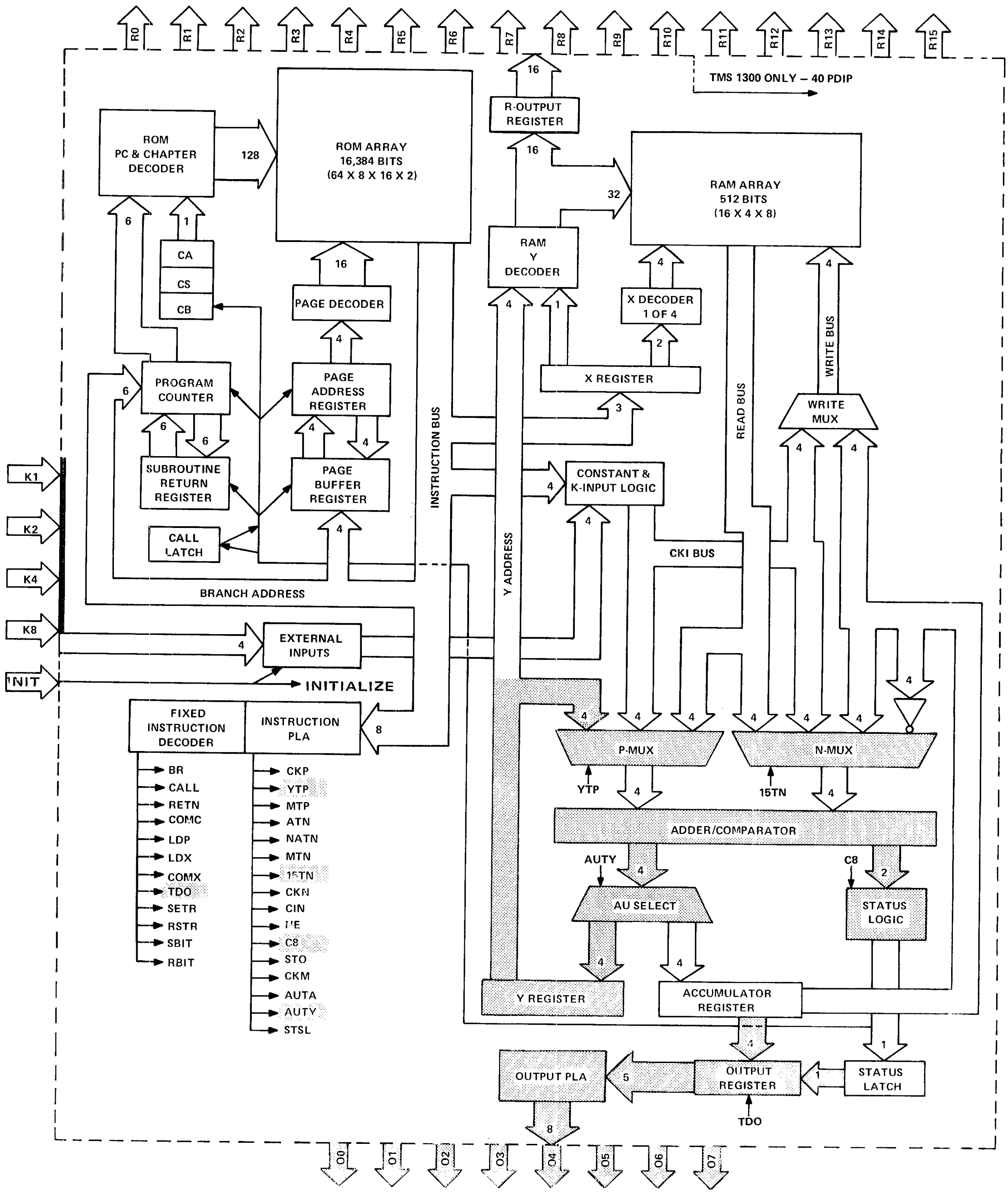


FIGURE 9-4.1 TDODYN INSTRUCTION DATA FLOW – TMS 1100/1300

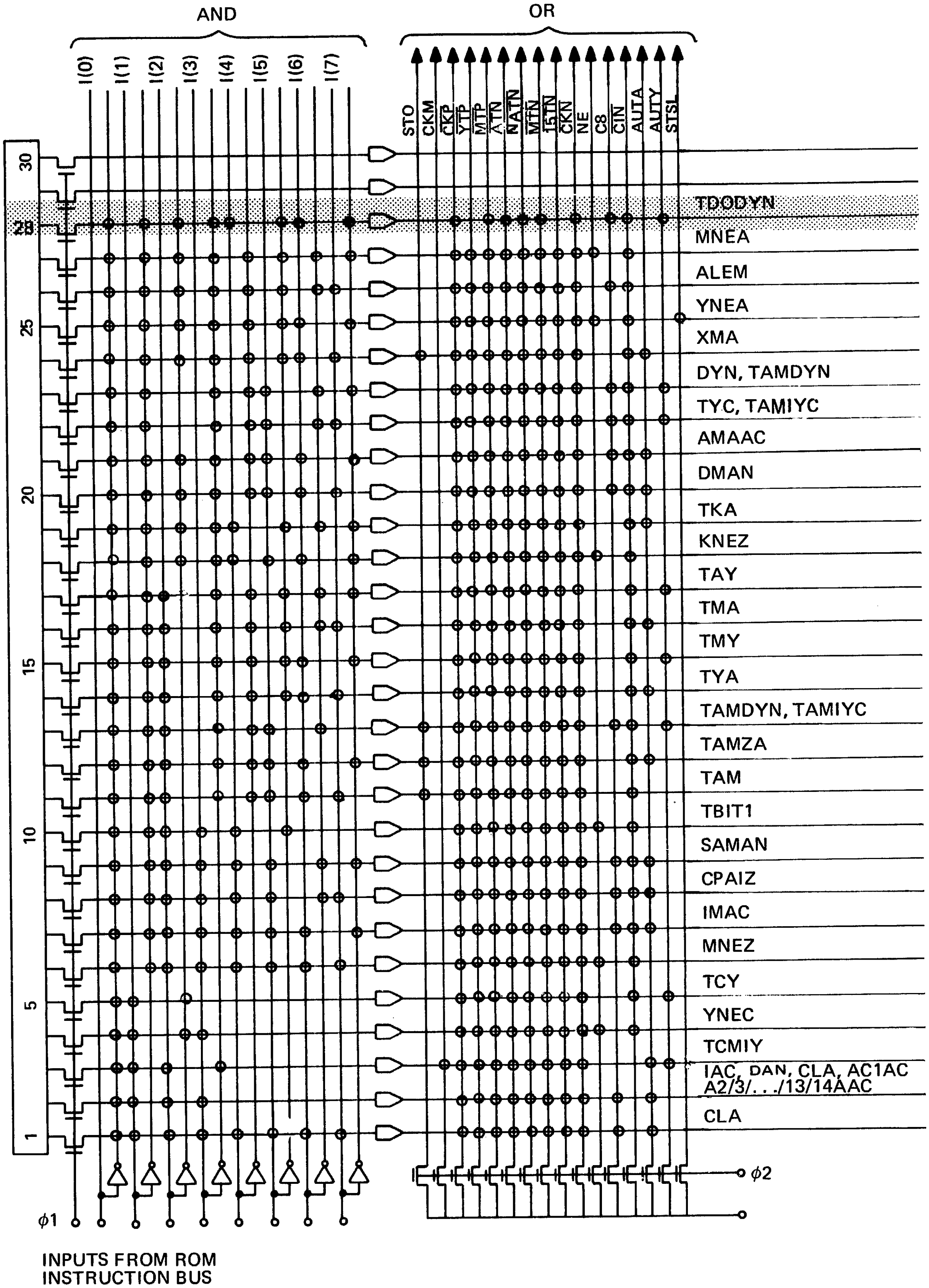


FIGURE 9-4.2 TMS 1100/1300 NON-STANDARD INSTRUCTION PLA

	INSTRUCTIONS	LDP
MNEA	IV0	4
ALEM	IV1	4
YNEA	IV2	4
XMA	IV3	4
DYN	IV4	4
IYC	IV5	4
AMAAC	IV6	4
DMAN	IV7	4
TKA	IV8	4
COMX	IV9	4
TDODYN	IV10	4
COMC	IV11	4
RSTR	IV12	4
SETR	IV13	4
KNEZ	IV14	4
RETN	IV15	4
LDP	II1, IIC	2
TAY	IV32	4
TMA	IV33	4
TMY	IV34	4
TYA	IV35	4
TAMDYN	IV36	4
TAMIYC	IV37	4
TAMZA	IV38	4
TAM	IV39	4
LDX	V5, VC	5
SBIT	III12, IIIC	3
RBIT	III13, IIIC	3
TBIT1	III14, IIIC	3
SAMAN	IV60	4
CPAIZ	IV61	4
IMAC	IV62	4
MNEZ	IV63	4
TCY	II4, IIC	2
YNEC	II5, IIC	2
TCMIY	II6, IIC	2
IAC	IV112	4
A9AAC	IV113	4
A5AAC	IV114	4
A13AAC	IV115	4
A3AAC	IV116	4
A11AAC	IV117	4
A7AAC	IV118	4
DAN	IV119	4
A2AAC	IV120	4
A10AAC	IV121	4
A6AAC	IV122	4
A14AAC	IV123	4
A4AAC	IV124	4
A12AAC	IV125	4
A8AAC	IV126	4
CLA	IV127	4
AC1AC	II7, IIC	V#15 2
BR	I2, BA	1
CALL	I3, BA	1
	END	

FIGURE 9-4.3 TMS 1100/1300 NON-STANDARD INSTRUCTION SET ASSEMBLER DEFINITION

6) The input that defines the new instruction PLA is shown in Figure 9-4.4. The OPCPLA section must be completely described to the simulator.

7) The list in Table 9-3.4 shows that TDODYN is accepted by the automatic test generation program. Thus, no extra NRE cost is incurred.

OPCPLA	
OPX 00=MTP,ATN,NE;	MNEA
OPX 01=MTP,NATN,CIN,C8;	ALEM
OPX 02=YTP,ATN,NE,STSL	YNEA
OPX 03=STO,MTP,AUTA;	XMA
OPB 00–00100=YTP,15TN,AUTY,C8;	DYN,TAMDYN
OPB 00–00101=YTP,CIN,AUTY,C8;	IYC,TAMIYC
OPX 06=MTP,ATN,AUTA,C8;	AMAAC
OPX 07=MTP,15TN,AUTA,C8;	DMAN
OPX 08=CKP,AUTA;	TKA
OPX 0E=CKP,NE;	KNEZ
OPX 20=ATN,AUTY;	TAY
OPX 21=MTP,AUTA;	TMA
OPX 22=MTP,AUTY;	TMY
OPX 23=YTP,AUTA;	TYA
OPB 0010010–=STO,YTP,15TN,CIN,AUTY,C8;	TAMDYN,TAMIYC
OPX 26=STO,AUTA;	TAMZA
OPX 27=STO;	TAM
OPB 001110––=CKP,CKN,MTP,NE;	TBIT1
OPX 3C=MTP,NATN,CIN,AUTA,C8;	SAMAN
OPX 3D=NATN,CIN,AUTA,C8;	CPAIZ
OPX 3E=MTP,CIN,AUTA,C8;	IMAC
OPX 3F=MTP,NE;	MNEZ
OPB 0100––––=CKP,AUTY;	TCY
OPB 0101––––=YTP,CKN,NE;	YNEC
OPB 0110––––=CKM,YTP,CIN,AUTY;	TCMIY
OPB 0111––––=CKP,ATN,CIN,AUTA,C8;	IAC,DAN,A2/3/. . ./13/14AAC,CLA,AC1AC
OPX 7F=CKP,CIN,AUTA,C8;	CLA
OPX 0A=YTP,15TN,AUTY,C8;	TDODYN

FIGURE 9-4.4 TMS 1100/1300 NON-STANDARD INSTRUCTION PLA CODING

9-4.2 BR EXAMPLE. For some algorithms a loop is used for critically timed output control. A solution is to execute the first word in the loop at the same time the branch to loop occurs. The branch address should be inconspicuous because all branch instructions with the same operand (regardless of the page address whereupon the branch occurs) produce the combined results of the fixed and programmable microinstructions. A branch-to-program counter address 20_{16} sends the control to the last location within a page. The next address in the loop is 00 (by the wrap-around shift-register counter effect). Let the decrement-Y-register (DYN) be the first instruction in the loop. Assume that reducing the instruction execution cycles by one is sufficient to speed up the loop's cycle time. Then the following steps are necessary to implement the new instruction that combines DYN with a branch instruction to address 20_{16}.

1) Assign BRDYN the opcode value of $A0_{16}$.

2) Define the microinstructions necessary to implement DYN. (YTP, 15TN, AUTY, C8)

3) Check limitations of the instruction PLA and timing constraints.

4) Define BRDYN to the assembler, and limit the validity of BR to exclude operands of 20_{16}.

5) Define the BRDYN code and the microinstructions to the simulator.

6) Check the list of instructions used by the automatic-test generator. The new instruction may require a software change to the test program.

9-4.3 REDUCING PLA TERMS. The reduction of terms is analegous to minimizing a Boolean expression by means of a Karnaugh map or algabraic methods. The purpose is to enable a logical expression with fewer gates. Due to the hardware structure found in PLAs, the technique of inspection is sufficient to solve most minimization problems. The TMS 1100/1300 standard-instruction set was minimized by inspection. The same is possible in the TMS 1000/1200 instruction set for users who must add microinstructions to fixed instructions (without losing existing instructions). The 30 term instruction PLA is filled by the existing TMS 1000/1200 coding. The following procedure reduces the 30 terms into 29 terms.

1) Looking at Table 3-4, The TMS 1000/1200 microinstruction index, some instructions seem to have similar or overlapping microinstructions. For example, notice CLA, TAM, and TAMZA.

2) After finding that the three instructions use only two positive-logic microinstructions, STO and AUTA, it is possible to use only two PLA terms to completely define the three instructions.

3) If one PLA term decodes CLA and TAMZA, and if another PLA term decodes TAM and TAMZA, TAMZA can be completely defined by the overlapping of the two terms.

   if: Term A enables AUTA for CLA and TAMZA, and Term B enables STO for TAMZA and TAM, then simultaneously Terms A and B enable both AUTA and STO, TAMZA.

4) The instruction map (Figure 3-1) helps visualizing how the next step is accomplished. A don't-care bit is assigned to Term A and Term B such that two opcodes enable each term. A third opcode must enable both terms.

5) Since the standard instructions' opcodes do not have the necessary similiarities, the opcodes are reassigned to fit the requirements:

MNEMONIC	OPCODE HEXADECIMAL	OPCODE BINARY
TAM	03	0 0 0 0 0 0 1 1
TAMZA	23	0 0 1 0 0 0 1 1
CLA	22	0 0 1 0 0 0 1 0
TYA	04	(replaced by TAMZA)
TMY	2F	(replaced by CLA)

The shaded bits are to be the don't cares for the two terms.

6) The simulator inputs for the two terms in question are:

OPB 00–00011=STO; TAM,TAMZA
OPB 0010001–=AUTA; TAMZA,CLA

TYA and TMY also require changes to the product matrix to reflect the reassigned opcodes.

9-5 PLA TERM MINIMIZATION IN THE OUTPUT PLA.

Since the reduction procedure described in 9-4.3 is common to all PLAs, application to the output PLA is possible. In many circumstances the user desires more than 20 output codes from the O PLA. Don't-care bits will enable two PLA terms to generate a third output code if both are enabled simultaneously. An example illustrating the reduction procedure follows:

Assume a seven-segment display provides a user with a hexadecimal output font. Five other codes are also needed. The output PLA has 20 terms, so the requirement for PLA terms must be reduced from 21 to 20. The coding in Figure 2-16.2 is a good starting point for this example. Note the overlapping nature of the output codes for zero, eight, and nine. If the data for zero and nine are logically ORed, the character eight is the output code.

Fortunately, zero and eight differ by only one bit, and eight and nine differ by only one bit:

O-REGISTER:	SL	$A_8A_4A_2A_1$	$O_7O_6O_5O_4O_3O_2O_1O_0$	–	O OUTPUTS
F_0	1	0 0 0 0	0 1 1 1 1 1 1 0		'zero'
	1	1 0 0 0	1 1 1 1 1 1 1 0		'eight'
F_9	1	1 0 0 1	1 1 0 1 1 1 1 0		'nine'

Thus, one term decoding the following O-register data enables zero and eight:

$$F_0 = SL \cdot \overline{A}_4 \cdot \overline{A}_2 \cdot \overline{A}_1$$

A second term decoding the following O-register data enables eight and nine:

$$F_9 = SL \cdot A_8 \cdot \bar{A}_4 \cdot \bar{A}_2$$

Thus, one don't care is selected in the following general Boolean equation for a given term F_i ($i = 0$ to 19) in the output PLA:

$$F_i = SL \cdot \prod_{\ell = 1,2,4,8} A_\ell, \qquad SL = 1,0,X; \; A = 1,0,X.$$

If both terms F_0 and F_9 are true for 18 in the output register, then the following Boolean expression determines the code for each O-output terminal O_j:

$$O_j = \sum_{i=0}^{19} F_i, \qquad \text{where } j = 0 \text{ to } 7$$

In the case where both F_0 and F_9 are true, O_1 through O_7 are all on and the character eight is enabled.

The above procedure can be applied also to the codes for C, E, and F. Each time the procedure is used, a new term is available for another specific O-output code.

SECTION X

SUBROUTINE SOFTWARE

10-1 GENERAL.

By using the subroutine capability of the TMS 1000 series, programs are substantially compacted, enabling the user to write very powerful algorithms within the 1024- or 2048-word limit. Normally "straight line" programming is used only for high-speed applications.

A subroutine is used to avoid duplication of ROM code when a particular section of code is used several times within a program.

A subroutine is a section of code terminated with a RETN instruction. A CALL instruction transfers program execution to the first instruction in the subroutine. At the completion of the subroutine program control is transferred to the instruction address immediately following the CALL. Examples of a subroutine and different calling techniques follow. Unless indicated otherwise, these examples and those in the following sections are compatible with the TMS 1100/1300. Only X addressing is changed when running these TMS 1000/1200 programs on the TMS 1100/1300.

10-2 EXAMPLE SUBROUTINE.

The following subroutine, CREG, will clear out digits 0 to 6 on a given RAM file. The RAM X address is set prior to CALLing this subroutine.

```
LABEL   OP CODE  OPERAND  COMMENT

CREG    TCY      0        INITIALIZE Y TO 0
C1      TCMIY    0        THE WORDS ARE CLEARED BY TRANSFERRING THE
                          CONSTANT 0 TO MEMORY WHILE INCREMENTING Y
        YNEC     7        CONTINUE UNTIL WORD 6 IS SET TO 0 (Y = 7)
        BR       C1
        RETN              WHEN Y = 7, RETURN TO THE CALLING PROGRAM
```

10-3 EXAMPLE CALLING SEQUENCE.

Recall that both CALL and BR instructions are conditional on status. To successfully execute a call, the CALL instruction must either follow a non-status affecting instruction, such as TCY or LDX, or follow a status affecting instruction that will leave status set at ONE. If the call is successful, then the contents of the page address and page buffer registers are exchanged. Therefore, care must be taken to ensure that the page buffer contents point to the subroutine page before attempting a call.

The subroutine labeled CREG is arbitrarily placed on ROM page 7. To demonstrate calling sequences, examples of calling from page 7 and calling off of page 7 are given.

10-3.1 CALLING A SUBROUTINE ON THE SAME PAGE. Remember that a successful branch to a page transfers the page buffer contents to the page address register, leaving them identical. The first example assumes that both registers are identical.

LABEL	OP CODE	OPERAND	COMMENT
	LDX	3	BY PRECEDING THE CALL WITH A NON-STATUS AFFECTING INSTRUCTION, THE CALL WILL BE UNCONDITIONAL
	CALL	CREG	SO THIS INSTRUCTION ALWAYS PASSES CONTROL TO CREG
	BR	XYZ	AFTER THE COMPLETION OF CREG, THIS IS THE NEXT INSTRUCTION EXECUTED.

When a subroutine is used initially, it is embedded directly in a straight line sequence rather than CALLed, to save a CALL instruction. This holds only if the page buffer and page address register contents are identical.

LABEL	OP CODE	OPERAND	COMMENT
	LDX	3	
CREG	TCY	0	
C1	TCMIY	0	
	YNEC	7	CREG SUBROUTINE
	BR	C1	
	RETN		
	BR	XYZ	

The following example shows how to correctly call a subroutine when the page buffer has been modified.

LABEL	OP CODE	OPERAND	COMMENT
	LDP	4	CHANGE PAGE BUFFER CONTENTS TO 4
	ALEC	6	IF A IS LESS THAN 7, BRANCH TO ABC ON PAGE 4.
	BR	ABC	
	LDP	7	IF NO BRANCH, PAGE BUFFER REMAINS = 4. IF A CALL IS ATTEMPTED NOW, CONTROL INCORRECTLY PASSES OVER TO PAGE 4. THUS THE PAGE BUFFER MUST BE SET TO 7.
	CALL	CREG	

10-3.2 CALLING A SUBROUTINE FROM A DIFFERENT PAGE. If the calling sequence is not on the same page as the subroutine, a LDP precedes the CALL so that the page buffer content is equal to the subroutine's page address. The following example demonstrates a CALL from one page to another and a conditional CALL.

LABEL	OP CODE	OPERAND	COMMENT
	LDP	7	SET PAGE BUFFER TO 7
	ALEC	0	IF A = 0, THEN CREG WILL BE SUCCESSFULLY CALLED.
	CALL	CREG	

Notice that the test instruction, ALEC 0, must immediately precede the CALL since status is affected for one instruction cycle only.

10-4 MULTIPLE ENTRY POINTS.

Often it is desired to use a subroutine several times, specifying different conditions each time for entering that subroutine. A call to the multiple entry points presets different conditions, and then a branch into the base subroutine is executed. Thus, rewriting the subroutine for each entry condition is avoided.

The following examples use the CREG routine as the basic subroutine. The CREG routine clears words 0 to 6 on a RAM page where the X address is set before CREG is CALLed (as in example, paragraph 10-3.1). If clearing words 0 to 6 is required more than once in a program, then it is advantageous to create a new subroutine (e.g., CREG3 which sets X to 3 before entering CREG).

```
LABEL   OP CODE  OPERAND   COMMENT

CREG    TCY      0
C1      TCMIY    0
        YNEC     6         BASIC SUBROUTINE
        BR       C1
        RETN
CREG3   LDX      3         SET X = 3
        BR       CREG
```

Now the calling sequence becomes:

```
LABEL   OP CODE  OPERAND   COMMENT

        •
        •
        CALL     CREG3
        •
        •
```

Note that the CREG subroutine is not modified and can be called again.

Another example, again using CREG as the base subroutine, is the subroutine CLALL. CLALL sets Y to 6 before entering the clearing routine, so every word on the RAM page is cleared, including words 7 to 15.

```
LABEL   OP CODE  OPERAND   COMMENT

CREG    TCY      0
C1      TCMIY    0
        YNEC     6         BASIC SUBROUTINE
        BR       C1
        RETN
CLALL   TCY      6         SET Y = 6
        BR       C1        BRANCH INTO THE CLEARING LOOP
```

Since an LDP instruction destroys the subroutine return address, a subroutine and its multiple entry points must be contained within one ROM page. Generally, less LDP instructions are needed when a subroutine resides on the same ROM page that the subroutine is called most frequently.

SECTION XI

ORGANIZING THE RAM

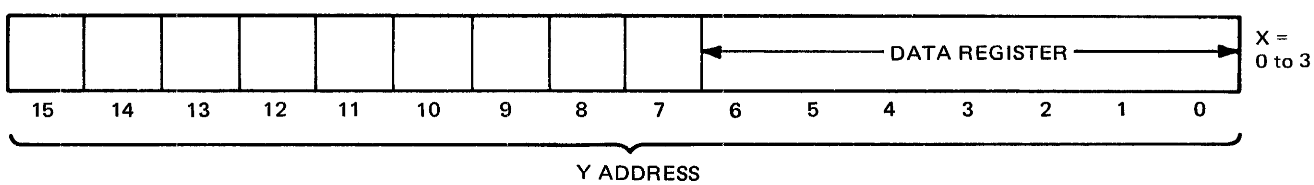
11-1 GENERAL.

To use the TMS1000 data storage efficiently, the locations of storage areas in the RAM must be assigned carefully. The RAM is normally subdivided into data "registers", flag bits, and temporary working areas. If the location assignments are chosen logically, unnecessary use of RAM addressing instructions, such as LDX and TCY, can be minimized.

The following paragraphs given general guidelines for RAM organization which is useful in most programs. The routines and guides given are applicable to the TMS 1100/1300 as well as the TMS 1000/1200.

11-2 DATA REGISTER ORGANIZATION.

To minimize X and Y addressing, data "registers" should be located on sequential locations on the same RAM page. For example, the following is a seven word "register" organization that defines a subset of any 16 word file:



If organized in this manner, a register left-shift subroutine requires only three Y addressing instructions.

NOTE

Assume in this example and in all succeeding examples, that register location Y = 0 is the least significant digit (LSD) and location Y = 6 is the most significant digit (MSD).

11-2.1 REGISTER LEFT SHIFT EXAMPLE.

LABEL	OP CODE	OPERAND	COMMENT
LSHFT	CLA		ENTER THIS SUBROUTINE WITH X SET. THE ACCUMULATOR WILL BE TRANSFERRED TO THE LSD, SO INITIALIZE TO 0.
LDATA	TCY	0	THE LOCATION Y = 0 IS THE LSD.
L1	XMA		EXCHANGE MEMORY AND ACCUMULATOR.
	IYC		INCREMENT Y
	YNEC	7	KEEP EXCHANGING UNTIL Y = 7.
	BR	L1	
	RETN		

Note that this subroutine could also be a data entry subroutine by setting the accumulator equal to the new data and CALLing LDATA.

For most programs, several data registers are required. Whenever possible, these registers are placed on different RAM pages, and their Y locations on these pages would be equal. Thus, X and Y addressing can be minimized on register-to-register operations such as transfers, addition and subtraction. If organized as in Figure 11-2.1, a transfer from register DR0 to register DR1 requires only three addressing instructions.

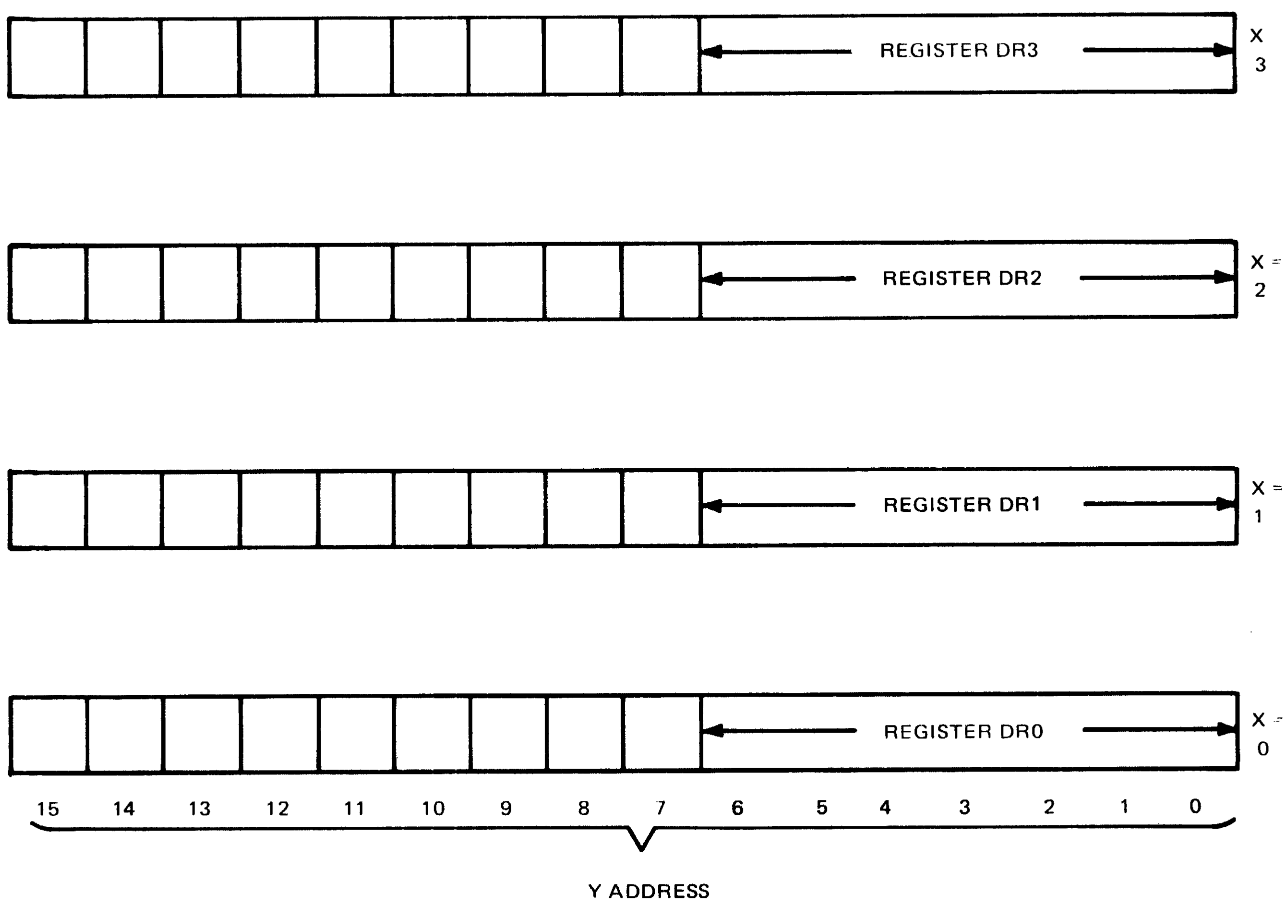


FIGURE 11-2.1 EXAMPLE OF RAM ORGANIZATION

11-2.2 TRANSFER FROM REGISTER 0 TO 1 (EXAMPLE).

LABEL	OP CODE	OPERAND	COMMENT
TR01	TCY	0	INITIALIZE Y = 0
T1	LDX	0	SET X = 0
	TMA		TRANSFER M(0,Y) TO ACCUMULATOR
	LDX	1	SET X = 1
	TAMIY		TRANSFER ACCUMULATOR TO M(1,Y) AND INCREMENT Y
	YNEC	7	CONTINUE UNTIL Y = 7
	BR	T1	
	RETN		

11-2.3 REGISTER TRANSFER EXAMPLE USING COMX. Notice in paragraph 11-2.2 that DR0 contents can be transferred to DR1, but DR1 cannot be transferred to DR0. Also, this subroutine cannot be used for transfers between any other register pairs.

This limitation can be overcome by using the COMX instruction in the following subroutine and by defining paired registers (two registers that transfer to and from each other) to be on complemented X addresses.

LABEL	OP CODE	OPERAND	COMMENT
TR03	LDX	3	THE BASE SUBROUTINE CAN TRANSFER REGISTER DR0 TO DR3, DR3 TO DR0, DR1 TO DR2, AND DR2 TO DR1. FOR DR0 TO DR3 PRESET X TO 3.
TR0	TCY	0	INITIALIZE Y = 0
T2	COMX		COMPLEMENT X
	TMA		TRANSFER M(X,Y) TO ACCUMULATOR
	COMX		COMPLEMENT X
	TAMIY		TRANSFER ACCUMULATOR TO M(X,Y), AND INCREMENT Y
	YNEC	7	TRANSFER UNTIL Y = 7
	BR	T2	
	RETN		
TR30	LDX	0	FOR THE TRANSFER OF DR3 TO DR0 PRESET X TO 0
	BR	TR0	
TR12	LDX	2	FOR THE TRANSFER OF DR1 TO DR2, PRESET X TO 2
	BR	TR0	
TR21	LDX	1	
	BR	TR0	

By using multiple entry points, four register transfers are accomplished with one base subroutine.

11-3 PLACING FLAG BITS.

One should carefully choose the location of flag bits. As usual, the objective is to minimize addressing instructions. The following are general suggestions on bit placement.

Registers and the registers' sign bits should be located in the same RAM page and located in adjacent Y addresses. This permits the transfer of sign bits by the same subroutine which transfers the register contents. For instance, in examples in paragraphs 11-2.2 and 11-2.3, sign bits located in

Y = 7 could be transferred, along with the data in the register, simply by changing the YNEC 7 to a YNEC 8 command.

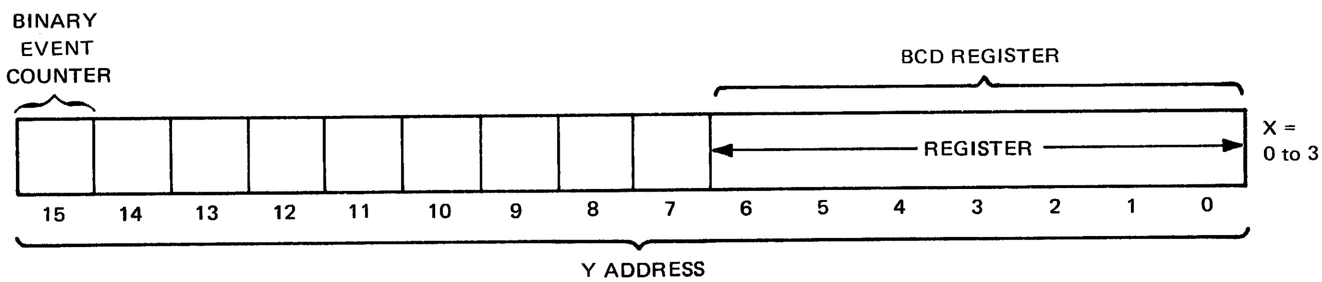
Different flags which are tested sequentially in a program should be placed in the same RAM word to eliminate both X and Y changes between tests.

11-4 TEMPORARY WORKING AREAS.

Temporary working areas are either full register length, which is required in a three register calculation (i.e., divide, when one register holds the intermediate results), or shorter length areas which are used for counters and pointers.

Data Register 1, in Figure 11-2.1, is a good location for storing the results from DR0 divided by DR3, as is demonstrated in a later example. To transfer the result back to DR0, a simple DR1 to DR0 transfer is required.

Pointers and counters should be placed on the same RAM file as the registers that they interact with (Register X number = file 0, 1, 2, or 3). The following example monitors an external event (detected at EXIT label), counts to 16 (e.g., 16 items loaded into a container), and then adds one to a BCD register in the same file for each count of 16.



```
LABEL    OP CODE  OPERAND   COMMENT

COUNT    TCY      15        SET Y TO BINARY EVENT (B.E.) COUNTER LOCATION (15)
         IMAC               BE COUNTER +1
         BR       ADD 1     IF CARRIED, BRANCH TO ADD1 WHICH ADDS 1
                            TO THE BCD REGISTER.
         TAM                IF NO CARRY TRANSFER ACCUMULATOR TO BE COUNTER
         BR       EXIT      AND BRANCH TO EXIT WHERE A ROUTINE IS LOCATED
                            WHICH WAITS FOR THE EVENT WHICH IS BEING COUNTED.
ADD1     TAMIY              IF 16'TH PULSE, ACCUMULATOR = 0, SO TRANSFER
                            ACCUMULATOR TO B.E. COUNTER AND INCREMENT
                            Y (BECOMES = 0)
INCM     IMAC               M(X,Y) + 1 TO ACCUMULATOR
         TAM                STORE ACCUMULATOR BACK INTO M(X,Y).
         ALEC     9         IF ACCUMULATOR WAS LESS THAN OR EQUAL TO 9, THEN
                            GO TO EXIT, SINCE THE REGISTER IS CORRECT IN BCD.
         BR       EXIT
         TCMIY    0         IF ACCUMULATOR > 9, SET THIS DIGIT = 0, AND ADD
                            ONE TO THE NEXT HIGHER ORDER DIGIT.
         BR       INCM
         •
         •
         •
EXIT     •                  EXTERNAL ROUTINE WAITS FOR THE
         •                  COMPLETION OF A
         •                  NEW EVENT THAT
         BR       COUNT     IS TO BE COUNTED
```

Notice that by placing the BCD counter and B.E. counter in the same file, no X addressing was required. The LSD of the BCD register was addressed with TAMIY, saving both ROM code and execution time.

SECTION XII

GENERAL PURPOSE SUBROUTINES

12-1 REGISTER RIGHT SHIFT.†

This routine right shifts the register and enters a 0 to the MSD.

NOTE

These subroutines assume that the data registers are organized as in Figure 11-2.1.

```
LABEL    OP CODE  OPERAND  COMMENT

RSHIFT   CLA
         TCY      6        SET Y = 6 (MSD)
R1       XMA               EXCHANGE MEMORY CONTENTS AND ACCUMULATOR.
         DYN               DECREMENT Y TO NEXT LOWER DIGIT
         BR       R1       CONTINUE UNTIL Y EQUALS ZERO.
         RETN
```

12-2 REGISTER EXCHANGE.†

This subroutine exchanges registers DR0 and DR3 or DR2 and DR1 depending on entry point. If entered at label EX03, DR0 is exchanged with DR3.

```
LABEL    OP CODE  OPERAND  COMMENT

EX03     LDX      3
EX0      TCY      0        PRESET Y = 0
EX1      TMA               TRANSFER M(X,Y) TO ACCUMULATOR.
         COMX              COMPLEMENT X
         XMA               EXCHANGE M(X,Y) AND ACCUMULATOR.
         COMX              COMPLEMENT X AGAIN
         TAMIY             TRANSFER ACCUMULATOR TO MEMORY AND INCREMENT Y
         YNEC     7        CONTINUE EXCHANGING UNTIL X = 7
         BR       EX1
         RETN
         •
         •
         •
         •
EX21     LDX      2        IF ENTERED HERE, REGISTER DR1 IS EXCHANGED WITH DR2.
         BR       EX0
```

†The routines in 12-1 and 12-2 are compatible with the TMS 1100/1300 except X address assignments.

12-3 DECIMAL ADDITION.

The following subroutine adds two registers word by word in BCD. In BCD, if the sum of two words is greater than nine, then the correction-factor six must be added to the result, and a carry must be propagated to the next higher order digit. In this subroutine, the carry bit is propagated by the accumulator. Register DR0 is added to register DR3 and the result stored in register DR0. As before, X may be preset in the beginning to have: DR0 + DR3 → DR3, DR1 + DR2 → DR1, or DR1 + DR2 → DR2.

LABEL	OP CODE	OPERAND	COMMENT
A030	TCY	0	PRESET Y = 0
	CLA		CLEAR ACCUMULATOR WHICH WILL BE USED AS THE CARRY DIGIT
AD1	COMX		COMPLEMENT X
	AMAAC		ADD M(X,Y) TO ACCUMULATOR (POSSIBLE CARRY)
	COMX		COMPLEMENT X
	AMAAC		ADD M(X,Y) + [M(X,Y) + CARRY], ANSWER TO ACCUMULATOR
	BR	GT9	BRANCH IF THE SUM WAS GREATER THAN 15
	ALEC	9	NOW TEST FOR A SUM LESS THAN 10
	BR	LT10	AND BRANCH TO LT10
GT9	A6AAC		IF SUM GREATER THAN 15, ADD 06.
	TAMZA		TRANSFER THE CORRECTED SUM TO MEMORY AND CLEAR THE ACCUMULATOR.
	IA		SET THE ACCUMULATOR (CONTAINS THE CARRY BIT) = 1
INCY	IYC		INCREMENT Y
	YNEC	7	CONTINUE ADDING UNTIL Y = 7
	BR	AD1	
	RETN		
LT10	TAMZA		FOR SUMS LESS THAN 10, TRANSFER THE ACCUMULATOR TO MEMORY AND CLEAR THE CARRY DIGIT.
	BR	INCY	

The following subroutine adds two multi-precision registers, DR0 and DR4, for TMS 1100/1300 applications of BCD addition. Since ALEC is not available in the TMS 1100/1300, six-plus-the-accumulator is used to generate a carry if the accumulator is greater than nine. If the accumulator was greater than nine, it is corrected to BCD. If the accumulator was less than or equal to nine, no carry results and ten (−6) is added to the accumulator which is stored in memory.

```
LABEL   OP CODE  OPERAND  COMMENT

A040    TCY      0        PRESET Y
        CLA

AD1     COMX              0→4
        AMAAC
        COMX              4→0
        AMAAC
        BR       GT15     RESULT GREATER THAN 15
        A6AAC             CORRECT TO BCD AND TEST
        BR       GT9      IF GREATER THAN 9, BRANCH
        A10AAC            CORRECT BACK TO BCD
        TAMZA
        BR       INCY

GT15    A6AAC             CORRECTION TO BCD

GT9     TAMZA
        IAC               SET CARRY = 1

INCY    IYC
        YNEC     7
        BR       AD1
        RETN
```

12-4 DECIMAL SUBTRACTION.

This subroutine subtracts two registers word by word in BCD. Subtraction is similar to addition in that if a borrow occurs, the correction factor ten must be added to the result, and the borrow bit that is propagated by the accumulator must be added to the next-higher-order subtrahend digit. In this example, register DR3 is subtracted from register DR0 with the result stored into register DR0. If the initial X address is modified as before, the subroutines DR3 – DR0 → DR3, DR1 – DR2 → DR1, or DR2 – DR1 → DR2 can be generated.

```
LABEL   OP CODE  OPERAND   COMMENT

S300    LDX      0         PRESET X TO 0
        TCY      0
        CLA                CLEAR THE ACCUMULATOR, TO BE USED AS THE
                           BORROW DIGIT.
S1      COMX               COMPLEMENT X; ADDRESS THE SUBTRAHEND.
        AMAAC              ADD SUBTRAHEND M(X,Y) + BORROW, → ACCUMULATOR
        COMX               COMPLEMENT X; ADDRESS THE MINUEND.
        SAMAN              MINUEND M(X,Y) - [SUBTRAHEND M(X,Y) + BORROW]
                           SENT TO ACCUMULATOR.
        BR       NOBOR
        A10AAC             IF BORROW OCCURS, ADD CORRECTION OF + 10
        TAMZA              TRANSFER THE RESULT TO MEMORY AND CLEAR
                           THE ACCUMULATOR.
        IA                 SET ACCUMULATOR (BORROW DIGIT) TO 1.
INCYS   IYC                INCREMENT Y
        YNEC     7         UNTIL Y = 7.
        BR       S1
        RETN
NOBOR   TAMZA              IN THE NO BORROW CASE, TRANSFER THE RESULT TO
                           MEMORY AND CLEAR THE ACCUMULATOR.
        BR       INCYS     IN THE NO BORROW CASE, THE ACCUMULATOR
                           (BORROW) REMAINS = 0.
```

†NOTE: When using the TMS 1100/1300 standard instruction set, IA must be replaced by either IAC or AC1AC 0. The assembler default mnemonic definition accepts either format.

SECTION XIII

EXAMPLE ROUTINES

13-1 GENERAL.

This section provides routines that are commonly found in TMS 1000/1200 and 1100/1300 applications.

13-2 DISPLAY AND KEYBOARD SCAN.

Many applications require the TMS1000 to scan a display and accept keyboard data from momentary and non-momentary switches. Figure 13-2.1 illustrates a typical keyboard and display configuration.

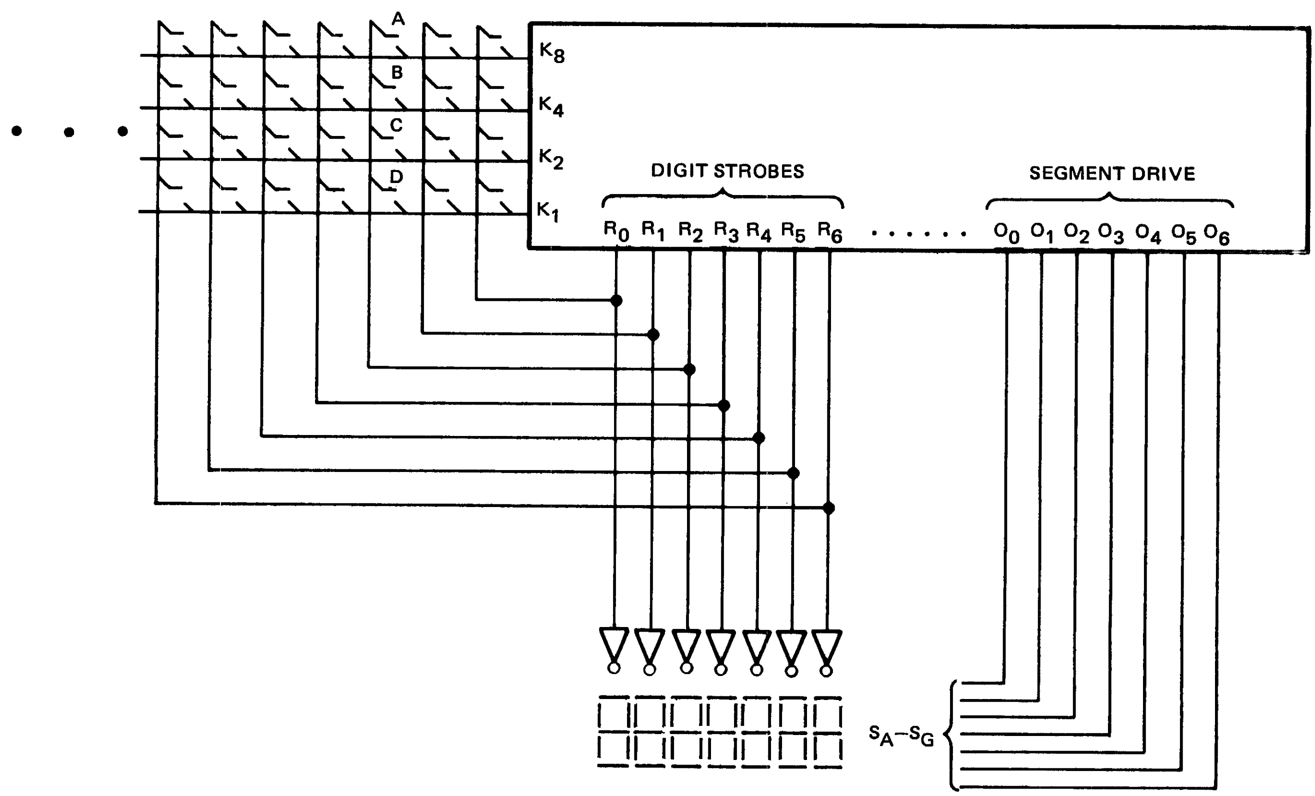


FIGURE 13-2.1. TYPICAL KEYBOARD AND DISPLAY CONFIGURATION

Data is displayed by sequentially setting R lines to display one character at a time. Segment information for that character is decoded through the output PLA which is programmed to provide seven-segment inverted or non-inverted, current drive. For seven-segment drive, an extra O output is available for driving a decimal point or other external logic in the user's system. The state of each switch can be checked while sequentially setting the R lines. For instance, during the time that R2 is on, if switch B is closed and switches A, C, and D are open, then the K inputs would be equal to 4 (binary 0100). Key decoding, debounce, multiple key push protection, and rollover are provided by software control. Also, external logic states are sensed by this scheme of using the R lines as data selectors (see Figure 13-2.2).

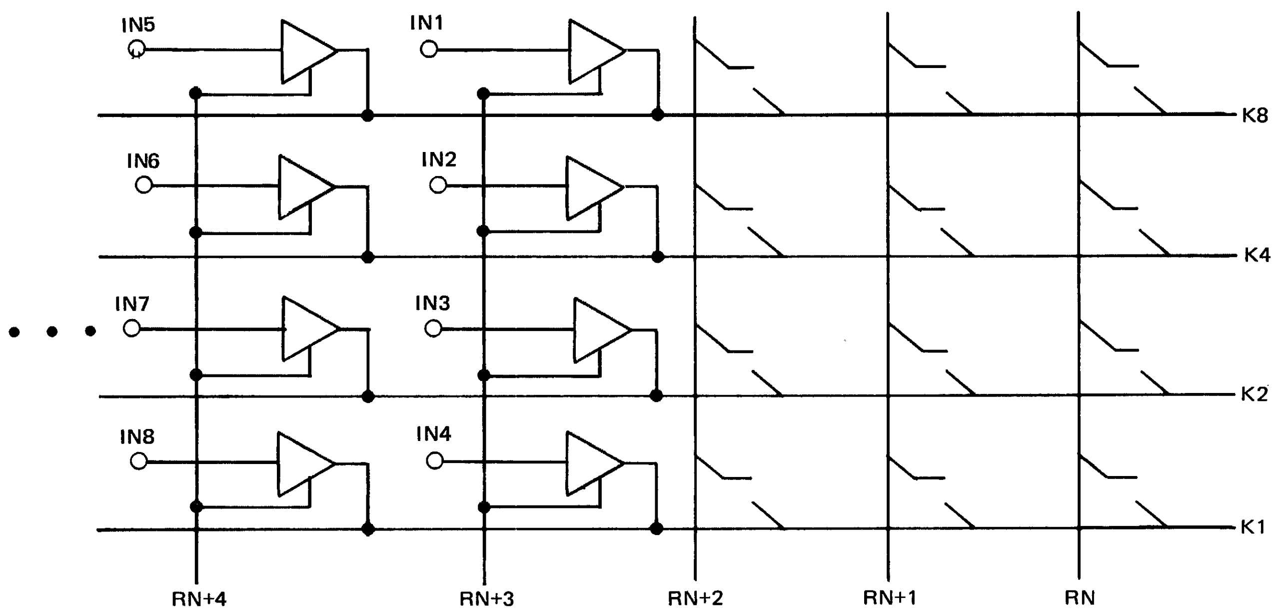


FIGURE 13-2.2. EXTERNAL DATA INPUT MULTIPLEXER

The requirements of the display scan depend upon the type of display used. Variables such as scan speed, direction of digit scan (right to left vs. left to right), duty cycle, digit or segment blanking and leading zero suppression are also controlled by software.

13-2.1 BASIC SCAN ROUTINE. This routine displays words 0 to 6 of a file while checking for a K input. As soon as a K input occurs, an exit is made.

LABEL	OP CODE	OPERAND	COMMENT
DISP	TCY	7	PRESET Y = 7 (SCAN FROM LEFT TO RIGHT)
DIS1	DYN		DECREMENT Y TO TRANSFER LOWER ORDER DIGIT TO THE ACCUMULATOR
	TMA		
	IYC		INCREMENT Y TO RESET HIGHER ORDER R LINE
	RSTR		
	DYN		DECREMENT Y TO SET LOWER ORDER R LINE.
	TDO		TRANSFER ACCUMULATOR TO OUTPUT DECODE
	SETR		
	KNEZ		TEST FOR K INPUT
	BR	EXIT	IF K INPUT IS NOT ZERO, BRANCH TO EXIT
	YNEC	15	TEST FOR COMPLETION OF SCAN, Y = 15
	BR	DIS1	IF NOT COMPLETE, BRANCH TO DIS1
	BR	DISP	IF COMPLETE, REINITIALIZE Y

The timing diagram in Figure 13-2.3 shows how the routine controls the output lines.

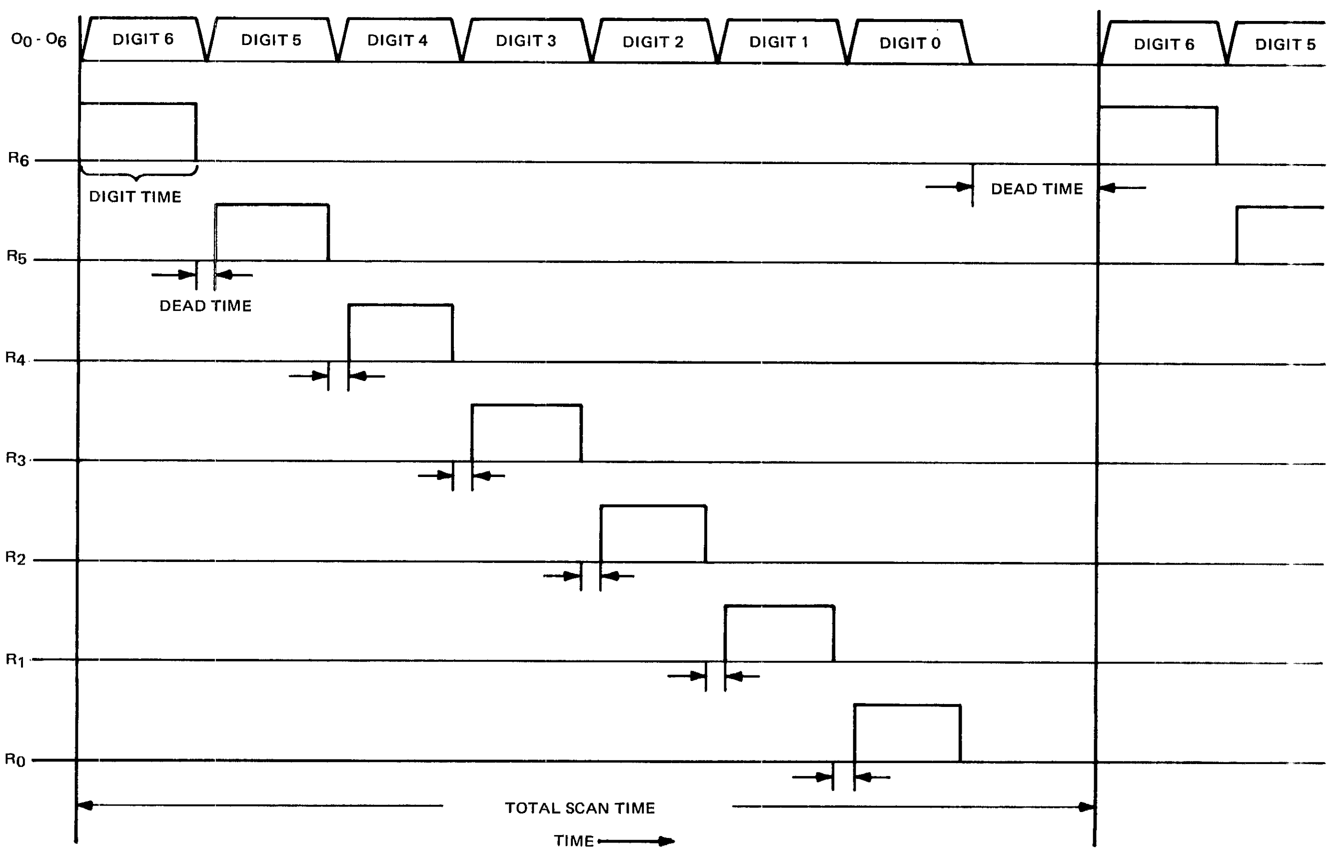


FIGURE 13-2.3. TYPICAL DISPLAY OUTPUT TIMING

Shown in Figure 13-2.3 are "dead times", or times when no data is displayed. The short dead time between sequential digits is caused by the decrement Y that must follow a RSTR. The longer dead time between digit zero and the following digit six is caused by a SETR(Y = 15) which acts like a no-op after digit zero is reset. Duty cycle defined by:

$$\text{DUTY CYCLE} = \frac{\text{INDIVIDUAL DIGIT TIME}}{\text{TOTAL SCAN TIME}}$$

is lowered by this dead time. For applications requiring high duty cycles, more complex display routines are written to keep the dead times to a minimum. To minimize the effects of dead time, individual digit times are also increased, as long as the total scan time is not excessive.

13-2.2 LEADING ZERO SUPPRESSION. To provide leading zero suppression, one character that is not decoded in the output PLA (therefore a blanked O output = all zeros) is loaded into the O-output register to provide a suppressed zero output. This code can be stored in the RAM file digits to be blanked before entering the display routine.

In the following routine 15 will be used as the blank code (not decoded according to output PLA programming). This routine stores 15's in the display register in all digit positions in the display register that are leading zeroes.

LABEL	OP CODE	OPERAND	COMMENT
BLANK	TCY	6	ADDRESS MSD
BL1	MNEZ		IF NON-ZERO, BLANKING COMPLETE SO
	BR	DISP	EXIT TO DISP
	TCMIY	15	IF NOT, SET THAT DIGIT = 15
	DYN		
	DYN		ADDRESS THE NEXT LOWER ORDER DIGIT
	BR	BL1	CONTINUE THIS PROCESS UNTIL Y = 0
	BR	DISP	

13-2.3 KEY DEBOUNCE. K inputs are debounced in software on the TMS1000 by delaying a specified time period after the first K input is detected and then retesting. If the signal is still valid at the second test, then the input is a true input and not noise. Figure 13-2.4 illustrates some of the typical noise problems.

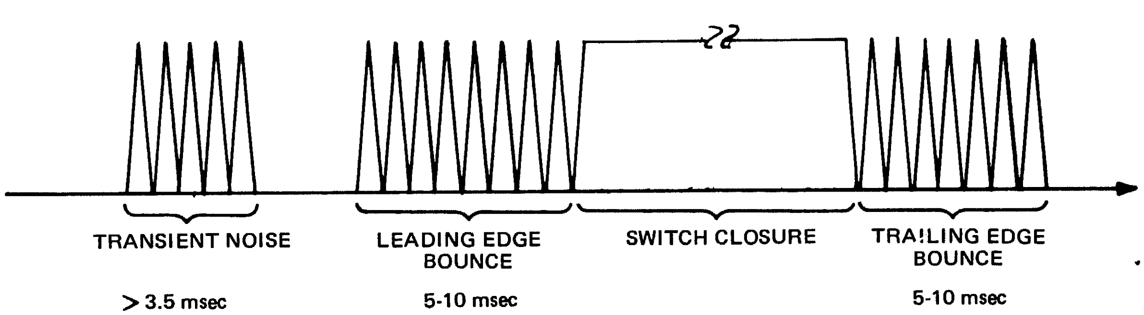


FIGURE 13-2.4. KEY NOISE TIMING

To provide the delay period required for deglitching an input, the display routine can be looped through the required number of times, or the following simple delay loop can be used.

LABEL	OP CODE	OPERAND	COMMENT
DELAY	DAN		THIS ROUTINE COUNTS THE ACCUMULATOR DOWN TO
	BR	DELAY	ZERO WITHIN A LOOP THAT COUNTS Y DOWN TO 0.
	DYN		IF THE ACCUMULATOR AND Y ARE INITIALIZED TO
	BR	DELAY	15, THEN THIS ROUTINE WILL DELAY 544
			INSTRUCTION CYCLES (=9.8 msec @ clock freq = 333 Khz)

13-3 ADDRESSING AN EXTERNAL RAM.

For some applications, the internal data storage capability of the TMS1000 must be supplemented with an external RAM. Most standard RAMS can be utilized by taking advantage of the TMS1000's unique output architecture.

The following routines demonstrate techniques used to address a RAM. A RAM organization of 256 x 4 is used in this example. To simplify the interface requirements RAM has separate input and output data lines as shown in Figure 13-3.1. To further simplify the example, no other external components, such as a keyboard or display, are used.

The RAM address is supplied by R0 to R7. READ/WRITE and CHIP SELECT are R8 and R9. O_0 to O_3 provides data that is read into the RAM. Data is read from the RAM output on K1 to K8.

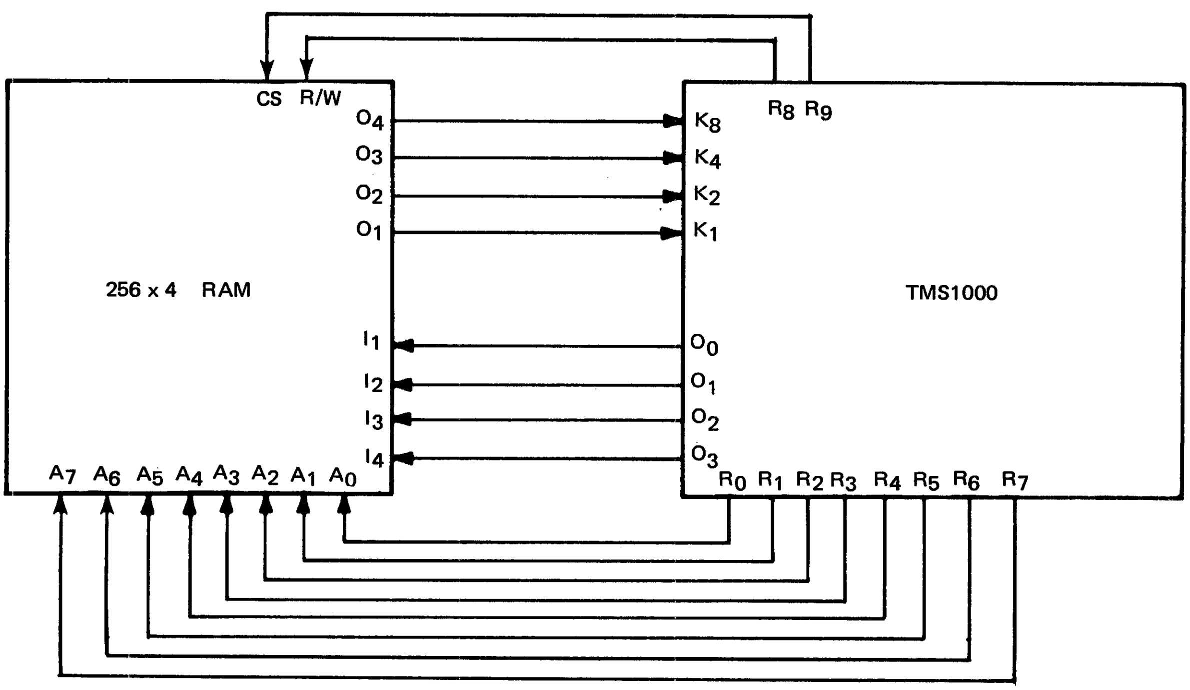


FIGURE 13-3.1. CONNECTIONS TO EXTERNAL RAM

In many applications the RAM address is stored in the TMS1000 in BCD. The first part of this routine converts a three-digit BCD number (000-255) into a two word binary address (00-FF). Next, a routine tests each bit of the two binary words and individually sets or resets corresponding R lines (R0-R7).

13-3.1 CONVERTING BCD TO BINARY. Assume that the number to be converted is stored in RAM words M(X, Y with Y = 1 to 3) with the hundreds digit in M(X,3), the tens digit in M(X,2), and the ones digit in M(X,1). This routine places the converted results in M(X,0) and M(X,1). The word at M(X,1) is the most significant.

First, the ones digit is transferred to M(X,0), and M(X,1) is cleared. Then the tens digit, M(X,2), is decremented until it equals zero. Every time it is decremented, binary ten (1010) is added to M(X,0) with any carry being added to M(X,1). When the tens digit equals zero, the hundreds digit is decremented. For every time that the hundreds digit is decremented, ten is placed into the tens digit (10 x 10) and the process is repeated (decrementing the tens digit). Finally, when the hundreds digit is decremented and it borrows, the conversion is complete.

```
LABEL   OP CODE  OPERAND  COMMENT

BCDBI   CLA               FIRST, TRANSFER ONES DIGIT TO M(X,0)
        TCY      1        AND CLEAR M(X,1)
        XMA               M(X,1) STORES THE CARRY FROM M(X,0).
        TCY      0
        TAM               ONES DIGIT, ALREADY BINARY, GOES TO M(X,0)
BCBI1   TCY      2        ADDRESS THE 10's DIGIT
        DMAN              (10's DIGIT) -1 → ACCUMULATOR
        BR       BCBI3    BRANCH IF NON ZERO TO BCBI3
        TCY      3        IF ZERO, ADDRESS 100's DIGIT
        DMAN              (100's DIGIT) -1 → ACCUMULATOR
        BR       BCBI2    BRANCH IF NOW ZERO TO BCBI2
        BR       BIOUT    IF ZERO, BCD TO BINARY CONVERSION COMPLETE.
BCBI2   TAM               TRANSFER 100's DIGIT -1 (ACCUMULATOR) → 10's DIGIT
        TCY      2        SET 10's DIGIT = 10
        TCMIY    10
        BR       BCBI1    AND BRANCH TO BCBI1 TO CONTINUE DECREMENTING
                          THE 10's DIGIT
BCBI3   TAM               TRANSFER 10's DIGIT -1 ACCUMULATOR → 10's DIGIT
        TCY      0        ADDRESS 1's DIGIT
        TMA               TRANSFER 1's DIGIT → ACCUMULATOR
        A10AAC            ADD 1's DIGIT + 10 → ACCUMULATOR
        BR       BCBI5    IF GREATER THAN 16, BRANCH TO BCBI5
        BR       BCBI6    IF NOT, BRANCH TO BCBI6
BCBI5   TAMIY             TRANSFER 1's DIGIT +10 → 1's DIGIT AND
                          INCREMENT Y
        IMAC              PROPAGATE CARRY TO M(X,1)
BCBI6   TAM               TRANSFER ACCUMULATOR → M
        BR       BCBI1
```

13-3.2 SETTING ADDRESS LINES OF THE EXTERNAL RAM FROM A BINARY NUMBER. Assuming that the binary address is now stored in M(X,1) (most significant) and M(X,0) (least significant), this routine tests each bit at the binary address and sets a corresponding R line if the bit is set. Initially, all R lines are reset. For example if M(X,1) contained 1101 and M(X,0) contained 0101, then this routine would set R7, R6, R4, R2, and R0.

```
LABEL   OP CODE  OPERAND  COMMENT

BIOUT   TCY      2        ADDRESS THE POINTER
        TCMIY    7        SET POINTER TO SEVEN
        TCY      1        ADDRESS M(X,1)
BIOT1   TBIT1    3        TEST BIT 3
        CALL     SET      IF SET, SET R(POINTER)
        CALL     DEC      DECREMENT POINTER
        TBIT1    2        REPEAT FOR BIT 2
        CALL     SET
        CALL     DEC
        TBIT1    1        REPEAT FOR BIT 1
        CALL     SET
        CALL     DEC
        TBIT1    0        REPEAT FOR BIT 0
        CALL     SET
DEC     TYA               THE DEC SUBROUTINE IS EMBEDDED HERE TO
                          SAVE A CALL. HOLD Y IN ACCUMULATOR
        TCY      2        ADDRESS POINTER DIGIT
        XMA               POINTER -1 → POINTER
        DAN
        XMA
        TAY               RETURN CURRENT LOCATION TO Y
        RETN
        DYN               X-1 → Y, IF Y IS NON-ZERO, BRANCH TO BIOT1 TO
        BR       BIOT1    WORK ON THE CONTENTS OF M(X,0)
*                         NOW THAT THE RAM ADDRESS IS SET, DATA CAN
                          EITHER BE READ ON THE K INPUTS OR WRITTEN
                          FROM THE O OUTPUTS.

        •
        •
        •

SET     TYA               SAVE POINTER FOR HEX WORD
        TCY      2        RECALL ADDRESS POINTER FOR R OUTPUT
        TMY
        SETR              SET ADDRESS
        TAY               RESTORE POINTER FOR HEX WORD
        RETN
```

13-4 INTEGER BCD MULTIPLY.

The following routine is an example of a simple integer multiply routine. Using the register organization of Figure 11-2.1, this routine multiplies DR0 and DR3 leaving the result in DR0. The DR1 register holds DR0 data during this operation. The MSD of DR1 is addressed first. Every time this digit can be decremented, DR0 is added to DR3 and results are stored in DR0 (a long call to A030). When that digit has been decremented to zero, DR0 is shifted left and the process repeated for the next-lower-order digit in DR1. When all the digits in BCD have been decremented to zero, the multiplication is complete. Since the multiplication of two N-digit numbers can result in a 2 x N-digit result, this routine does not work for numbers greater than three digits since the product register, DR0, is only a seven-digit register. This restriction is overcome by increasing the length of DR0 or by formatting the data in floating-point mode.

In floating point, each register is left justified prior to multiplication, and the order of multiplication reversed; that is, the LSD of DR1 is decremented first, and a right shift is used rather than a left shift. This results in the six least-significant-digits of the result being discarded.

```
LABEL   OP CODE  OPERAND   COMMENT

MULT    CALLL    TR01      TRANSFER DR0 → DR1 (PARAGRAPH 11-2.3).
        TCMIY    6         UPON EXIT FROM TR01, X = 1 AND Y = 7,
                           A POINTER WILL BE STORED THERE THAT WILL
                           BE USED TO ADDRESS THE DIGIT IN DR1.
        LDX      0         SET POINTER TO 6, CLEAR THE DR0
        CALL     CREG

ML2     LDX      1         ADDRESS THE POINTER
        TCY      7
        TMY                SET Y = POINTER CONTENTS
        DMAN               DR1 (POINTER) -1 → ACCUMULATOR
        BR       NOBR      BRANCH TO NOBR IF DR1 (POINTER) IS NON-ZERO
        TCY      7         IF DR1 (POINTER) = 0, SET POINTER = POINTER -1
        DMAN               POINTER -1 → ACCUMULATOR
        BR       ML1       BRANCH TO ML1 IF POINTER IS NON-ZERO
        BR       EXIT      IF 0, THEN MULTIPLICATION COMPLETE, SO BRANCH TO
                           SOME EXIT ROUTINE.
ML1     TAM                ACCUMULATOR TO POINTER
        LDX      0         SHIFT DR0 LEFT
        CALLL    LSHFT     PARAGRAPH 6-2.1
        BR       ML2
NOBR    TAM                IF DR1 (POINTER) WAS NON-ZERO, TRANSFER
        CALLL    A030      ACCUMULATOR TO DR1 (POINTER) AND ADD DR0
                           TO DR3 → DR0 (PARAGRAPH 7-3)
        BR       ML2
```

13-5 INTEGER BCD DIVIDE.

The following routine is an example of a simple integer divide. In this routine, the DR0 is divided by DR3 with the result left in DR1. The result is transferred to DR0 upon completion. In this routine, DR3 is repeatedly subtracted from DR0. Every time this subtraction is accomplished with no borrow, the DR1 register is incremented. When the first borrow occurs, the division is complete.

This routine is the simplest form of division although it has the longest execution time. If both numbers were in floating point mode, and left justified prior to division, then several division techniques exist that substantially decrease the execution time.

```
LABEL   OP CODE  OPERAND  COMMENT

DIVID   LDX      1        CLEAR C
        CALLL    CREG
D1      CALLL    S300     DR0 - DR3 → DR0
        ALEC     0        IF NO BORROW, AT EXIT FROM S300 THE ACCUMULATOR = 0.
        BR       INCC     IF SO, BRANCH TO INCC
        CALLL    TR10     IF BORROW, DIVISION COMPLETE SO
        BR       EXIT     TRANSFER DR1 → DR0 AND EXIT DIVIDE LOOP
INCC    LDX      1        INCREMENT DR1 BY ADDING 1 TO
        TCY      0        DIGIT 0 AND PROPAGATING THE
D2      IMAC              CARRY IF THE RESULT IS GREATER THAN 9
        TAM               THERE IS NO CHECK IN THIS ROUTINE FOR DIVIDE BY 0.
        ALEC     9
        BR       D1
        TCMIY    0
        BR       D2
```

The following routine uses the TMS 1100/1300 instruction set for integer divide. The ALEC command is replaced by an appropriate AC 1AC command accompanied by a change in the branch logic. For the TMS 1100/1300 DR0 is divided by DR4, and the results are saved in DR0.

LABEL	OP CODE	OPERAND	COMMENT
DIVID	LDX	1	
	CALLL	CREG	
D1	CALLL	S400	SUBTRACT DR4 FROM DR0, RESULTS PLACED IN DR0
	DAN		IF BORROW AT EXIT FROM S400, THE ACCUMULATOR = 1, DIVISION COMPLETE
	BR	TR10	
	LDX	1	IF NO BORROW, ADD ONE TO QUOTIENT REGISTER, DR1
	TCY	0	
DZ	IMAC		
	TAM		
	A6AAC		IF RESULT WAS GREATER THAN 9
	BR	OVER9	
	BR	D1	
OVER9	TCMIY	0	RESULT TO ZERO, INCREMENT NEXT MOST-SIGNIFICANT-WORD
	BR	D2	
TR10	TCY	0	TRANSFER COMPLETED RESULT TO DR0
TR	LDX	1	
	TMA		
	LDX	0	
	TAMIY		
	YNEC	7	
	BR	TR	
	BR	EXIT	

SECTION XIV

EXAMPLE PROGRAM

14-1 GENERAL

In this section all the examples previously demonstrated are combined to form a working TMS1000 program. This example program illustrates the TMS1000 used as an interval timer and performs integer BCD multiplication, division, addition, and subtraction.

14-2 EXAMPLE INPUT/OUTPUT.

The schematic in Figure 14-2.1 illustrates the external connections required.

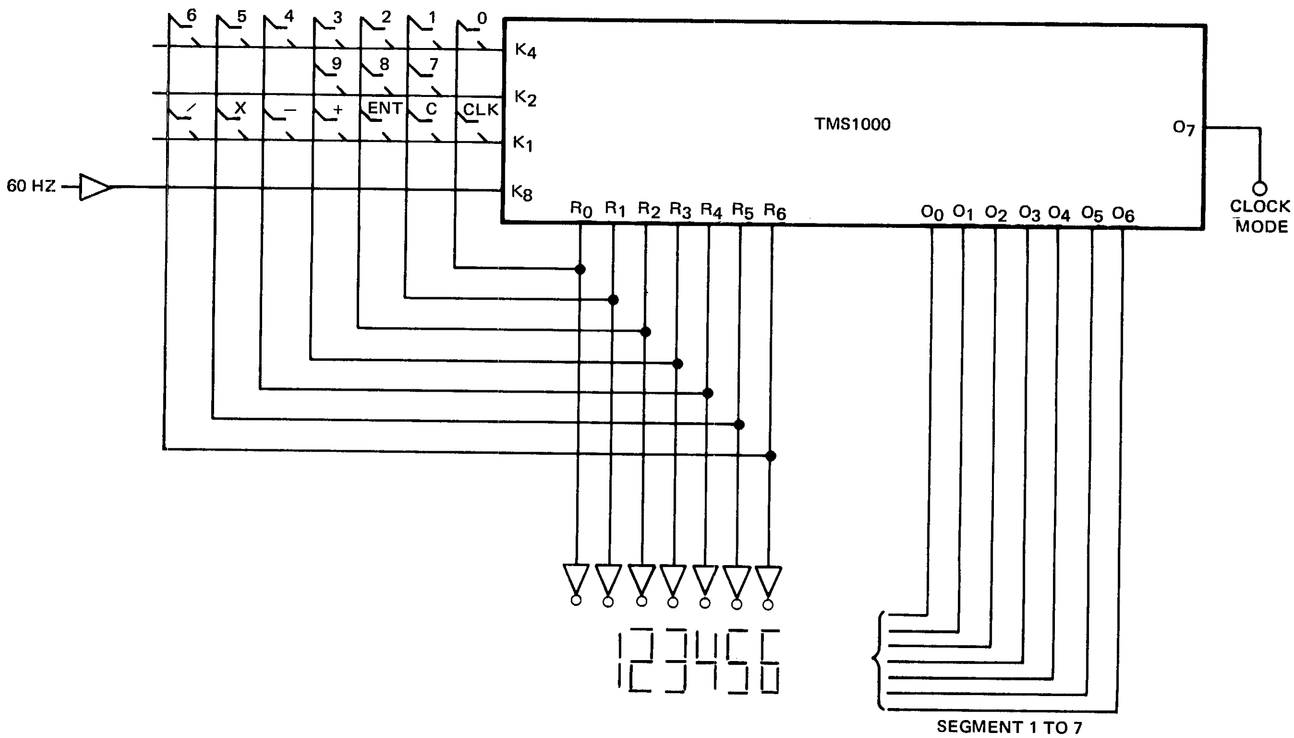
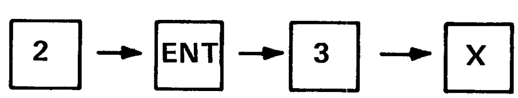


FIGURE 14-2.1. INTERCONNECT SCHEMATIC FOR EXAMPLE PROGRAM

The R lines are used to strobe the display and keyboard. The output PLA encodes the accumulator and status latch into seven segment data.

Data keys are on K4 and K2. Function keys are on K1. A 60 hz clock signal feeds into K8. The function keys are +, –, X, ÷, ENT, and C. To execute a multiply, the key sequence would be:



which would result in a 6 in the display. The C key is used to clear all the working registers.

To use the interval timer feature, enter the amount of time to be counted with the data keys. When the CLK key is depressed, the program will transmit a ONE on O7 which can be used to communicate to external logic that the circuit is in the clock mode. The program then enters a countdown mode that decrements the display every second. During this countdown mode none of the keys are active. When the display has been decremented to zero, O7 is reset, and the keyboard is reactivated. For the purposes of illustration, this program has not been compacted to give a mimimal ROM word count.

14-3 RAM ORGANIZATION.

The example program uses the RAM organization shown in Figure 14-3.1.

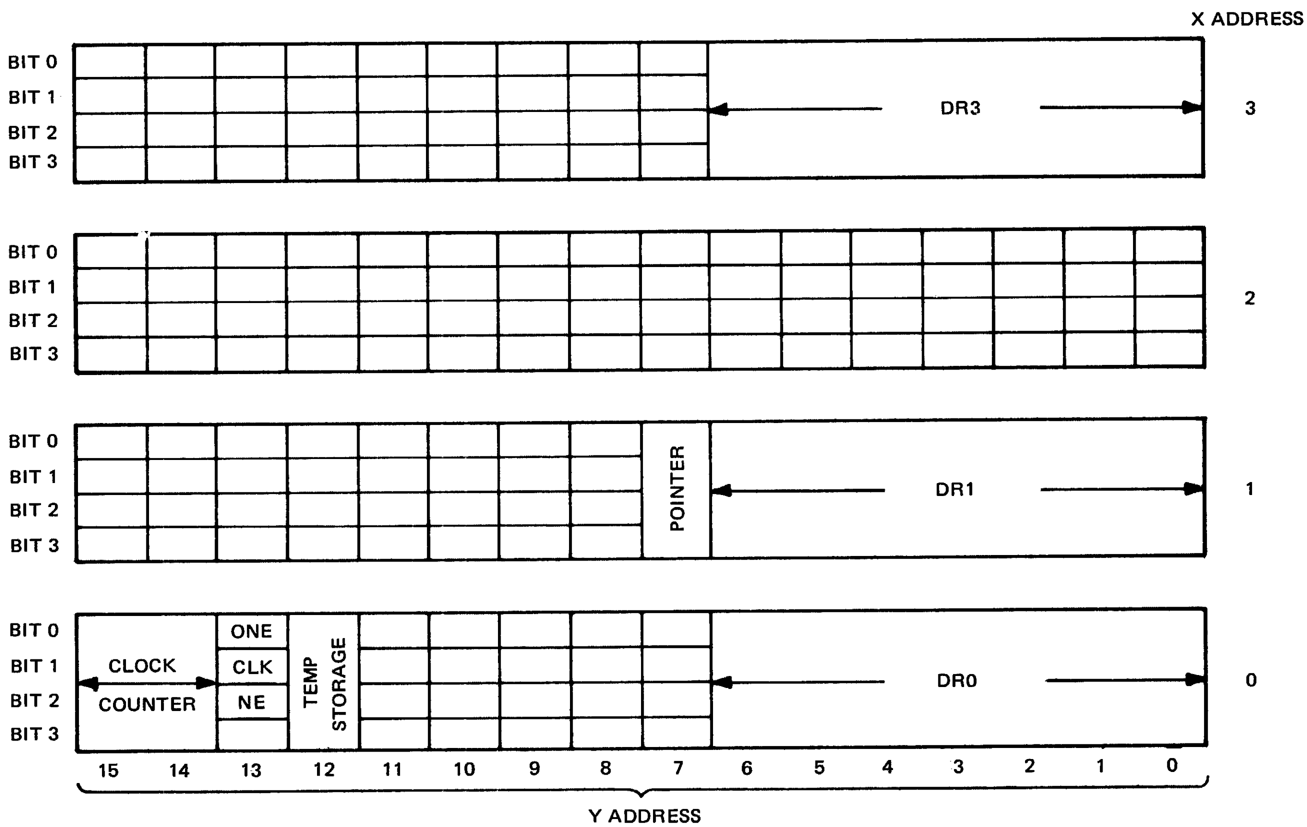


FIGURE 14-3.1. RAM ORGANIZATION FOR EXAMPLE PROGRAM

14-3.1 DATA REGISTERS.

- DR0. Display register. Storage register for data entered and for final results.
- DR3. Storage register for the first factor in a calculation. Data is transferred to DR3 when the ENT key is depressed.
- DR1. Working register used for multiply and divide.

14-3.2 FLAG BITS AT M(0,13).

- Bit 0, the "ONE" flag. In the clock mode, this flag is set when a clock signal is first on and accepted. Prevents double counting of the clock signal. The "ONE" flag is reset when the clock signal goes low.
- Bit 1, the "CLK" flag. This flag differentiates between the two modes of operation, clock and compute. Set when the CLK key is accepted. Reset when the display register is decremented to zero.
- Bit 3, "NE" flag. In the compute mode, this flag is set when new data is being entered into the display register, and reset upon completion of a function. In the data entry routine, this flag is tested before the new data is entered.

If "NE" equals 0, then the DR0 (display) is cleared before the new data is accepted. If "NE" is set, then the DR0 is left shifted with the new data going to the LSD.

14-3.3 TEMPORARY WORKING AREAS

- M(0,12): Used in the display routine to hold the R location that is currently being displayed.
- M(0,14-15): Clock signal counter. In the clock mode, this counter is incremented every clock pulse (every 1/60 second). When this counter equals 60, one second has elapsed.
- M(1,7): Pointer used in the multiply routine.

```
 SSSSSSSSSS      AAAAAAAA     MM        MM   PPPPPPPPPPP    LL             EEEEEEEEEEEE
SSSSSSSSSSSS    AAAAAAAAAA    MMM      MMM   PPPPPPPPPPPP   LL             EEEEEEEEEEEE
SS        SS   AA        AA   MMMM    MMMM   PP        PP   LL             EE
SS             AA        AA   MM MM  MM MM   PP        PP   LL             EE
SS             AA        AA   MM  MMMM  MM   PP        PP   LL             EE
SSSSSSSSSSS    AA        AA   MM   MM   MM   PP        PP   LL             EEEEEEEE
 SSSSSSSSSSS   AAAAAAAAAAAA   MM        MM   PPPPPPPPPPPP   LL             EEEEEEEE
          SS   AAAAAAAAAAAA   MM        MM   PPPPPPPPPPP    LL             EE
          SS   AA        AA   MM        MM   PP             LL             EE
SS        SS   AA        AA   MM        MM   PP             LL             EE
SSSSSSSSSSSS   AA        AA   MM        MM   PP             LLLLLLLLLLLL   EEEEEEEEEEEE
 SSSSSSSSSS    AA        AA   MM        MM   PP             LLLLLLLLLLLL   EEEEEEEEEEEE

          JJ    OOOOOOOOOO    BBBBBBBBBBB                   9999999999     3333333333     9999999999     3333333333
          JJ   OOOOOOOOOOOO   BBBBBBBBBBBB                 999999999999   333333333333   999999999999   333333333333
          JJ   OO        OO   BB        BB                 99        99   33        33   99        99   33        33
          JJ   OO        OO   BB        BB                 99        99             33   99        99             33
          JJ   OO        OO   BB        BB                 99        99             33   99        99             33
          JJ   OO        OO   BBBBBBBBBBB                  999999999999           333    999999999999           333
          JJ   OO        OO   BBBBBBBBBBB                   99999999999           333     99999999999           333
          JJ   OO        OO   BB        BB                           99             33             99             33
JJ        JJ   OO        OO   BB        BB                           99             33             99             33
JJ        JJ   OO        OO   BB        BB                           99   33        33             99   33        33
JJJJJJJJJJJJ   OOOOOOOOOOOO   BBBBBBBBBBBB                 999999999999   333333333333   999999999999   333333333333
 JJJJJJJJJJ     OOOOOOOOOO    BBBBBBBBBBB                  99999999999     3333333333    99999999999     3333333333

RRRRRRRRRRR    TTTTTTTTTTTT   777777777777   BBBBBBBBBBB
RRRRRRRRRRRR   TTTTTTTTTTTT   777777777777   BBBBBBBBBBBB
RR        RR        TT        77        77   BB        BB
RR        RR        TT                 77    BB        BB
RR        RR        TT                77     BB        BB
RRRRRRRRRRRR        TT               77      BBBBBBBBBBB
RRRRRRRRRRR         TT               77      BBBBBBBBBBB
RR    RR            TT              77       BB        BB
RR     RR           TT              77       BB        BB
RR      RR          TT             77        BB        BB
RR       RR         TT             77        BBBBBBBBBBBB
RR        RR        TT             77        BBBBBBBBBBB

FFFFFFFFFFFF   TTTTTTTTTTTT      000000       666666666666   FFFFFFFFFFFF      000000         000000          11
FFFFFFFFFFFF   TTTTTTTTTTTT     00000000     6666666666666   FFFFFFFFFFFF     00000000       00000000        111
FF                  TT         00      00    66              FF              00      00     00      00      1111
FF                  TT        00        00   66              FF             00        00   00        00       11
FF                  TT        00        00   66              FF             00        00   00        00       11
FFFFFFFFF           TT        00        00   66666666666     FFFFFFFFF      00        00   00        00       11
FFFFFFFFF           TT        00        00   666666666666    FFFFFFFFF      00        00   00        00       11
FF                  TT        00        00   66        66    FF             00        00   00        00       11
FF                  TT        00        00   66        66    FF             00        00   00        00       11
FF                  TT         00      00    66        66    FF              00      00     00      00        11
FF                  TT          00000000     666666666666    FF               00000000       00000000       111111
FF                  TT           000000       6666666666     FF                000000         000000        111111
```

14-4

```
TMS1000 ASSEMBLER (VERSION B.1 05/01/75)  11/24/75  14:30:46
COPYRIGHT (C) 1975 TEXAS INSTRUMENTS INCORPORATED
TEXAS INSTRUMENTS OWNS AND IS RESPONSIBLE FOR TMS1000 ASSEMBLER

ROM SOURCE PROGRAM     'SAMPLE'


PAGE NO.  0


AD    LOC  OBJECT CODE        STMT       SOURCE STATEMENT


                               1          TITLE SAMPLE X
                               2          OPTION LIST
                               3          OPTION XREF
                               4          OPTION STAT
                               5          OPTION ROM

      000                      7         ONE   EQU      0
      001                      8         CLK   EQU      1
      003                      9         NE    EQU      3


                              11         *************************************************************************
                              12         *                       DISPLAY AND KEYBOARD SCAN ROUTINE. THIS ROUTINE       *
                              13         *                       SCANS THE KEYBOARD AND DISPLAY, AND DETECTS A         *
                              14         *                       CLOCK INPUT. THERE ARE TWO MODES OF OPERATION,        *
                              15         *                       CALCULATOR (NORMAL MODE) AND CLOCK MODE. IN THE       *
                              16         *                       CALCULATOR MODE THE CLOCK INPUT IS IGNORED, ONLY      *
                              17         *                       KEYS CAN BE ENTERED. IN THE CLOCK MODE, KEY           *
                              18         *                       DEPRESSIONS ARE IGNORED AND ONLY CLOCK SIGNALS        *
                              19         *                       CAN BE INPUT. BIT 1 OF M(0,13), THE CLK FLAG,         *
                              20         *                       WILL BE SET IN THE CLOCK MODE.                        *
                              21         *************************************************************************
000   003  00001000           23         LOCK  TKA               INPUT THE K INPUTS AND MASK OFF K8, THE CLOCK INPUT.
001   004  0111 1110          24         KIN   ALEC     7        IF ANY KEY THAT WAS PREVIOUSLY PUSHED IS STILL DOWN,
003   00C  10 001111          25               BR       K1       THEN CONTROL REMAINS LOCKED IN THIS LOOP.
007   01C  00000001           26               A8AAC             *
00F   03C  00001111           27         K1    RETN              *
01F   03F  0111 0000          28               ALEC     0        *
03F   03E  10 111101          29               BR       RESET    *
03E   039  10 000000          30               BR       LOCK     *

03D   036  0100 0110          32         RESET TCY      6        RESET ANY R LINE THAT MAY STILL BE SET DUE TO A
03B   02E  00001100           33         RE1   RSTR              PREVIOUS KEY PUSH.
037   01E  00101100           34               DYN               *
02F   03D  10 111011          35               BR       RE1      *

01F   038  0100 0110          37         DISP  TCY      6        START THE DISPLAY LOOP.
03C   031  10 000110          38               BR       SCAN2


039   026  00100011           40         SCAN1 TYA               TRANSFER Y, THE CURRENT R LOCATION, TO THE ACCUMULATOR.
033   00F  0100 0011          41               TCY      12       ADDRESS M(0,12).
027   010  00100000           42               TAMIY             STORE THE R LOCATION IN M(0,12) WHILE ADDRESSING
                              43         *                       M(0,13).
00E   038  00001000           44               TKA               TRANSFER THE K INPUTS TO THE ACCUMULATOR.
010   037  001110 10          45               TBIT1    CLK      TEST THE CLOCK FLAG TO CHECK THE CURRENT OPERATING
03A   029  10 100100          46               BR       CLOK     MODE (CLOCK OR CALCULATE).
035   016  11 000001          47               CALL     KIN      IF CALCULATE, CALL KIN WHICH MASKS OFF THE CLOCK INPUT
02B   020  0111 0000          48               ALEC     0        IF NO KEY INPUT, ACC = 0, BRANCH TO NOK.
016   018  10 111000          49               BR       NOK
```

```
02C   032  00000111       DELAY DAN               IF KEY INPUT, ENTER THE DELAY LOOP.
018   020  10 101100            BR       DELAY    *
030   001  00101100             DYN               *
021   005  10 101100            BR       DELAY    *
002   008  11 000000            CALL     LOCK     NOW TEST THE INPUTS AGAIN (MASKING OFF THE CLOCK INPUT)
005   014  0111 0000            ALEC     0        IF NO KEY INPUT, THEN THE INPUT WAS CAUSED BY
00B   02C  10 111000            BR       NOK      NOISE OR LEADING EDGE KEY BOUNCE.
017   01F  00010100             BL       KEY      IF THE KEY IS STILL DOWN, THEN IT IS A VALID INPUT.
02F   03A  10 000000

01C   030  001101 00      NOCLK RBIT     ONE      BRANCH HERE IN CLOCK MODE WITH NO CLOCK INPUT.
038   021  0100 0011      NOK   TCY      12       SET Y = CURRENT R LOCATION.
031   006  00100010             TMY               *
023   000  00101100             DYN               TRANSFER M(0,Y-1) CONTENTS TO THE ACCUMULATOR.
                                FETCH    1
006   013  00100001       SCAN2 TMA               *
00D   034  00101011             IYC               RESET R(Y).
01B   02F  00001100             RSTR              *
036   019  00101100             DYN               SET Y = Y-1.
02D   035  00001010             TDO               TRANSFER THE ACCUMULATOR AND STATUS LATCH
                          *                       TO THE OUTPUT PLA.
01A   023  00001101             SETR              SET R(Y-1).
034   011  0101 1111            YNEC     15       CONTINUE UNTIL Y = 15.
029   025  10 111001            BR       SCAN1    *
012   008  10 011110            BR       DISP     START A NEW LOOP WHEN Y ='S 15.

024   012  0111 1110      CLOK  ALEC     7        BRANCH HERE IF IN THE CLOCK MODE.
008   023  10 011100            BR       NOCLK    IF NO CLOCK INPUT, K8 = 0, BRANCH TO NOCLK.
011   007  001110 00            TBIT1    ONE      IF CLOCK INPUT, CHECK THE ONE FLAG TO INSURE THAT
022   00A  10 111000            BR       NOK      A PULSE IS NOT COUNTED TWICE.

004   013  001100 00            SBIT     ONE      IF THE ONE FLAG IS NOT SET, SET THE ONE FLAG AND
009   024  0100 0110            TCY      6        START THE TIMER UPDATE.
013   00F  00011000       TSTZ  LDP      1        FIRST CHECK TO SEE IF THE A REGISTER IS ZERO.
026   01A  00100110             MNEZ              *
00C   033  10 000000            BR       CLK1     IF NON-ZERO, BRANCH TO CLK1.
019   027  00010000             LDP      0        *
032   009  00101100             DYN               *
025   015  10 010011            BR       TSTZ     *

                          *                       IF THE A REGISTER IS ZERO, RESET THE STATUS LATCH
                          *                       WHICH WILL RESET O7 WHEN THE NEXT TDO OCCURS, AND
                          *                       ALSO CLEAR THE CLK AND ONE FLAGS.
00A   028  00100011             TYA               SET Y = ACCUMULATOR.
015   017  00000010             YNEA              THIS INSTRUCTION WILL THEN RESET THE STATUS LATCH.
                          *                       SINCE Y IS NOW = THE ACCUMULATOR.
02A   02A  0100 1011            TCY      13       ADDRESS THE CONTROL FLAGS IN M(0,13).
014   010  0110 0000            TCMIY    0        CLEAR CLK AND ONE FLAGS.
028   022  10 111000            BR       NOK      RETURN TO THE DISPLAY LOOP, NOW IN THE CALCULATE MODE.
010   000  10 000000               BR    LOCK        ***** NO-OP INSTRUCTION *****
020   002  10 000000               BR    LOCK        ***** NO-OP INSTRUCTION *****
                          *  THE PROGRAM COUNTER SEQUENCE IS SHOWN ENTIRELY FOR ALL 64 INSTRUCTIONS.
                          *  THE PROGRAM COUNTER IS IN THE COLUMN ON THE LEFT.  TO THE IMMEDIATE RIGHT
                          *  THE LOCATION INDEX ( LOC ) GIVES THE SEQUENCED ADDRESS OF THE INSTRUCTION
                          *  IN THE ROM OBJECT CODE.  THE 'OPTION ROM' OUTPUT IS PRINTED AT THE END OF AN
                          *  ASSEMBLY RUN.
                                PAGE


**********************************************************************************


PAGE NUMBER  0 CONTAINS 64 ROM INSTRUCTIONS.
```

PAGE NO. 1

```
PAD   LOC  OBJECT CODE        STMT       SOURCE STATEMENT

                               108       ****************************************************************************************
                               109       *                          UPDATE TIMER. IF A IS NON-ZERO, INCREMENT THE CLOCK     *
                               110       *                          COUNTER FIELD M(0,14) AND M(0,15) TILL IT OVERFLOWS     *
                               111       *                          (='S 60). WHEN THIS OCCURS, 1 SECOND HAS ELAPSED AND    *
                               112       *                          THE A REGISTER IS DECREMENTED.                          *
                               113       ****************************************************************************************

000   043  0100 0111           115       CLK1  TCY      14        INCREMENT M(0,14).
001   044  00101000            116             IMAC               *
003   04C  0111 1001           117             ALEC     9         IF LESS THAN 10, RETURN TO THE DISPLAY LOOP.
007   05C  10 111100           118             BR       CLK3
00F   07C  0110 0000           119             TCMIY    0         IF GREATER THAN 9, SET M(0,14)=0 AND INCREMENT M(0,15).
01F   07F  00101000            120             IMAC               *
03F   07E  0111 1010           121             ALEC     5         IF LESS THAN 6, RETURN TO THE DISPLAY LOOP.
03F   079  10 111100           122             BR       CLK3

03D   076  0110 0000           124             TCMIY    0         IF GREATER THAN 5, THEN OVERFLOW.  SET M(0,15) =0
03B   06E  00101010            125       CLK2  DMAN               WHILE ADDRESSING M(0,0). M(0,Y) - 1 TO ACCUMULATOR.
037   05E  10 111100           126             BR       CLK3      IF NON-ZERO, RETURN TO THE DISPLAY LOOP.
02F   07D  0110 1001           127             TCMIY    9         IF ZERO, SET = 9 AND ADDRESS THE HIGHER ORDER DIGIT.
01E   078  10 111011           128             BR       CLK2


03C   071  00000011            130       CLK3  TAM
039   066  00010000            131             BL       NOK
033   04F  10 111000           132
                               133             PAGE

******************************************************************************

PAGE NUMBER  1 CONTAINS 16 ROM INSTRUCTIONS.
```

```
PAD  LOC  OBJECT CODE    SOURCE STATEMENT

                         ************************************************************************************
                         *                                  VALID KEY ENTRY. FIRST CLEAR THE BLANK CODES    *
                         *                                  FROM THE A REGISTER. THEN DECODE THE KEYS.      *
                         ************************************************************************************
000  093  0100 0110      KEY   TCY    6          CLEAR THE 15'S FROM THE A REGISTER.
001  084  00101110       CLEN  XMA               *
003  08C  0111 1001            ALEC   9          *
007  09C  10 011111            BR     CLEN1      *
00F  0BC  00101111             CLA               *
01F  0BF  00101110       CLEN1 XMA               *
03F  0BE  00101100             DYN               *
03E  089  10 000001            BR     CLEN       *

03D  086  0100 1011            TCY    13         ADDRESS M(0,13) WHICH HOLDS THE CONTROL FLAGS.
03B  0AE  00011100             LDP    3          THE ACCUMULATOR STILL HOLDS THE K INPUT VALUE.
037  095  0111 1000            ALEC   1          IF THE ACC = 1, THEN A FUNCTION KEY ON K1 WAS DEPRESSED
02F  080  10 000000            BR     FUNC

                         ************************************************************************************
                         *                                  DATA KEY ENTRY.                                 *
                         ************************************************************************************
01E  0B8  00010100             LDP    2          RESET THE PAGE BUFFER REGISTER = 2.
03C  081  001110 11            TBIT1  NE         TEST THE NUMBER ENTRY FLAG. IF ZERO,
039  0A6  10 011101            BR     NE=1       THEN CLEAR THE A REGISTER.
033  08E  001100 11            SBIT   NE         AND SET THE NUMBER ENTRY FLAG. IF THE NUMBER
027  090  00011111             CALLL  CREG       ENTRY FLAG IS 1, BYPASS THIS SUBROUTINE CALL.
00F  098  11 110111
01D  087  0100 0011      NE=1  TCY    12         ADDRESS M(0,12) WHICH CONTAINS THE KEY'S R LOCATION.
03A  0A9  00101110             XMA               TRANSFER THE R LOCATION TO THE ACCUMULATOR AND THE K
035  096  001110 01            TBIT1  2          LOCATION TO M(0,12). NUMBER KEYS ON K4 = THEIR R
02B  0A0  10 101100            BR     K4IN       LOCATION. THE 2 KEY IS ON R2, THE 3 KEY IS ON R3, ETC.
                         *                       IF THE KEY IS ON K4, THEN BIT 2 WILL BE SET. IF NOT,
                         *                       THEN THE KEY IS ON K2. IF THE KEY IS ON K2, ADD 6
                         *                       TO THE R LOCATION TO GET THE KEYS VALUE.
016  098  00000110             A6AAC             THE 7 KEY IS ON R1, THE 8 KEY IS ON R2, ETC.
02C  082  0100 1010      K4IN  TCY    5          TEST THE MSD DIGIT. IF NON-ZERO, THEN THE REGISTER
018  0A0  00100110             MNEZ              IS FULL SO EXIT.
030  081  10 000101            BR     EXNUM
021  085  00011110             CALLL  LDATA      LEFT SHIFT THE A REGISTER. ENTER THE ACCUMULATOR TO
002  083  11 111010
005  094  00011111       EXNUM BL     BLANK      THE LSD. RETURN TO THE BLANK ROUTINE.
00B  0AC  10 110011
                               PAGE

*****************************************************************************************

PAGE NUMBER  2 CONTAINS 30 ROM INSTRUCTIONS.
```

```
PAD   LOC  OBJECT CODE      STMT    SOURCE STATEMENT

                             177    ***************************************************************************
                             178    *                              FUNCTION KEY DECODE.                       *
                             179    ***************************************************************************

000   0C3  001101 11         181    FUNC  RBIT    NE       IF ANY FUNCTION KEY IS DEPRESSED, RESET THE NE FLAG.
001   0C4  0100 0011         182          TCY     12       ADDRESS M(0,12), WHICH HOLDS THE KEY'S R LOCATION.
003   0CC  00101010          183          DMAN             IF THE R LOCATION IS NOT ZERO, BRANCH TO NOT=0.
007   0DC  10 111011         184          BR      NOT=0    THE ACCUMULATOR ='S THE R LOCATION - 1.

                             186    *                          IF M(0,12) IS ZERO, THEN THIS IS THE CLOCK KEY.
00F   0FC  0100 1011         187          TCY     13       ADDRESS M(0,13), WHICH HOLDS THE CONTROL FLAGS.
01F   0FF  001100 10         188          SBIT    CLK      SET THE CLOCK FLAG.
03F   0FF  00000010          189                  YNEA     SINCE Y = 13 AND THE ACCUMULATOR = 15, THIS INSTRUCTION
                             190    *                          WILL SET THE STATUS LATCH, WHICH WILL SET O7 WHEN
                             191    *                          THE NEXT TDO OCCURS.
03F   0F9  00010000          192          BL      LOCK     RETURN TO THE DISPLAY ROUTINE.
03D   0F6  10 000000         193

03B   0EE  00011111          195    NOT=0 LDP     15       IF THE ACCUMULATOR = 0, THEN THIS IS THE CLEAR KEY.
037   0DE  0111 0000         196          ALEC    0
02F   0FD  10 000111         197          BR      CLER
01E   0F8  00010110          198          LDP     6        IF THE ACCUMULATOR = 1, THEN THIS IS THE ENTER KEY.
03C   0F1  0111 1000         199          ALEC    1
039   0F6  10 111101         200          BR      TRAB
033   0CE  00010010          201          LDP     4        IF THE ACCUMULATOR = 2, THEN THIS IS THE + KEY.
027   0DD  0111 0100         202          ALEC    2
00E   0FB  10 000000         203          BR      PLUS
01D   0F7  0111 1100         204          ALEC    3        IF THE ACCUMULATOR = 3, THEN THIS IS THE - KEY.
03A   0E9  10 001111         205          BR      MINS
035   0D6  00011110          206          LDP     7        IF THE ACCUMULATOR = 4, THEN THIS IS THE * KEY.
02B   0ED  0111 0010         207          ALEC    4
016   0D8  10 000000         208          BR      MULT
                             209    *                          IF THE ACCUMULATOR = 5, THEN THIS IS THE / KEY.

                             211    ***************************************************************************
                             212    *                              DIVIDE ROUTINE                             *
                             213    ***************************************************************************
02C   0F2  001111 10         214    DIVID LDX     1        CLEAR THE C REGISTER.
018   0E0  00011111          215          CALLL   CREG
030   0C1  11 110111         216
021   0C5  00011010          217    D1    CALLL   SBAA     SUBTRACT B FROM A.
002   0CB  11 111010         218
005   0D4  0111 0000         219          ALEC    0        IF NO BORROW, THEN UPON EXIT FROM SBAA,
00B   0EC  10 110110         220          BR      INCC     THE ACCUMULATOR WILL = 0. IF SO, BRANCH TO INCC.
017   0DF  0100 0000         221          TCY     0        TRANSFER C TO A.
02E   0FA  001111 10         222    TRCA  LDX     1        *
01C   0F0  00100001          223          TMA              *
038   0F1  001111 00         224          LDX     0        *
031   0C6  00100000          225          TAMIY            *
023   0CD  0101 1110         226          YNEC    7        *
006   0DB  10 101110         227          BR      TRCA     *
00D   0F4  00011111          228          BL      BLANK    AND EXIT THE DIVIDE LOOP.
01B   0FF  10 110011         229
036   0D9  001111 10         230    INCC  LDX     1        INCREMENT C BY ADDING 1 TO DIGIT 0 AND PROPAGATING
02D   0F5  0100 0000         231          TCY     0        THE CARRY IF THE RESULT IS GREATER THAN 9.
01A   0F8  00101000          232    D2    IMAC
034   0D1  00000011          233          TAM              THERE IS NO CHECK IN THIS ROUTINE FOR A DIVIDE BY 0.
029   0E5  0111 1001         234          ALEC    9
012   0C8  10 100001         235          BR      D1
024   0D2  0110 0000         236          TCMIY   0
008   0F3  10 011010         237          BR      D2
                             238          PAGE

***************************************************************************

PAGE NUMBER  3 CONTAINS 47 ROM INSTRUCTIONS.
```

```
PAGE NO.  4

PAD   LOC  OBJECT CODE        STMT      SOURCE STATEMENT

                               239      ***************************************************************************************
                               240      *                            + KEY.                                                   *
                               241      ***************************************************************************************
000   103  00011010            242      PLUS  CALLL     AABA      ADD B TO A.
001   104  11 000000           243
003   10C  00011111            244      TOB   BL        BLANK     RETURN TO THE BLANK ROUTINE.
007   11C  10 110011           245

                               247      ***************************************************************************************
                               248      *                            - KEY.                                                   *
                               249      ***************************************************************************************
00F   13C  00011010            250      MINS  CALLL     SBAA      SUBTRACT B FROM A. IF A BORROW OCCURS, THEN UPON
01F   13F  11 111010           251
03F   13F  0111 0000           252            ALEC      0         EXIT FROM SBAA, THE ACCUMULATOR WILL BE NON-ZERO.
03F   139  10 000011           253            BR        TOB       IF NO BORROW, RETURN TO THE BLANK ROUTINE.
03D   136  00011111            254            CALLL     CREG      IF A BORROW OCCURS, CLEAR THE A REG.
03B   12E  11 110111           255
037   11E  10 000011           256            BR        TOB       AND RETURN TO THE BLANK ROUTINE.
                               257            PAGE

******************************************************************************

PAGE NUMBER  4 CONTAINS 11 ROM INSTRUCTIONS.
```

```
PAD   LOC  OBJECT CODE         STMT       SOURCE STATEMENT

                                258       *******************************************************************************
                                259       *                                     REGISTER ADDITION SUBROUTINE                *
                                260       *******************************************************************************
000   143  001111 00            261       AA3A  LDX       0           TO SUM B INTO A, INITIALIZE X = 0.

001   144  0100 0000            263             TCY       0           INITIALIZE Y = 0.
003   14C  00101111             264             CLA                   CLEAR THE ACCUMULATOR WHICH WILL BE USED AS CARRY.
007   15C  00000000             265       AD1   COMX                  COMPLIMENT X.
00F   17C  00100101             266             AMAAC                 ADD M(XBAR,Y) TO THE ACCUMULATOR (CARRY).
01F   17F  00000000             267             COMX                  COMPLIMENT X.
03F   179  00100101             268             AMAAC                 ADD DIGITS M(X,Y) + (M(XBAR,Y) + CARRY).
03E   179  10 110111            269             BR        GT9         BRANCH IF THE SUM IS GREATER THAN 15.
03D   176  0111 1001            270             ALEC      9           NOW TEST FOR A SUM LESS THAN 10. IF SO,
03B   16E  10 001110            271             BR        LT10        BRANCH TO LT10.
037   15F  00000110             272       GT9   A6AAC                 IF THE SUM IS GREATER THAN 9, ADD 6.
02F   170  00000100             273             TAMZA                 TRANSFER THE CORRECTED SUM TO M(X,Y) AND CLEAR THE ACC.
01E   178  00001110             274             IA                    SET THE ACCUMULATOR (CARRY DIGIT) = 1.
03C   171  00101011             275       INCY  IYC                   INCREMENT Y.
039   166  0101 1110            276             YNEC      7           CONTINUE ADDING UNTIL Y = 7.
033   14F  10 000111            277             BR        AD1
027   150  00001111             278             RETN
03E   172  00000100             279       LT10  TAMZA                 FOR SUMS LESS THAN 10, TRANSFER THE ACCUMULATOR TO
01D   177  10 111100            280             BR        INCY        M(X,Y) AND CLEAR THE ACCUMULATOR (CARRY DIGIT).

                                282       *******************************************************************************
                                283       *                                     REGISTER SUBTRACTION SUBROUTINE.            *
                                284       *******************************************************************************
03A   169  001111 00            285       SBAA  LDX       0           TO SUBTRACT B FROM A, INITIALIZE X = 0.

035   156  0100 0000            287             TCY       0           INITIALIZE Y = 0.
02B   160  00101111             288             CLA                   CLEAR THE ACCUMULATOR, THE BORROW DIGIT.
016   158  00000000             289       S1    COMX                  COMPLIMENT X, ADDRESS THE SUBTRAHEND.
02C   172  00100101             290             AMAAC                 ADD SUBTRAHEND (Y) + BORROW.
018   160  00000000             291             COMX                  COMPLIMENT X, ADDRESS THE MINUEND.
030   141  00100111             292             SAMAN                 MINUEND(Y) - (SUBTRAHEND(Y) + BORROW) TO ACCUMULATOR.
021   145  10 110001            293             BR        NOBOR       BRANCH IF NO BORROW OCCURS.
002   148  00000101             294             A10AAC                IF BORROW, ADD CORRECTION, + 10.
005   154  00000100             295             TAMZA                 TRANSFER THE RESULT TO M(X,Y) AND CLEAR THE ACCUMULATOR
00B   16C  00001110             296             IA                    SET THE ACCUMULATOR (BORROW DIGIT) = 1.
017   15F  00101011             297       INCYS IYC                   INCREMENT Y.
02E   17A  0101 1110            298             YNEC      7           UNTIL Y = 7.
01C   170  10 010110            299             BR        S1
038   161  00001111             300             RETN

031   146  00000100             302       NOBOR TAMZA                 IN THE NO BORROW CASE, TRANSFER THE RESULT TO M(X,Y).
023   140  10 010111            303             BR        INCYS       CLEAR THE ACCUMULATOR (BORROW DIGIT).
                                304             PAGE

**************************************************************************************

PAGE NUMBER  5 CONTAINS 36 ROM INSTRUCTIONS.
```

PAGE NO. 6

```
PAD   LOC  OBJECT CODE        STMT    SOURCE STATEMENT

                              305     ********************************************************************************
                              306     *                                TRANSFER THE A REGISTER CONTENTS TO C SUBROUTINE.    *
                              307     ********************************************************************************
000   183  0100 0000          308     TRAC  TCY     0        INITIALIZE Y = 0.
001   184  001111 00          309     T1    LDX     0        SET X = 0.
003   18C  00100001           310           TMA              TRANSFER M(0,Y) TO THE ACCUMULATOR.
007   19C  001111 10          311           LDX     1        SET X = 1.
00F   1BC  00100000           312           TAMIY            TRANSFER THE ACCUMULATOR TO M(1,Y) AND INCREMENT Y.
01F   1BF  0101 1110          313           YNEC    7        CONTINUE UNTIL Y = 7.
03F   1BF  10 000001          314           BR      T1
03F   1B9  00001111           315           RETN

                              317     ********************************************************************************
                              318     *                                COMPLEMENTED REGISTER TRANSFER SUBROUTINE.           *
                              319     ********************************************************************************
030   1B6  001111 11          320     TRAB  LDX     3        TO TRANSFER A TO B, INITIALIZE X = 3.
03B   1AE  0100 0000          321     TRO   TCY     0        INITIALIZE Y = 0.
037   19E  00000000           322     T2    COMX             COMPLEMENT X.
02F   1BD  00100001           323           TMA              TRANSFER M(X,Y) TO ACCUMULATOR.
01E   1B8  00000000           324           COMX             COMPLEMENT X.
03C   1B1  00100000           325           TAMIY            TRANSFER THE ACCUMULATOR TO M(XBAR,Y), AND INCREMENT Y.
039   1A6  0101 1110          326           YNEC    7        TRANSFER UNTIL Y = 7.
033   18F  10 110111          327           BR      T2
027   19D  00001111           328           RETN
00E   1BB  00011111           329           BL      BLANK    THIS SUBROUTINE IS USED AS AN ENTER KEY.
01D   1B7  10 110011          330

                              332     ********************************************************************************
                              333     *                                REGISTER RIGHT SHIFT SUBROUTINE.                     *
                              334     ********************************************************************************
03A   1A9  00101111           335     RSHFT CLA                THE ACCUMULATOR, WHICH IS SET = 0, IS TRANSFERRED TO
035   196  0100 0110          336           TCY     6        THE MSB. INITIALIZE Y = 6 (MSB LOCATION).
02B   1AD  00101110           337     R1    XMA              EXCHANGE MEMORY CONTENTS AND THE ACCUMULATOR.
016   198  00101100           338           DYN              DECREMENT Y TO SHIFT THE NEXT DIGIT.
02C   1B2  10 101011          339           BR      R1       CONTINUE UNTIL Y EQUALS ZERO.
018   1A0  00001111           340           RETN
                              341           PAGE

***************************************************************************

PAGE NUMBER  6 CONTAINS 25 ROM INSTRUCTIONS.
```

```
PAD  LOC  OBJECT CODE     STMT    SOURCE STATEMENT

                          342     *********************************************************************************
                          343     *                               MULTIPLY ROUTINE                                *
                          344     *********************************************************************************
000  1C3  00010110        345     MULT  CALLL    TRAC     TRANSFER THE CONTENTS OF A TO C.
001  1C4  11 000000       346
003  1CC  0110 0110       347           TCMIY    6        UPON EXIT FROM TRAC, X = 1 AND Y = 7. A POINTER WILL
                          348     *                       BE STORED THERE THAT WILL BE USED TO ADDRESS THE DIGIT
                          349     *                       IN C. INITIALIZE THE POINTER = 6.
007  1DC  001111 00       350           LDX      0        CLEAR THE A REGISTER.
00F  1FC  00011111        351           CALLL    CREG
01F  1FF  11 110111       352
03F  1FF  001111 10       353     ML2   LDX      1        ADDRESS THE POINTER.
03E  1F9  0100 1110       354           TCY      7
03D  1F6  00100010        355           TMY               SET Y = POINTER.
03B  1FF  00101010        356           DMAN              C(POINTER) - 1 TO THE ACCUMULATOR.
037  1DE  10 100001       357           BR       NOBR     BRANCH TO NOBR IF C(POINTER) IS NON-ZERO.
02F  1FD  0100 1110       358           TCY      7        IF C(POINTER) = 0, SET THE POINTER = POINTER - 1.
01F  1F8  00101010        359           DMAN              POINTER - 1 TO THE ACCUMULATOR.
03C  1F1  10 100111       360           BR       ML1      BRANCH TO ML1 IF THE POINTER IS NON-ZERO.
039  1F6  00011111        361           BL       BLANK    IF ZERO, THEN MULTIPLICATION IS COMPLETE.
033  1CF  10 110011       362
027  1D0  00000011        363     ML1   TAM               TRANSFER THE ACCUMULATOR TO THE POINTER.
00E  1FB  001111 00       364           LDX      0        SHIFT A LEFT.

                          366     *********************************************************************************
                          367     *                       REGISTER LEFT SHIFT SUBROUTINE. THIS SUBROUTINE         *
                          368     *                       LEFT SHIFTS DIGITS 0 THRU 6 OF A GIVEN FILE. IF         *
                          369     *                       ENTERED AT LSHFT, 0 IS TRANSFERRED TO THE LSD. IF       *
                          370     *                       ENTERED AT LDATA, THE ACC WILL BE TRANSFERRED           *
                          371     *                       TO THE LSD.                                             *
                          372     *********************************************************************************
01D  1F7  00101111        373     LSHFT CLA               FOR ENTRY AT LSHFT, INITIALIZE ACC = 0.

03A  1F9  0100 0000       375     LDATA TCY      0        SET Y = LSD.
035  1D6  00101110        376     L1    XMA               EXCHANGE MEMORY AND THE ACCUMULATOR.
02B  1ED  00101011        377           IYC               INCREMENT Y.
016  1D8  0101 1110       378           YNEC     7        KEEP EXCHANGING UNTIL Y = 7.
02C  1F2  10 110101       379           BR       L1
018  1F0  00001111        380           RETN

030  1C1  10 111111       382           BR       ML2
021  1C5  00000011        383     NOBR  TAM               IF C(POINTER) WAS NON-ZERO, TRANSFER THE ACCUMULATOR
                          384     *                       TO C(POINTER).
002  1C3  00011010        385           CALLL    AABA     AND ADD B TO A.
005  1D4  11 000000       386
00B  1EC  10 111111       387           BR       ML2
                          388           PAGE

*************************************************************

PAGE NUMBER  7 CONTAINS 30 ROM INSTRUCTIONS.
```

PAGE NO. 15

```
PAD   LOC  OBJECT CODE        STMT      SOURCE STATEMENT

                               396      ******************************************************************************
                               397      *                                    POWER UP CLEAR ROUTINE.                 *
                               398      *                                    WHEN POWER IS FIRST APPLIED, CONTROL IS PASSED TO    *
                               399      *                                    PAGE 15, LOCATION 00. FOR THIS EXAMPLE, THE WORKING  *
                               400      *                                    REGISTERS WILL BE CLEARED AND ALL THE FLAGS IN FILE  *
                               401      *                                    0 WILL BE RESET.                        *
                               402      ******************************************************************************
000   3C3  00101111            404            CLA                    THIS PROGRAM REQUIRES THE STATUS LATCH = 0 ON POWER
001   3C4  0100 0000           405            TCY     0              UP. THE FIRST TDO INSTRUCTION WILL THEN SET 07 = 0.
003   3CC  00000010            406            YNEA
007   3DC  001111 11           407      CLER  LDX     3              CLEAR THE B REGISTER.
00F   3FC  11 110111           408            CALL    CREG
01F   3FF  001111 10           409            LDX     1              CLEAR THE C REGISTER.
03F   3FE  11 110111           410            CALL    CREG
03E   3F9  001111 00           411            LDX     0              CLEAR ALL OF FILE 0, INCLUDING THE A REGISTER.

03D   3F6  0100 1110           413      CLALL TCY     7              CLEAR ALL DIGITS OF A FILE SUBROUTINE.
03B   3EE  10 101111           414            BR      C1

037   3DE  0100 0000           416      CREG  TCY     0              CLEAR REGISTER SUBROUTINE. CLEAR DIGITS 0 THRU 6
02F   3FD  0110 0000           417      C1    TCMIY   0              *
01E   3F8  0101 1110           418            YNEC    7              *
03C   3F1  10 101111           419            BR      C1             *
039   3E6  00001111            420            RETN                   *

                               422      ******************************************************************************
                               423      *                                    LEADING ZERO SUPPRESSION ROUTINE. STORE A BLANK CODE  *
                               424      *                                    (15) IN DIGITS WHICH HOLD LEADING ZEROES.             *
                               425      ******************************************************************************
033   3CE  001111 00           426      BLANK LDX     0              ADDRESS M(0,6), THE MSD OF THE DISPLAY REGISTER.
027   3DD  0100 0110           427            TCY     6              (THE A REGISTER)
00E   3F3  00100110            428      BL1   MNEZ                   TEST FOR A NON ZERO DIGIT.
01D   3F7  10 101100           429            BR      BRLCK          WHEN A NON ZERO DIGIT IS FOUND, BRANCH TO THE DISPLAY
03A   3E9  0110 1111           430            TCMIY   15             ROUTINE. IF THE DIGIT IS ZERO, STORE A 15.
035   3D6  00101100            431            DYN
02B   3ED  00101100            432            DYN
016   3D8  10 001110           433            BR      BL1            CONTINUE UNTIL ALL OF THE DIGITS HAVE BEEN TESTED.
02C   3F2  00010000            434      BRLCK BL      LOCK
018   3E0  10 000000           435
                               436               END

**********************************************************************************

PAGE NUMBER 15 CONTAINS 25 ROM INSTRUCTIONS.
```

```
END OF SOURCE PROGRAM DATA

NO. OF ERRORS IN ASSEMBLY=     0


PAGE NO.      NO. OF INSTRUCTIONS

   0                64
   1                16
   2                30
   3                47
   4                11
   5                36
   6                25
   7                30
   8                 0
   9                 0
  10                 0
  11                 0
  12                 0
  13                 0
  14                 0
  15                25

TOTAL NO. OF INSTRUCTIONS =   284
```

CROSS REFERENCE TABLE

SYMBOL	VALUE	DEFN	REFERENCES								
AABA	0140	261	242	261	385						
AD1	0147	265	265	277							
ALEC	0070	OPC	196	199	202	204	207	219	234	252	270
			24	28	48	56	77	117	121	140	149
AMAAC	0025	OPC	266	268	290						
A10AAC	0005	OPC	294								
A6AAC	0006	OPC	168	272							
A8AAC	0001	OPC	26								
BL	0080	OPC	58	131	174	192	228	244	329	361	434
BLANK	03F3	426	174	228	244	329	361	426			
BL1	03CE	428	428	433							
BR	0080	OPC	357	360	379	382	387	414	419	429	433
			271	277	280	293	299	303	314	327	339
			205	208	220	227	235	237	253	256	269
			145	150	157	164	171	184	197	200	203
			89	99	100	101	118	122	126	128	141
			49	52	54	57	74	75	78	80	86
			25	29	30	35	38	46			
BRLCK	03EC	434	429	434							
CALL	00C0	OPC	47	55	408	410					
CALLL	00C0	OPC	172	215	217	242	250	254	345	351	385
			159								
CLA	002F	OPC	142	264	288	335	373	404			
CLALL	03FD	413	413								
CLEN	0081	139	139	145							
CLEN1	009F	143	141	143							
CLER	03C7	407	197	407							
CLK	0001	8	8	45	188						
CLK1	0040	115	86	115							
CLK2	007B	125	125	128							
CLK3	007C	130	118	122	126	130					
CLOK	0024	77	46	77							
COMX	0000	OPC	265	267	289	291	322	324			
CREG	03F7	416	159	215	254	351	408	410	416		
C1	03EF	417	414	417	419						
DAN	0007	OPC	51								
DELAY	002C	51	51	52	54						
DISP	001E	37	37	75							
DIVID	00EC	214	214								
DMAN	002A	OPC	125	183	356	359					
DYN	002C	OPC	34	53	64	69	88	144	338	431	432
D1	00E1	217	217	235							
D2	00DA	232	232	237							
END	0000	OPC	436								
EQU	0000	OPC	7	8	9						
EXNUM	0085	174	171	174							
FETCH	0000	OPC	65								
FUNC	00C0	181	150	181							
GT9	0177	272	269	272							
IA	000E	OPC	274	296							
IMAC	0028	OPC	116	120	232						
INCC	00F6	230	220	230							
INCY	017C	275	275	280							
INCYS	0157	297	297	303							
IYC	002B	OPC	67	275	297	377					
KEY	0080	138	58	138							
KIN	0001	24	24	47							
K1	000F	27	25	27							

LDATA	01FA	375	172	375							
LDP	0010	OPC	84	87	148	155	195	198	201	206	
LDX	003C	OPC	311	320	350	353	364	407	409	411	426
			214	222	224	230	261	285	309		
LOCK	0000	23	23	30	55	100	101	192	434		
LSHFT	01DD	373	373								
LT10	014E	279	271	279							
L1	01F5	376	376	379							
MINS	010F	250	205	250							
ML1	01E7	363	360	363							
ML2	01FF	353	353	382	387						
MNEZ	0026	OPC	85	170	428						
MULT	01C0	345	208	345							
NE	0003	9	9	156	158	181					
NE=1	009D	161	157	161							
NOBOR	0171	302	293	302							
NOBR	01E1	383	357	383							
NOCLK	001C	61	61	78							
NOK	0038	62	49	57	62	80	99	131			
NOT=0	00FB	195	184	195							
ONE	0000	7	7	61	79	82					
OPTION	0000	OPC	4	5							
PAGE	0000	OPC	341	388	389	390	391	392	393	394	395
			107	133	176	238	257	304			
PLUS	0100	242	203	242							
RBIT	0034	OPC	61	181							
RESET	003D	32	29	32							
RETN	000F	OPC	27	278	300	315	328	340	380	420	
RE1	003B	33	33	35							
RSHFT	01BA	335	335								
RSTR	000C	OPC	33	68							
R1	01AB	337	337	339							
SAMAN	0027	OPC	292								
SBAA	017A	285	217	250	285						
SBIT	0030	OPC	82	158	188						
SCAN1	0039	40	40	74							
SCAN2	0006	66	38	66							
SETR	000D	OPC	72								
SPACE	0000	OPC	316	331	365	374	381	403	412	415	421
			180	185	194	210	246	262	281	286	301
			60	76	81	90	114	123	129	146	151
			6	10	22	31	36	39	50		
S1	0156	289	289	299							
TAM	0003	OPC	130	233	363	383					
TAMIY	0020	OPC	42	225	312	325					
TAMZA	0004	OPC	273	279	295	302					
TBIT1	0038	OPC	45	79	156	163					
TCMIY	0060	OPC	98	119	124	127	236	347	417	430	
TCY	0040	OPC	321	336	354	358	375	405	413	416	427
			161	169	182	187	221	231	263	287	308
			32	37	41	62	83	97	115	138	147
TDO	000A	OPC	70								
TKA	0008	OPC	23	44							
TMA	0021	OPC	66	223	310	323					
TMY	0022	OPC	63	355							
TOB	0103	244	244	253	256						
TRAB	0180	320	200	320							
TRAC	0180	308	308	345							
TRCA	00FE	222	222	227							
TR0	01BB	321	321								
TSTZ	0013	84	84	89							
TYA	0023	OPC	40	94							
T1	0181	309	309	314							
T2	0187	322	322	327							
XMA	002E	OPC	139	143	162	337	376				
YNEA	0002	OPC	95	189	406						
YNEC	0050	OPC	73	226	276	298	313	326	378	418	

OPCODE USAGE STATISTICS

OPCODE	COUNT
ALEC	18
AMAAC	3
A10AAC	1
A6AAC	2
A8AAC	1
BL	9
BR	60
CALL	4
CALLL	10
CLA	6
COMX	6
DAN	1
DMAN	4
DYN	9
END	1
EQU	3
FETCH	1
IA	2
IMAC	3
IYC	4
LDP	9
LDX	16
MNEZ	3
OPTION	2
PAGE	15
RBIT	2
RETN	8
RSTR	2
SAMAN	1
SBIT	3
SETR	1
SPACE	34
TAM	4
TAMIY	4
TAMZA	4
TBIT1	4
TCMIY	8
TCY	27
TDO	1
TKA	2
TMA	4
TMY	2
TYA	2
XMA	5
YNEA	3
YNEC	8

ROM CODE ASSEMBLY

```
000 90 0 SAMPLE   0080 002C 0080 0008 007E 00AC 0022 0038 009E 002C 00B8 00C0 008F 002C 0043 0018 0060
011 A1 0 SAMPLE   005F 007E 0030 0070 0093 00C1 0002 00B8 002C 0026 0021 0001 0020 002C 0014 00AC 0043
022 38 0 SAMPLE   00B8 009C 0046 00B9 0023 0010 000D 00A4 004B 0023 00B8 0070 000C 000C 0034 0086 0007
033 0C 0 SAMPLE   0080 002B 000A 0046 003A 0046 0080 0080 0008 000F 00BB 00BD 0070 0000 0000 0000 0047
044 63 0 SAMPLE   0028 0000 0000 0000 0000 0000 0000 0000 0079 0000 00B8 0000 0000 0000 0000 0000 0000
055 33 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00BC 0000 00BC 0000 0000 0000 0000 0000 0000 0000
066 FD 0 SAMPLE   0010 0000 0000 0000 0000 0000 0000 0000 002A 0000 0000 0003 0000 0000 0000 0000 0060
077 90 0 SAMPLE   0000 00BB 00BC 0000 0000 0060 0069 007A 0028 0000 0085 0000 0046 002E 001E 0000 0000
088 74 0 SAMPLE   0000 0000 0000 00FA 0079 0000 0033 0000 0000 0000 0000 0000 001F 0000 0039 0000 0006
099 40 0 SAMPLE   0000 0000 0000 009F 001F 0078 0000 0026 0000 0000 0000 0000 0000 009D 0000 0000 002E
0AA 33 0 SAMPLE   0000 0000 00B3 00AC 001C 0000 0000 003B 004A 0000 0000 0000 004B 0043 0014 0081 0000
0BB FF 0 SAMPLE   00F7 002F 0080 002C 002E 0000 00F7 00C0 0037 0043 001A 0020 0000 00A1 0000 0000 00FA
0CC 82 0 SAMPLE   002A 005E 0012 0000 0000 0003 0060 0000 0070 0000 001E 0000 0080 003E 0000 00AE 00BB
0DD F5 0 SAMPLE   0074 0070 0040 001F 003C 0000 009A 0000 0079 00BD 0000 0028 008F 0000 0000 00B6 0072
0EE 56 0 SAMPLE   001F 00B3 0021 0078 003E 0000 001F 0040 0080 007C 0016 0010 003E 0080 004B 0087 0002
0FF D6 0 SAMPLE   0032 0000 0000 0000 001A 00C0 0000 0000 0000 0000 0000 0000 0000 001F 0000 0000 0000
110 B9 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00B3 0000 0083 0000 0000
121 E7 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 00F7 0000 0000 0000
132 80 0 SAMPLE   0000 0000 0000 0000 001F 0000 0000 0083 0000 0000 001A 0000 0070 00FA 0000 0027 0000
143 39 0 SAMPLE   003C 0040 00B1 0004 0000 0000 0000 0000 0005 002F 0097 0087 0000 0000 0000 0000 0000
154 18 0 SAMPLE   0004 0000 0040 0000 0000 0000 0000 0000 0000 000F 0006 002B 0000 000F 0000 0000 0000
165 4F 0 SAMPLE   0000 005E 0000 0000 003C 0000 0000 000E 002F 008E 0000 0096 002B 0025 0000 0000 0000
176 63 0 SAMPLE   0079 00BC 000E 00B7 005E 0004 0025 00C4 0025 0000 0000 0000 0000 0040 003C 0000 0000
187 5A 0 SAMPLE   0000 0000 0000 0000 0000 0021 0000 00B7 0000 0000 0000 0000 0000 0000 0000 0046 0000
198 81 0 SAMPLE   002C 0000 0000 0000 003E 000F 0000 0000 000F 0000 0000 0000 0000 0000 005E 0000 0000
1A9 ED 0 SAMPLE   002F 0000 0000 0000 002E 0040 0000 0000 0020 00AB 0000 0000 0000 003F 00B3 0000 000F
1BA 6E 0 SAMPLE   0000 001F 0020 0021 0081 005E 0000 00BF 0000 0016 00C0 0003 0000 0000 0000 0000 0000
1CB B5 0 SAMPLE   001A 0066 0000 00B3 0000 0000 0000 0000 0000 00C0 0000 002E 0000 005E 0000 0000 0000
1DC 16 0 SAMPLE   003C 0003 00A1 0000 000F 0000 0000 00C0 0000 0000 001F 0000 0000 0040 0000 0000 00BF
1ED EF 0 SAMPLE   002B 002A 0000 0000 00A7 00B5 0000 00C0 0000 0022 002F 002A 004E 0000 003C 001F 004E
1FE CC 0 SAMPLE   003E 00F7 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
20F EF 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
220 DE 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
231 CD 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
242 BC 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
253 AB 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
264 9A 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
275 89 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
286 78 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
297 67 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
2A8 56 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
2B9 45 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
2CA 34 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
2DB 23 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
2EC 12 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
2FD 01 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
30E EF 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 00C0 0000 0000 0000 0000 0000 0000 0000 0000 0000
31F DE 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
330 CD 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
341 BC 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
352 AB 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
363 9A 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
374 89 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
385 78 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
396 67 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
3A7 56 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
3B8 D6 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 002F 0040 0000 0000 0000 0000
3C9 3C 0 SAMPLE   0000 0000 0000 0002 0000 003C 0000 0000 0000 0000 0000 0000 0000 002C 0000 008E 0000
3DA 60 0 SAMPLE   0000 0000 003F 0046 0040 0000 0080 0000 0000 0000 0000 0000 000F 0000 0000 006F 0000
3EB BE 0 SAMPLE   0000 0000 002C 00AF 0000 0000 00AF 0010 0000 0000 0000 004E 00AC 005E 003C 0000 0026
3FC 75 0 SAMPLE   00F7 0060 00F7 003E 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
40D EF 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
41E DE 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
42F CD 0 SAMPLE   0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
```

END OF ROM CODE ASSEMBLY

```
TMS1000 SIMULATOR  (VERSION C.1 6/16/75)    11/24/75    14:31:32
COPYRIGHT (C) 1975 TEXAS INSTRUMENTS INCORPORATED
TEXAS INSTRUMENTS OWNS AND IS RESPONSIBLE FOR SIM1000
ASSEMBLER INPUT: TMS1000 OBJECT: 'SAMPLE'CREATED 11/24/75 14:30:46 BY VERSION B(05/01/75) ASSM
   KEYBOARD
 KEY DOWN =  750 INSTRUCTION CYCLES; FETCH AT ROM LOCATION 0 06 (HEX)
 KEY      0     3 0 0
 KEY      1     3 1 0
 KEY      2     3 2 0
 KEY      3     3 3 0
 KEY      4     3 4 0
 KEY      5     3 5 0
 KEY      6     3 6 0
 KEY      7     2 1 0
 KEY      8     2 2 0
 KEY      9     2 3 0
 KEY      CLC   1 0 0
 KEY      C     1 1 0
 KEY      ENT   1 2 0
 KEY      +     1 3 0
 KEY      -     1 4 0
 KEY      *     1 5 0
 KEY      /     1 6 0
   SNAPSHOT
    MAP     0123456789ABCDE
    REGAD   0(6-0);;0(9);;;;0;
    REGB    0(F-0);
    REGC    1(F-0);
    REGD    2(F-0);
    REGE    3(F-0);
   OPLA
    OUT -0=00111111
    OUT -1=00000110
    OUT -2=01011011
    OUT -3=01001111
    OUT -4=01100110
    OUT -5=01101101
    OUT -6=01111101
    OUT -7=00000111
    OUT -8=01111111
    OUT -9=01101111
    OUTB 1-----=10000000
***END OF DATA ON FT09F001***
```

*** KEY-BOARD DEFINITION ***

	R00	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10
K1	CLC	C	ENT	+	-	*	/				
K2		7	8	9							
K4	0	1	2	3	4	5	6				
K8											

TEST

DATA CARD 1:1234567 ENT 999999 - 2 ENT 3 + 4 ENT 2 * C 6 ENT 3 / 3 ENT 6 / C 2 ENT 3 - 3 ENT

	.	000600000FFFFFFF	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000204
1	1.	008600000FFFFFF1	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000808
2	12.	008600000FFFFF12	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000808
3	123.	008600000FFFF123	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000806
4	1234.	008600000FFF1234	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000804
5	12345.	008600000FF12345	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000809
6	123456.	008600000F123456	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000804
7	123456.	008600000F123456	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000809
ENT	123456.	000600000F123456	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000807
9	9.	008600000FFFFFF9	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000809
9	99.	008600000FFFFF99	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000803
9	999.	008600000FFFF999	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000803
9	9999.	008600000FFF9999	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000803
9	99999.	008600000FF99999	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000803
9	999999.	008600000F999999	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000803
-	876543.	000600000F876543	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000806
2	2.	008600000FFFFFF2	6007400700000000	8001C20760347006	6300B22B10123456	IC= 00000807
ENT	2.	000600000FFFFFF2	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000807
3	3.	008600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000803
+	5.	000600000FFFFFF5	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000804
4	4.	008600000FFFFFF4	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000808
ENT	4.	000600000FFFFFF4	6007400700000000	8001C20760347006	6300B22B10000004	IC= 00000807
2	2.	008600000FFFFFF2	6007400700000000	8001C20760347006	6300B22B10000004	IC= 00000805
*	8.	000600000FFFFFF8	6007400700000000	8001C20760347006	6300B22B10000004	IC= 00001172
C	.	000600000FFFFFFF	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000830
6	6.	008600000FFFFFF6	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000851
ENT	6.	000600000FFFFFF6	6007400700000000	8001C20760347006	6300B22B10000006	IC= 00000807
3	3.	008600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000006	IC= 00000803
/	.	000600000FFFFFFF	6007400700000000	8001C20760347006	6300B22B10000006	IC= 00000952
3	3.	008600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000006	IC= 00000804
ENT	3.	000600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000003	IC= 00000807
6	6.	008600000FFFFFF6	6007400700000000	8001C20760347006	6300B22B10000003	IC= 00000850
/	2.	000600000FFFFFF2	6007400700000002	8001C20760347006	6300B22B10000003	IC= 00001116
C	.	000600000FFFFFFF	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000830
2	2.	008600000FFFFFF2	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000806
ENT	2.	000600000FFFFFF2	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000807
3	3.	008600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000803
-	1.	000600000FFFFFF1	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000806
3	3.	008600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000002	IC= 00000803
ENT	3.	000600000FFFFFF3	6007400700000000	8001C20760347006	6300B22B10000003	IC= 00000807

DATA CARD 2:2 - C

2	2.	008600000FFFFFF2	6007400700000000	8001C20760347006	6300B22B10000003	IC= 00000805
-	.	000600000FFFFFFF	6007400700000000	8001C20760347006	6300B22B10000003	IC= 00000808
C	.	000600000FFFFFFF	6007400700000000	8001C20760347006	6300B22B10000000	IC= 00000831

```
DATA CARD  3:$ 3 $ CLC PRT
 3                      3.     008600000FFFFFF3      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000804
 CLC              0000003.     0026000000000003      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00000802
PA=0 PC=06 IR=21 X=0 Y=5 A=0 S=1 SL=1 CL=0 PB=0 SR=06 KL=00
R(0-A)=00000010000      0(7-0)=10111111   KEY=
M0-3(F-0)=0026000000000003 6007400700000000 8001C20760347006 6300B22B10000000

DATA CARD  4:PAT K8 139 F0   RUN 1112
 RUN              0000003.     0120000000000003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD  5:RUN 1112
 RUN              0000003.     0220000000000003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD  6:RUN 1112
 RUN              0000003.     0321000000C00003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD  7:RUN 1112
 RUN              0000003.     0422000000000003      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD  8:RUN 1112
 RUN              0000003.     0522000000000003      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD  9:SET M 0E 85 PRT
PA=0 PC=3A IR=A4 X=0 Y=D A=0 S=1 SL=1 CL=0 PB=0 SR=3A KL=08
R(0-A)=00100000000      0(7-0)=10111111   KEY=
M0-3(F-0)=5822000000000003 6007400700000000 8001C20760347006 6300B22B10000000

DATA CARD 10:RUN 1112
 RUN              0000003.     5923000000000003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD 11:SET M 00 1

DATA CARD 12:RUN 1112                  RUN *
 RUN              0000000.     0024000000000000      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112
 RUN              0000000.     0003000000000000      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000074

DATA CARD 13:SET K 00                   PRT
PA=0 PC=06 IR=21 X=0 Y=2 A=F S=1 SL=0 CL=0 PB=0 SR=06 KL=08
R(0-A)=00010000000      0(7-0)=10111111   KEY=
M0-3(F-0)=0003000000000000 6007400700000000 8001C20760347006 6300B22B10000000

DATA CARD 14:PAT K8 0 0         PRT
PA=0 PC=06 IR=21 X=0 Y=2 A=F S=1 SL=0 CL=0 PB=0 SR=06 KL=00
R(0-A)=00010000000      0(7-0)=10111111   KEY=
M0-3(F-0)=0003000000000000 6007400700000000 8001C20760347006 6300B22B10000000

DATA CARD 15:C 3 $ CLC $ PRT
 C                       .     000600000FFFFFFF      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000804
 3                      3.     008600000FFFFFF3      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000804
 CLC              0000003.     0026000000000003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000802
PA=0 PC=06 IR=21 X=0 Y=5 A=0 S=1 SL=1 CL=0 PB=0 SR=06 KL=00
R(0-A)=00000010000      0(7-0)=10111111   KEY=
M0-3(F-0)=0026000000000003 6007400700000000 8001C20760347006 6300B22B10000000

DATA CARD 16:CLK K8 555 555  RUN 1112 $  RUN 1112 $ SET M 0E 85   RUN 1112  SET M 00 1
 RUN              0000003.     0120000000000003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112
 RUN              0000003.     0220000000000003      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00001112
 RUN              0000003.     5921000000000003      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00001112

DATA CARD 17:RUN 1112  RUN *   CLK K8 0 0   4 CLC PAT K8 555 A    RUN 2224
 RUN              0000000.     0022000000000000      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00001112
 RUN              0000000.     0000000000000000      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000092
 4                      4.     008600000FFFFFF4      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000808
 CLC              0000004.     0026000000000004      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00000802
 RUN              0000004.     0220000000000004      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00002224

DATA CARD 18:RUN 4448    RUN 8896
 RUN              0000004.     0623000000000004      60C7400700000000      8001C20760347006      6300B22B10000000   IC= 00004448
 RUN              0000004.     1433000000000004      6007400700000000      8001C20760347006      6300B22B10000000   IC= 00008896

END OF FILE ON FT05F001
```

APPENDIX A
TMS 1000/1200 AND TMS 1100/1300 ELECTRICAL SPECIFICATIONS

A1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

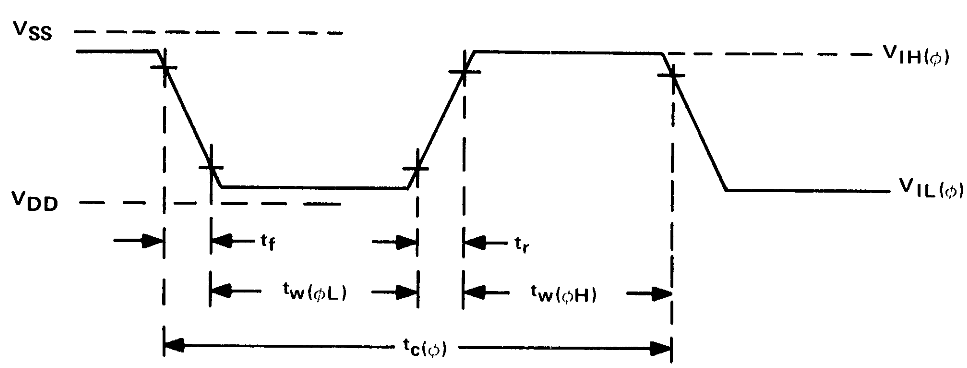
Voltage applied to any device terminal (see Note 1)		−20 V
Supply voltage, V_{DD}		−20 V to 0.3 V
Data input voltage		−20 V to 0.3 V
Clock input voltage		−20 V to 0.3 V
Average output current (see Note 2):	O outputs	−24 mA
	R outputs	−14 mA
Peak output current:	O outputs	−48 mA
	R outputs	−28 mA
Continuous power dissipation:	TMS 1000/1100 NL	400 mW
	TMS 1200/1300 NL	600 mW
Operating free-air temperature range		0°C to 70°C
Storage temperature range		−55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

A2 RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD} (see Note 3)		−14	−15	−17.5	V
High-level input voltage, V_{IH} (see Note 4)	K	−1.3	−1	0.3	V
	INIT or Clock	−1.3	−1	0.3	
Low-level input voltage, V_{IL} (see Note 4)	K	V_{DD}		−4	V
	INIT or Clock	V_{DD}	−15	−8	
Clock cycle time, $t_{c(\phi)}$		2.5	3	10	μs
Instruction cycle time, t_c		15		60	μs
Pulse width, clock high, $t_{w(\phi H)}$		1			μs
Pulse width, clock low, $t_{w(\phi L)}$		1			μs
Sum of rise time and pulse width, clock high, $t_r + t_{w(\phi H)}$		1.25			μs
Sum of fall time and pulse width, clock low, $t_f + t_{w(\phi L)}$		1.25			μs
Oscillator frequency, f_{osc}		100		400	kHz
Operating free-air temperature, T_A		0		70	°C

NOTES:
1. Unless otherwise noted, all voltages are with respect to V_{SS}.
2. These average values apply for any 100-ms period.
3. Ripple must not exceed 0.2 volts peak-to-peak in the operating frequency range.
4. The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.



NOTE: Timing points are 90% (high) and 10% (low).

EXTERNALLY DRIVEN CLOCK INPUT WAVEFORM

A3 ELECTRICAL CHARACTERISTICS OVER RECOMMENDED OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_I	Input current, K inputs		$V_I = 0$ V		50	300	500	μA
V_{OH}	High-level output voltage (see Note 1)	O outputs	$I_O = -10$ mA		−1.1‡	−0.6‡		V
		R outputs	$I_O = -2$ mA		−0.75	−0.4		V
I_{OL}	Low-level output current		$V_{OL} = V_{DD}$				−100	μA
$I_{DD(av)}$	Average supply current from V_{DD} TMS 1000/1200 (see Note 2)		All outputs open			−6	−10	mA
$I_{DD(av)}$	Average power dissipation TMS1100/1300 (see Note 2)		All outputs open			−7	−11	mA
$P_{(AV)}$	Average power dissipation TMS 1000/1200 (see Note 2		All outputs open			90	175	mW
$P_{(AV)}$	Average power dissipation TMS1100/1300 (see Note 2)		All outputs open			105	193	mW
f_{osc}	Internal oscillator frequency		$R_{ext} = 50$ kΩ,	$C_{ext} = 47$ pF	250	300	350	kHz
C_i	Small-signal input capacitance, K inputs		$V_I = 0$,	f = 1 kHz		10		pF
$C_{i(\phi)}$	Input capacitance, clock input		$V_I = 0$,	f = 100 kHz		25		pF

†All typical values are at $V_{DD} = -15$ V, $T_A = 25°C$.

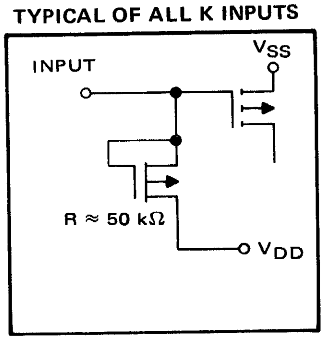
‡Parts with V_{OH} of −2 V minimum, −1.3 V typical, are available if requested.

NOTES: 1. The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.

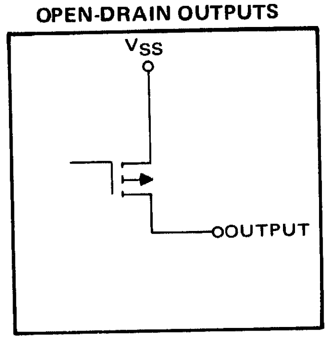
2. Values are given for the open-drain O and R output configurations. Pull-down resistors are optionally available on all outputs and increase I_{DD} (see Section A4).

A4 SCHEMATICS OF INPUTS AND OUTPUTS

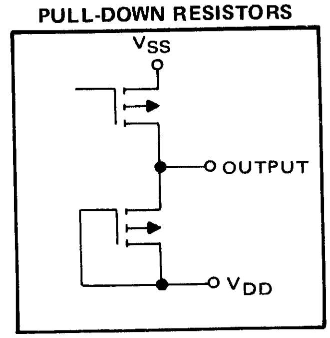
TYPICAL OF ALL K INPUTS



TYPICAL OF ALL O AND R OPEN-DRAIN OUTPUTS



TYPICAL OF ALL O AND R OUTPUTS WITH OPTIONAL PULL-DOWN RESISTORS



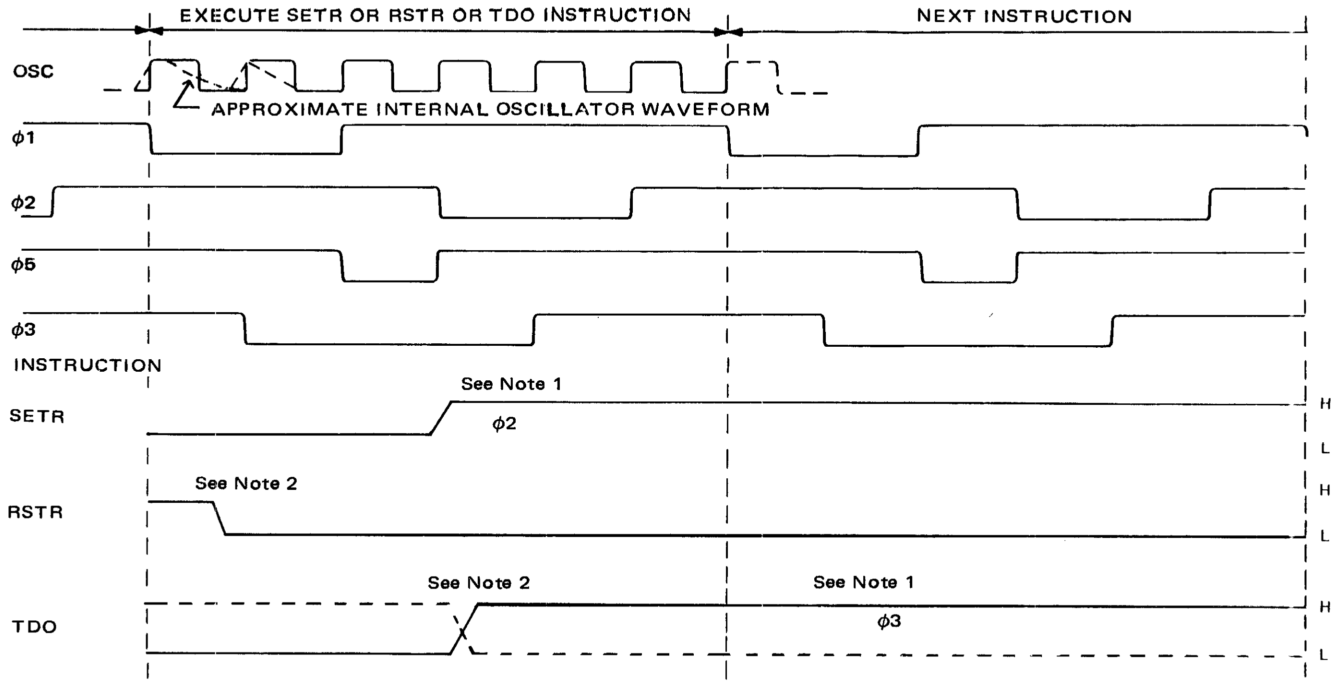
The O outputs have nominally 60 Ω on-state impedance; however, upon request a 130-Ω buffer can be mask programmed (see note [‡] section A3).

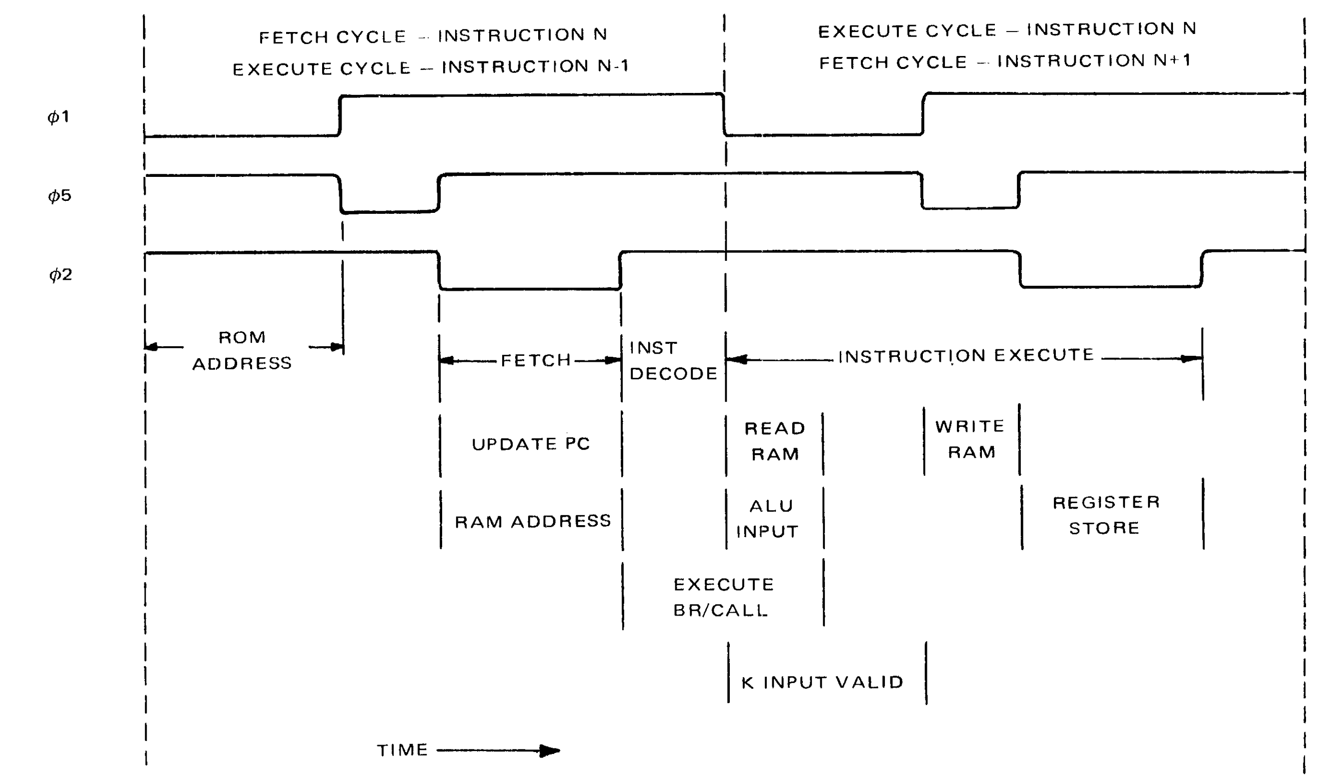
The value of the pull-down resistors is mask alterable and provides the following nominal short-circuit output currents (outputs shorted to V_{SS}):

O outputs: 100, 200, 300, 500, or 900 μA

R outputs: 100, 150, or 200 μA

A5 OUTPUT, INPUT, AND INSTRUCTION TIMING





NOTES: 1. Initial rise time is load dependent. The high-level output voltage, V_{OH}, is characterized following the indicated clock period.

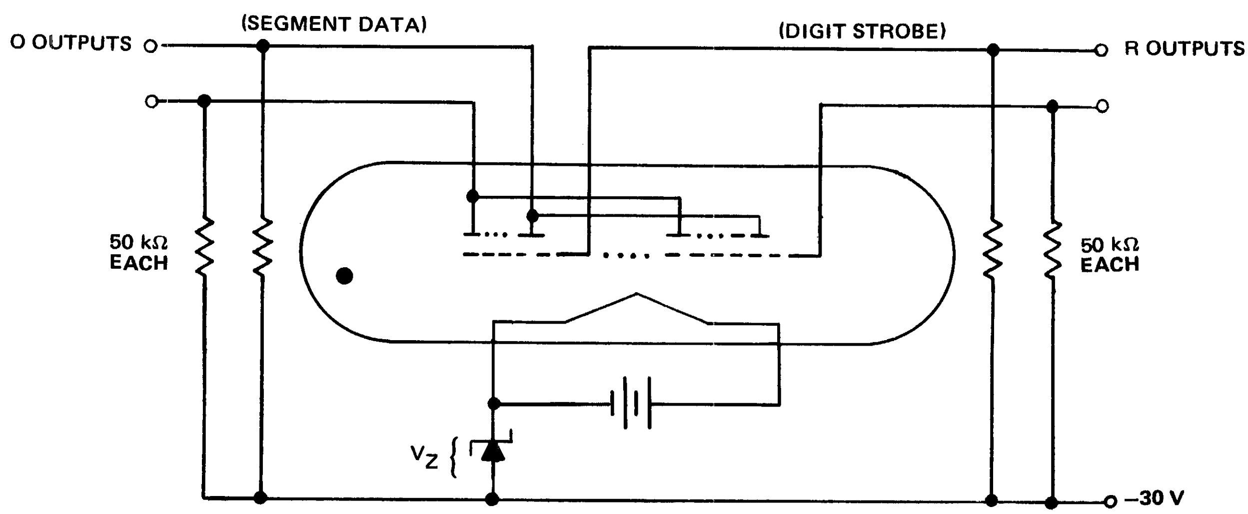
2. Rise and fall times are load dependent.

APPENDIX B
TMS 1070 AND TMS 1270 MICROCOMPUTERS

B1 INTRODUCTION

The TMS 1000 series flexibility is augmented by two versions of high-voltage (35-volt) microcomputers, the TMS 1070 and the TMS 1270. The standard instruction set and operation is identical to that of the TMS 1000/1200. Architecturally, the devices are identical to the TMS 1000/1200 except that two additional O-output OR-matrix terms were added to provide a total of ten O outputs in the TMS 1270, a 40-pin package unit. The TMS 1070/1270 provides direct interface to low-voltage flourescent displays. The TMS 1070/1270 interfaces with all circuits requiring up to 35-volt levels.

The accompanying diagram shows an interface to a 30-volt fluorescent display.



STROBED FLUORESCENT DISPLAY INTERCONNECT

B2 DESIGN SUPPORT

The TMS 1070/1270 simulation is provided by several time-sharing services. The assembler and simulator programs are accessed by specifying the appropriate device option in the assembler TITLE command.

Functional hardware simulation is accomplished by an SE-1 or an HE-2. To emulate more than eight O outputs in the TMS 1270 with an HE-2 requires an external decoder. Level-shifting buffers allow functional evaluation in the high-voltage prototyping systems.

B3 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)*

Voltage applied to any device terminal (see Note 1) . . . . –20 V
Supply voltage, V_{DD} . . . . –20 V to 0.3 V
Data input and output voltage with V_{DD} applied (see Note 2) . . . . –35 V to 0.3 V
Clock input and INIT input voltage . . . . –20 V to 0.3 V
Average output current (see Note 3): O outputs . . . . –2.5 mA
R outputs . . . . –12 mA
Peak output current: O outputs . . . . –5 mA
R outputs . . . . –24 mA
Continuous power dissipation: TMS 1070 NL . . . . 400 mW
TMS 1270 NL . . . . 600 mW
Operating free-air temperature range . . . . 0°C to 70°C
Storage temperature range . . . . –55°C to 150°C

*Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

B4 RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD} (see Note 4)		–14	–15	–17.5	V
High-level input voltage, V_{IH} (see Note 5)	K	–6		0.3	V
	INIT or Clock	–1.3	–1	0.3	
Low-level input voltage, V_{IL} (see Note 5)	K (See Note 2)	–35		–8	V
	INIT or Clock	V_{DD}	–15	–8	
Clock cycle time, $t_{c(\phi)}$		2.5	3	10	µs
Instruction cycle time, t_c		15		60	µs
Pulse width, clock high, $t_{w(\phi H)}$		1			µs
Pulse width, clock low, $t_{w(\phi L)}$		1			µs
Sum of rise time and pulse width, clock high, $t_r + t_{w(\phi H)}$		1.25			µs
Sum of fall time and pulse width, clock low, $t_f + t_{w(\phi L)}$		1.25			µs
Oscillator frequency, f_{osc}		100		400	kHz
Operating free-air temperature, T_A		0		70	°C

NOTES:
1. Unless otherwise noted, all voltages are with respect to V_{SS}.
2. V_{DD} must be within the recommended operating conditions specified in B4.
3. These average values apply for any 100-ms period.
4. Ripple must not exceed 0.2 volts peak-to-peak in the operating frequency range.
5. The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.

B5 ELECTRICAL CHARACTERISTICS OVER RECOMMENDED OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
I_I	Input current, K inputs		$V_I = 0$ V		40	100	300	μA
V_{OH}	High-level output voltage (see Note 1)	O outputs	$I_O = -1$ mA		−1	−0.5		V
		R outputs	$I_O = -10$ mA		−4.5	−2.25		V
I_{OL}	Low-level output current		$V_{OL} = V_{DD}$				−100	μA
$I_{DD(av)}$	Average supply current from V_{DD}		All outputs open			−6	−10	mA
$P_{(AV)}$	Average power dissipation		All outputs open			90	175	mW
f_{osc}	Internal oscillator frequency		$R_{ext} = 50$ kΩ,	$C_{ext} = 47$ pF	250	300	350	kHz
C_i	Small-signal input capacitance, K inputs		$V_I = 0$ V,	f = 1 kHz		10		pF
$C_{i(\phi)}$	Input capacitance, clock input		$V_I = 0$ V,	f = 100 kHz		25		pF

†All typical values are at $V_{DD} = -15$ V, $T_A = 25°C$.

NOTE 1: The algebraic convention where the most-positive (least-negative) limit is designated as maximum is used in this specification for logic voltage levels only.