

Microprocessor Course

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, such information does not convey to the purchaser of the product described any license under the patent rights of Motorola, Inc. or others.

Motorola reserves the right to change specifications without notice.

EXORciser, EXORdisk, EXORtape and EXORterm are trademark of Motorola Inc.

Motorola Inc. 1979
"All Rights Reserved"

EUROPEAN MOTOROLA SEMICONDUCTOR SALES OFFICES

DENMARK Motorola A/S Brædbovej 23 DK-2800 Lyngby Tel. (01) 88 44 55	Virnsbergerstrasse 43 8500 Nürnberg Tel. (0911) 6 57 61 Stralsunder Strasse 1 7032 Sindelfingen Tel. (07031) 8 30 74/8 30 75 Abraham-Lincoln-Strasse 28 6200 Wiesbaden Tel. (06121) 76 19 21	Sales Office Via Portanova 10 40123 Bologna — Tel. 26 69 05 Sales Office Via Costantino Maes 68 00162 Roma — Tel. 831 47 46	UNITED KINGDOM Motorola Ltd. Headquarter York House, Empire Way Wembley Middlesex Tel. (01) 902 88 36 Sales Office 10-12, Mount Street, Television House Manchester M2 5WS, Lancs Tel. (61) 833 07 31/833 07 34 Sales Office Colvilles Road, Kelvin Estate East Kilbride, Scotland Tel. (3552) 3 91 01
FRANCE Motorola Semiconducteurs S.A. Headquarter 15-17, avenue de Ségur 75007 Paris Tel. 561 50 61 Sales Office 42, avenue de La Plaine-Flourie 38240 Meylan (Grenoble) Tel. (76) 90 22 61	HOLLAND Motorola B.V. Emmalaan 41 Utrecht Tel. (030) 51 02 07	NORWAY Motorola A/B (Service Office) Brugt 1 Oslo 1 — Tel. (02) 41 91 40	HEADQUARTERS EUROPEAN OPERATIONS SWITZERLAND Motorola Inc. Semiconductor Products Group 15, chemin de la Voie-Creuse, P.O. Box 8 1211 Genève 20 Tel. (022) 33 56 07
WEST GERMANY Motorola GmbH, Geschäftsbereich Halbleiter Headquarter Münchner Strasse 18 8043 Unterföhring Tel. (089) 92 481 Sales Offices Hans-Böckler-Strasse 30 3012 Langenhagen — Hannover Tel. (0511) 77 20 37	ITALY Motorola S.p.A. Headquarter Via Ciro Menotti 11 20125 Milano Tel. 738 61 41/2/3	SWEDEN Motorola AB. Virebergsvägen 19 17140 Solna — Tel. (08) 82 02 95	
		SOUTH AFRICA Motorola South Africa (Pty) Ltd. P.O. Box 39586 Braamvlei 2019 Tel. 786 11 84	
		SWITZERLAND Motorola Semiconductor Products S.A. Alte Landstrasse 101 8702 Zollikon — Tel. (01) 65 56 56	

FRANCHISED MOTOROLA SEMICONDUCTOR DISTRIBUTORS

AUSTRIA Elbatex GmbH Endraserstrasse 54 — 1238 Wien Tel. (222) 88 55 11	EBV Elektronik Vertriebs-GmbH Mylustrasse 54 6000 Frankfurt 1 Tel. (0611) 72 04 16 EBV Elektronik Vertriebs-GmbH Alexanderstrasse 69 7000 Stuttgart 1 Tel. (0711) 24 74 81 Jermyn GmbH Postfach 1130 6277 Camberg Tel. (06434) 60 05 Mütron Müller & Co. KG Bernerstrasse 22 2800 Bremen Tel. (0421) 31 04 85 RTG, E. Springorum GmbH + Co. (Main Office) Postfach 426 46 Dortmund 2 Tel. (0231) 5 49 51 RTG, E. Springorum GmbH + Co. Friedrich-Ebert Damm 112 2000 Hamburg 70 Tel. (040) 693 70 61/62 RTG, E. Springorum GmbH + Co. Ungerstrasse 43 8000 München 40 Tel. (089) 36 65 00 RTG, E. Springorum GmbH + Co. Reutlingerstrasse 87 7000 Stuttgart-Degerloch Tel. (0711) 76 64 28 RTG, E. Springorum GmbH + Co. Mendelssohn-Bartholdy-Strasse 16 6200 Wiesbaden Tel. (06121) 52 73 09 SASCO Vertrieb von elektronischen Bauelementen GmbH (Main Office) Hermann-Oberth-Strasse 16 8001 Putzbrunn b. München Tel. (089) 46 40 61/69 SASCO Vertrieb von elektronischen Bauelementen GmbH Postfach 3066 4005 Düsseldorf/Meerbusch 3 Tel. (02150) 14 33 SASCO Vertrieb von elektronischen Bauelementen GmbH Postfach 890214 3000 Hannover Tel. (0511) 96 25 98 SASCO Vertrieb von elektronischen Bauelementen GmbH Lorenzer Strasse 15 8500 Nürnberg Tel. (0911) 20 41 52 SASCO Vertrieb von elektronischen Bauelementen GmbH Staffenbergerstrasse 24 — 7000 Stuttgart 1 Tel. (0711) 24 45 21 SPOERLE Electronic (Ab 1.1.1979) Otto-Hahn-Strasse 13 — 6072 Dreieich b. Frankfurt Tel. (06103) * 04-1 Technoprojekt (Main Office) Heinrich-Ebner-Strasse 13 7000 Stuttgart — Bad Cannstatt Tel. (0711) 56 17 12 Technoprojekt Osting 151 — 6231 Schwalbach/Ts Tel. (06196) 8 21 00	HOLLAND E.V. Diode Hollantlaan 22 — Utrecht Tel. (030) 88 42 14 Manudax Nederland B.V. Meeustraat 7 5473 ZG Heeswijk (N.B.) — P.O. Box 25 Tel. (41) 39 12 52	SOUTH AFRICA L'Electron 704 Main Pretoria Road, Wynberg Tul. P.O. Box 10544, Johannesburg 2000 Tel. 40 62 96
BELGIUM Diede Belgium Rue Picard 202-204 — 1020 Bruxelles Tel. (02) 428 51 05			SPAIN Hispano Electronica S.A. (Main Office) 12233 Enskede — Tel. (08) 49 25 05 AB Gösta Bäckström Alströmergatan 22 — Box 12009 10221 Stockholm Tel. (08) 54 10 80
DENMARK Distributøreren Interelco Aps Hovedgaden 16 — 4622 Havdrup Tel. (03) 38 57 16			SWEDEN Interelco AB. Sandsborgsvägen 50 12233 Enskede — Tel. (08) 49 25 05 AB Gösta Bäckström Alströmergatan 22 — Box 12009 10221 Stockholm Tel. (08) 54 10 80
FINLAND Field Oy Veneentekijantie 18 — 00210 Helsinki 21 Tel. (80) 692 25 77			SWITZERLAND Elbatex AG Alb, Zwyssig-Strasse 28 — 5430 Wettingen Tel. (056) 26 56 41 Omniv Ray AG Dufourstrasse 56 — 8008 Zürich Tel. (01) 34 07 66
FRANCE Ballion Electronique Zone Industrielle de Kerscao/Brest 29219 Le Relecq-Kerhuon — B.P. 16 Tel. (98) 28 03 03 Caldis S.A. 53, rue Charles Frérot — 94250 Gentilly Tel. (01) 581 90 20 Ets. F. Fautrier S.A. (Main Office) Rue des Trois-Glorieuses 42270 St-Priest-en-Jarez (St-Etienne) Tel. (77) 74 67 33 Ets. F. Fautrier S.A. Avenue Laplace — Zone industrielle 13470 Carnoux Tel. (42) 82 16 41 Fautrier Ile de France 29, rue Ladru-Rollin — 92150 Suresnes Tel. (01) 772 46 46 Ets. Gros S.A. (Main Office) 13, rue Victor-Hugo — BP 63 59360 Saint-André-lez-Lille Tel. (20) 51 21 33 Ets. Gros S.A. 14, avenue du Général-Leclerc — 54000 Nancy Tel. (83) 35 17 35 Ets. Gros S.A. 5, rue Pascal — 94800 Villejuif Tel. (01) 678 27 27 S.C.A.I.B. S.A. 80, rue d'Arcueil — Zone Silic 94150 Rungis Tel. (01) 687 23 13 Sis Commercial Toutélectrique (Main Office) 15-17, Boulevard Bonrepos — 31008 Toulouse Tel. (61) 62 11 33 Sis Commercial Toutélectrique 80-83, quai des Quatrièmes — 33100 Bordeaux Tel. (56) 86 50 31			TURKEY ERAK Elektronik Sanyii Ve Ticaret A.S. Ankara Intibati Bürosu Kazi Mustafa Kemal Bul. 12 GAT14D79 Yenisehir/Ankara — Tel. 25 49 33
GERMANY Allfred Neye Eneatechnik GmbH (Ab 1.1.1979) Schillerstrasse 14 — D-2085 Quickborn/Hamburg Tel. (04) 106 61 21 Distron oHG Behmstrasse 3 1000 Berlin 10 Tel. (030) 342 10 41/45 EBV Elektronik Vertriebs-GmbH (Main Office) Gabriel-Max-Strasse 72 8000 München 90 Tel. (089) 64 40 55 EBV Elektronik Vertriebs-GmbH In der Meinenorth 9 3006 Burgbebel 1 / Hannover Tel. (051) 39 45 70 EBV Elektronik Vertriebs-GmbH Oststrasse 129 4000 Düsseldorf Tel. (0211) 8 46 46			UNITED KINGDOM A.M. Lock & Co. Ltd Neville Street, Middleton Road Oldham, Lancs OL9 6LF Tel. (061) 652 04 31 Caldis Electronics Ltd 37-39 Loverock Road Reading, Berks, RG3, 1ED Tel. (0734) 585 171 Cramer Components Ltd Hawke House Green Street Sunbury on Thames, Middlesex, England Tel. (0271) 8 55 77 Crellon Electronics Ltd 380, Bath Road Slough, Berks SL1 6JE Tel. (06286) 6 36 11 ITT Electronic Services Edinburgh Way Harlow, Essex CM20 2DF Tel. Harlow (0279) 26 777 Jermyn Industries Versty Estate — Sevenoaks, Kent Tel. (732) 51 17 74 Macro-Marketing Ltd. 396, Bath Road Slough, Berks SL1 6JD Tel. (06286) 630 11
			YUGOSLAVIA Elektrotehna Ljubljana Export-Import Titova 51 — P.O. Box 34-1 61000 Ljubljana Tel. (61) 32 02 41 Elektrotehna Ljubljana Filiale Beograd Marsala Tita 6/1 11000 Beograd Tel. (011) 69 69 24

EUROPEAN SEMICONDUCTOR FACTORIES

FRANCE Motorola Semiconducteurs S.A. Cimex-Laussetto Le Mirail 31023 Toulouse CEDEX Tel. (61) 40 11 88	GERMANY Motorola GmbH Münchner Strasse 18 8043 Unterföhring Tel. (089) 92 481	UNITED KINGDOM Motorola Semiconductors Ltd. Colvilles Road, Kelvin Estate East Kilbride/Glasgow (Scotland) Tel. (3552) 39 101
---	--	--

Table of contents

	Page
1. Number Systems	5
2. Microprocessing Unit	17
3. Memory	39
4. Peripheral Interface Adapter	45
5. ACIA	67
6. Addressing Modes	83
7. Assembler Techniques	93
8. Instruction Set	139
9. Program Problems	151
10. Example Programs	171
11. System Configuration	211


Number Systems

NUMBER SYSTEM

The Motorola 6800 Microprocessor is an 8-bit system. It has 8 data lines, 16 address lines, and functions with 8- and 16-bit registers. It is, therefore, convenient to use the Hexadecimal Number System when interfacing with the M6800 system. However, before concentrating on the Hexadecimal Number System, a discussion of several other number systems would be beneficial.

The most familiar number system is Base 10 or Decimal—i.e., 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9—since this is the system in general use. What does a typical base-10 number represent? Take, for example, 2743. The number 2743_{10} really represents $3 \times 10^0 + 4 \times 10^1 + 7 \times 10^2 + 2 \times 10^3$; and, as can be seen, the least significant digit (LSD) is 3 and the most significant digit (MSD) is 2.

In digital computers, numbers are represented in base 2 or binary form, i.e., 1's and 0's. One method of converting base 10 numbers to binary numbers is known as "repeated division by 2". Using 47_{10} for example:

$\begin{array}{r} 23 \\ 2 \overline{)47} \\ \underline{11} \\ 23 \\ \underline{5} \\ 211 \\ \underline{2} \\ 25 \\ \underline{1} \\ 22 \\ \underline{0} \\ 21 \end{array}$	R = 1 LSBit R = 1 R = 1 R = 1 R = 0 R = 1 MSBit		101111_2
--	--	---	------------

Converting 101111_2 back to a base-10 number, we have:

$$\begin{aligned}
 101111_2 &= 1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 + 0 \times 2^4 + 1 \times 2^5 \\
 &= 1 \times 1 + 1 \times 2 + 1 \times 4 + 1 \times 8 + 0 \times 16 + 1 \times 32 \\
 &= 1 + 2 + 4 + 8 + 0 + 32 \\
 &= 47_{10}
 \end{aligned}$$

In general, converting from a number in any base to a number in base 10 is accomplished as follows:

$$(A_0 B^0 + A_1 B^1 + A_2 B^2 + A_3 B^3 + A_4 B^4 \dots + A_n B^n)$$

where B is the base of the number system and A is the particular digit in the original number corresponding to its position to the left of the decimal point. On the example just completed, $(101111)_2$.

$$A_0 = 1, A_1 = 1, A_2 = 1, A_3 = 1, A_4 = 0, \text{ and } A_5 = 1 \text{ and } B = 2 \text{ (base 2).}$$

N-2 Number System

Another number system used with digital computers is octal, or base 8, since octal is a more convenient way of representing binary 2. To illustrate, the conversion of 61 in base 10 to a number in base 8 and a number in base 2 using the method of repeated division is shown below:

$\begin{array}{r} 7 \\ 8 \overline{)61} \\ \underline{0} \\ 8 \overline{)7} \end{array}$	R = 5 LSD R = 7 MSD	\uparrow	75_8 (Octal)
$\begin{array}{r} 30 \\ 2 \overline{)61} \\ \underline{15} \\ 2 \overline{)30} \\ \underline{7} \\ 2 \overline{)15} \\ \underline{3} \\ 2 \overline{)7} \\ \underline{1} \\ 2 \overline{)3} \\ \underline{0} \\ 2 \overline{)1} \end{array}$	R = 1 LSB R = 0 R = 1 R = 1 R = 1 R = 1 MSB	\uparrow	111101_2 (Binary)

As a proof that $75_8 = 111101_2$, convert each digit of 75 base 8 to base 2 by continuous division.

Convert 7_8 to base 2:

$\begin{array}{r} 3 \\ 2 \overline{)7} \\ \underline{1} \\ 2 \overline{)3} \\ \underline{0} \\ 2 \overline{)1} \end{array}$	R = 1 R = 1 R = 1	\uparrow	111_2
---	-------------------------	------------	---------

Convert 5_8 to base 2:

$\begin{array}{r} 2 \\ 2 \overline{)5} \\ \underline{1} \\ 2 \overline{)2} \\ \underline{0} \\ 2 \overline{)1} \end{array}$	R = 1 R = 0 R = 1	\uparrow	101_2
---	-------------------------	------------	---------

This demonstrates that an octal number (base 8) can be used to easily represent a string of binary bits (base 2). Therefore, $75_8 = 111101_2$, which is the same pattern of 1s and 0s as derived by converting from base 10 to base 2.

As previously mentioned, the M6800 Microprocessor utilizes 8- and 16-bit registers. But when trying to use the octal number system, there is a slight problem. Since in octal each digit represents 3 binary bits, and 8 and 16 bits cannot be selected evenly into groups of 3. This is resolved with the hexadecimal number system. Hexadecimal is a base-16 number system and can be handled in exactly the same manner

as base 8 or base 2. In Hexadecimal, four bits (in binary) represent one Hexadecimal number. Thus, an 8-bit register can be represented by a 2-digit hex number. To illustrate, assume there exists the binary number 01100111 in an 8-bit register. If this bit pattern is divided into two 4-bit groups of 0110 and 0111, then the hex representation would be 67₁₆. The following is offered as a proof:

$$\begin{array}{r}
 \uparrow \quad \quad \uparrow \\
 1100111_2 = \underline{1 \times 2^0} + \underline{1 \times 2^1} + \underline{1 \times 2^2} + \underline{0 \times 2^3} + \underline{0 \times 2^4} + \underline{1 \times 2^5} + \underline{1 \times 2^6} + \underline{0 \times 2^7} \\
 \text{MSB} \quad \quad \text{LSB} = 1 + 2 + 4 + 0 + 0 + 32 + 64 + 0 \\
 \\
 = 103_{10}
 \end{array}$$

and

$$\begin{array}{r}
 \text{LS Digit (LS half-byte)} \\
 \swarrow \quad \searrow \\
 67_{16} = 7 \times 16^0 + 6 \times 16^1 \\
 \swarrow \quad \searrow \\
 \text{MS Digit (MS half-byte)} \\
 \\
 = 7 \times 1 + 6 \times 16 \\
 = 7 + 96 \\
 = 103_{10}
 \end{array}$$

Therefore, 67₁₆ = 01100111₂ = 103₁₀

From this simple example, one might wonder how hexadecimal digits (base 16) are represented for numbers above 9. The following table shows the solution to this dilemma.

Base 10 (Decimal)	Base 16 (Hexadecimal)
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Table I

N-4 Number System

To convert any decimal (base 10) number to hexadecimal (base 16), again use the repeated division method. As an example, convert 1023_{10} to hexadecimal.

$\begin{array}{r} 63 \\ 16 \overline{)1023} \end{array}$	$R = 15_{10} = F$ * LSD	
$\begin{array}{r} 3 \\ 16 \overline{)63} \end{array}$	$R = 15_{10} = F$	\uparrow $3FF_{16}$
$\begin{array}{r} 0 \\ 16 \overline{)3} \end{array}$	$R = 3$ MSD	

therefore, $1023_{10} = 3FF_{16}$. As a check, convert $3FF_{16}$ back to a base 10 number.

$$\begin{aligned}
 3FF_{16} &= 15 \times 16^0 + 15 \times 16^1 + 3 \times 16^2 \\
 &= 15 \times 1 + 15 \times 16 + 3 \times 256 \\
 &= 15 + 240 + 768 \\
 &= 1023_{10}
 \end{aligned}$$

To further elaborate on the relationship between hexadecimal and binary, convert 1023_{10} to binary and then back to hexadecimal. First, 1023_{10} to binary:

$\begin{array}{r} 511 \\ 2 \overline{)1023} \end{array}$	$R = 1$ LSBit	
$\begin{array}{r} 255 \\ 2 \overline{)511} \end{array}$	$R = 1$	\uparrow 1111111111_2
$\begin{array}{r} 127 \\ 2 \overline{)255} \end{array}$	$R = 1$	
$\begin{array}{r} 63 \\ 2 \overline{)127} \end{array}$	$R = 1$	
$\begin{array}{r} 31 \\ 2 \overline{)63} \end{array}$	$R = 1$	
$\begin{array}{r} 15 \\ 2 \overline{)31} \end{array}$	$R = 1$	
$\begin{array}{r} 7 \\ 2 \overline{)15} \end{array}$	$R = 1$	
$\begin{array}{r} 3 \\ 2 \overline{)7} \end{array}$	$R = 1$	
$\begin{array}{r} 1 \\ 2 \overline{)3} \end{array}$	$R = 1$	
$\begin{array}{r} 0 \\ 2 \overline{)1} \end{array}$	$R = 1$ MSBit	

Now, arranging this number into three groups of four bits each and then converting each group to its hexadecimal counterpart, the result is 1023_{10} , represented in hexadecimal $1111111111_2 = 0011\ 1111\ 1111_2 = 3FF_{16}$. Where $0011_2 = 3_{16}$ and $1111_2 = F_{16}$ (from Table 1).

In summary, remember that *each hexadecimal (base 16) digit is a representation of 4 binary bits*. It is easy to convert from hex to binary and binary to hex. For convenience, a limited conversion chart follows.

*From Table I.

CONVERSION CHART

Decimal	Octal	Hexadecimal	Binary
0	0	0	0000 0000
1	1	1	0000 0001
2	2	2	0000 0010
3	3	3	0000 0011
4	4	4	0000 0100
5	5	5	0000 0101
6	6	6	0000 0110
7	7	7	0000 0111
8	10	8	0000 1000
9	11	9	0000 1001
10	12	A	0000 1010
11	13	B	0000 1011
12	14	C	0000 1100
13	15	D	0000 1101
14	16	E	0000 1110
15	17	F	0000 1111
16	20	10	0001 0000
17	21	11	0001 0001
18	22	12	0001 0010
19	23	13	0001 0011
20	24	14	0001 0100
21	25	15	0001 0101
22	26	16	0001 0110
23	27	17	0001 0111
24	30	18	0001 1000
25	31	19	0001 1001
26	32	1A	0001 1010
27	33	1B	0001 1011
28	34	1C	0001 1100
29	35	1D	0001 1101
30	36	1E	0001 1110
31	37	1F	0001 1111
32	40	20	0010 0000
33	41	21	0010 0001
34	42	22	0010 0010
35	43	23	0010 0011
36	44	24	0010 0100
37	45	25	0010 0101
38	46	26	0010 0110
39	47	27	0010 0111
40	50	28	0010 1000

Two's Complement

The M6800 system does not do direct subtraction, so the method of 2's complement addition is used to accomplish the subtraction. The 2's complement of any binary number is its *additive inverse*. That is, a binary number plus its 2's complement always equals zero. Or,

$$\begin{array}{r} 11011011 \\ + 00100101 \quad \text{2's complement of 11011011} \\ \hline 00000000 \end{array}$$

How is the 2's complement of a binary number computed? There are several methods. One way of calculating the 2's complement is to take the number to be converted, invert all the digits, then add one.

For example, find the 2's complement of 01011011. (91_{10})

$$\begin{array}{r} \text{First, invert} \quad 01011011 \\ \text{Equals} \quad 10100100 \\ \text{Add 1} \quad + \quad \underline{\quad 1} \\ \hline 10100101 \end{array}$$

Therefore, 10100101 is the 2's complement of 01011011.

The following are examples of subtraction by the method of 2's complement addition:

- Given $61 - 12 = ?$ (base 10), or

	Binary Notation.	Hex Notation
61 =	00111101	3D
- 12	- <u>00001100</u>	- <u>0C</u>

But to do the subtraction, first convert 00001100 (12_{10}) to a 2's complement number.

	Binary Notation	Hex Notation
Therefore, given	00001100	0C
a. Invert	11110011	F3
b. Add	+ <u>00000001</u>	+ <u>01</u>
	11110100 2's complement	F4

So the subtraction becomes 2's complement addition

	00111101	3D
	+ <u>11110100</u>	+ <u>F4</u>
ans	00110001	ans 31

As a check

$$\begin{array}{r} 61_{10} \\ - 12_{10} \\ \hline = 49_{10} = 00110001_2 = 31_{16} \end{array}$$

2. Given $61 - 2 = ?$ (base 10)

$$\begin{array}{r} 61 \\ - 2 \\ \hline \end{array}$$

Binary Notation

$$\begin{array}{r} 00111101 \\ 00000010 \\ \hline \end{array}$$

Hex Notation

$$\begin{array}{r} 3D \\ - 02 \\ \hline \end{array}$$

Doing the 2's complement addition

Binary Notation

$$\begin{array}{r} 00111101 \\ + 11111110^* \\ \hline \end{array}$$

ans

$$00111011$$

Hex Notation

$$\begin{array}{r} 3D \\ FE^{**} \\ \hline \end{array}$$

ans

$$3B$$

For further information see page 1-21 of the M6800 Microprocessor Applications Manual

* 2's complement of 00000010

** Hex notation of 2's complement

8-BIT 2'S COMPLEMENT

DECIMAL	BINARY	HEXADECIMAL
+127	0111 1111	7F
•	•	•
•	•	•
•	•	•
+64	0100 0000	40
•	•	•
•	•	•
•	•	•
+2	0000 0010	02
+1	0000 0001	01
0	0000 0000	00
-1	1111 1111	FF
-2	1111 1110	FE
•	•	•
•	•	•
•	•	•
-64	1100 0000	C0
•	•	•
•	•	•
•	•	•
-127	1000 0001	81
-128	1000 0000	80

Homework – Number Systems

Convert the following base 10 numbers to base 2 and base 16. Prove each base 2 and base 16 number is equal to its original base 10 number.

	Decimal	Binary	Hexadecimal
a)	92_{10}	_____	_____
b)	144_{10}	_____	_____
c)	4091_{10}	_____	_____
d)	254_{10}	_____	_____
e)	256_{10}	_____	_____
f)	$64,522_{10}$	_____	_____
g)	2000_{10}	_____	_____

Convert the following decimal numbers to 8-bit 2's complement representation.

- h) -17_{10} _____
- i) $+25_{10}$ _____
- j) -5_{10} _____
- k) -128 _____

Turn this page over to check your answers.

N-10 Number System

Answers

	Decimal	Binary	Hexadecimal
a)	92	1011100	5C
b)	144	10010000	90
c)	4091	111111111011	FFB
d)	254	11111110	FE
e)	256	100000000	100
f)	64522	1111110000001010	FCOA
g)	2000	11111010000	7D0

	Decimal	8-Bit 2's Complement Representation
h)	-17	11101111
i)	+25	00011001
j)	-5	11111011
k)	-128	10000000

MPU

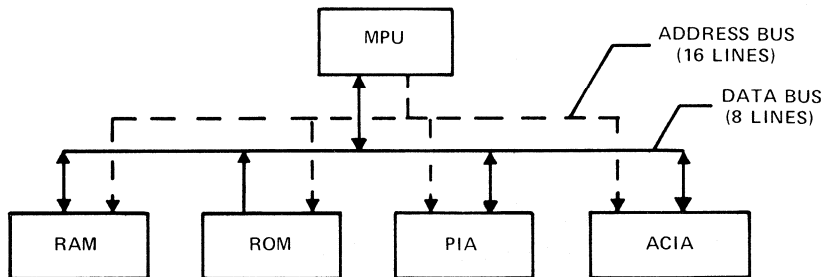
MICROPROCESSING UNIT (MC6800)

Introduction

The Motorola M6800 Microcomputer System of standard LSI (Large Scale Integration) devices permits the systems designer to configure and connect a total system with a minimum amount of time and effort. The MC6800 Microprocessing Unit (MPU) forms the nucleus of the system. LSI circuits available which may be used to configure a total system in conjunction with the MC6800 MPU include: 1) MCM6810 Random Access Memory (RAM), 2) MCM6830 Read Only Memory (ROM), 3) MC6821 Peripheral Interface Adapter (PIA), and 4) MC6850 Asynchronous Communications Interface Adapter (ACIA).

The MPU communicates with the rest of the system via a 16-bit address bus and an 8-bit data bus. The 16-bit address bus provides the MPU with the capability of addressing up to 64K. The 8-bit data bus is bidirectional in that data is transferred both into the MPU or out of the MPU over the same bus. A read/write (R/W) line is provided to allow the MPU to control the direction of data transfer. Since the same bus is used both for data into the MPU and out of the MPU, a separate 8-line bus is saved.

Other features of the M6800 system include a single +5 volt supply, operation at clock rates from 100 kilohertz to 1 megahertz, plus hardware and software interrupt-capability.

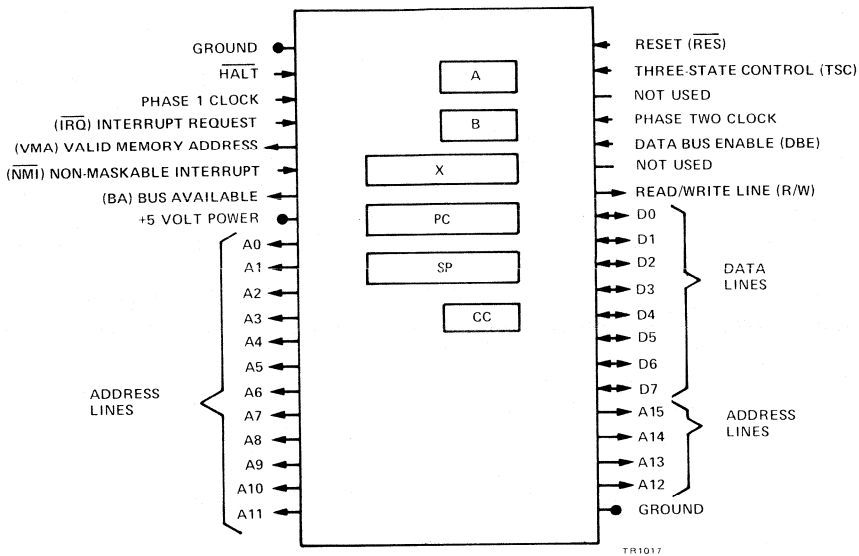


D1526

MICROPROCESSING UNIT (MC6800)

The nucleus of the M6800 Microcomputer Family is the microprocessing unit (MPU). The MPU is enclosed in a 40-pin package as shown on the following page.

MPU-2 Microprocessing Unit (MC6800)



Features included in the MPU are:

1. Two accumulators (A and B)
2. One index register (X)
3. One program counter register (PC)
4. One stack pointer register (SP)
5. One condition code register (CC)
6. 72 instructions
7. Six addressing modes
8. System clock range of 100 kilohertz to 1 megahertz
9. Program interrupt capability

ACCUMULATORS

The MPU contains 2 accumulators designated A and B. Each accumulator is 8 bits (one byte) long and is used to hold operands and data from the arithmetic logic unit.

INDEX REGISTER

The index register (X) is a 16-bit (2 byte) register which is primarily used to store a memory address in the indexed mode of memory addressing. The index register may be decremented, incremented, and stored.

PROGRAM COUNTER

The program counter (PC) is a 16-bit register that contains the address of the next byte to be fetched from memory. When the current value of the program counter is placed on the address bus, the program counter will be incremented automatically.

STACK POINTER

The stack pointer (SP) is a 16-bit (2 byte) register that contains a beginning address, normally in RAM, where the contents of the MPU registers may be stored when the MPU has other functions to perform such as an interrupt or a Subroutine. The address in the stack pointer is the starting address of sequential memory locations in RAM where MPU contents of the registers will be stored. The contents of the MPU will be stored in the RAM as follows:

Stack Point Address	:	contents of PCL
Stack Pointer Address-1	:	contents of PCH
Stack Pointer Address-2	:	contents of XL
Stack Pointer Address-3	:	contents of XH
Stack Pointer Address-4	:	contents of A
Stack Pointer Address-5	:	contents of B
Stack Pointer Address-6	:	contents of CC

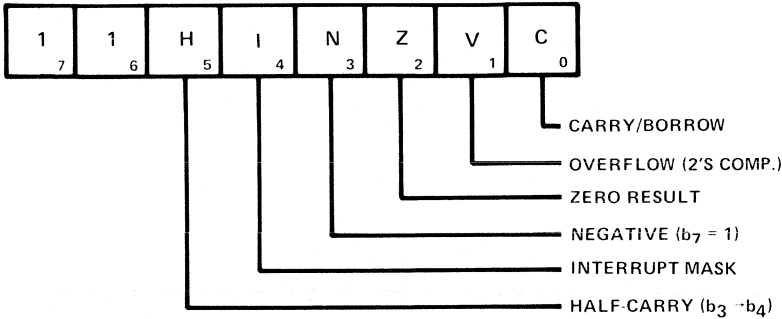
After the contents of each register is stored on the stack, the stack pointer will be decremented. When the stack is unloaded (contents of registers restored), the contents of the last register on the stack will be the first register that is restored.

CONDITION CODE REGISTER (CC)

The condition code register is an 8-bit register. Each *individual* bit may be set or cleared by execution of an instruction. To see how each instruction affects the condition code register, refer to the M6800 programming manual. This register is used by the conditional branch instructions. Bits 6 and 7 are not used and remain at logic "1".

Carry-Borrow	For addition, the carry-borrow condition code (C) in the zero-bit position, represents a carry. This bit gets set (C = 1) to indicate a carry, and is reset (C = 0), if there is no carry.
Overflow	The V-bit (bit 1) of the condition code register is set (V = 1) when two's complement overflow results from an arithmetic operation, and is reset (V = 0), if two's complement overflow does not occur.
Zero	The Z-bit (bit 2) of the condition code register is set (Z = 1), if the result of an arithmetic operation is zero, and is reset (Z = 0), if the result is not zero.

CONDITION CODE REGISTER



BITS SET AS A RESULT OF PREVIOUS OPERATION!

ABA:	A= 1000 1000	A+B=	$\overset{1}{\boxed{1}}0001\ 0000$;	H = 1, Z = 0
	B= 1000 1000				C = 1, N = 0
					V = 1
DEC A	A= 0000 0001	A-1=	0000 0000	;	Z = 1, N = 0
LDA A = \$80		A=	1000 0000	;	N = 1, Z = 0, V = 0
COM A	A= 1000 0000	\bar{A} =	0111 1111	;	N = 0, V = 0, C = 1, Z = 0
OR DEC A				;	N = 0, V = 1, C = UNCHANGED, Z = 0
ABA	A= 1000 0010 = -126 ₁₀	A+B =	$\overset{1}{\boxed{1}}0000\ 0100$;	V = 1, Z = 0
	B= 1000 0010 = -126 ₁₀	=	+4 ₁₀	;	C = 1, N = 0
					H = 0

TR1022

- Negative** The N-bit (bit 3) of the condition code register is set (N = 1), if bit 7 of any operation is set (equal to 1). The N-bit is reset (N = 0), if bit 7 of any result is equal to 0.
- Interrupt Mask** If this bit is set (I = 1), IRQ interrupts are inhibited. If I = 0, the processor may be interrupted by IRQ being in the low state. The I bit is set via SEI instruction or by an interrupt occurring (IRQ, NMI, or SWI). This bit is cleared with RTI (assuming I bit was clear before interrupt) or CLI.
- Half-Carry** The half-carry bit H (bit 5) of the condition code register is set (H = 1) during execution of any of the instructions ABA, ADC, or ADD, if there is a carry from bit position 3 to bit position 4. The half-carry is reset (H = 0) during these operations, if there is no carry from bit position 3.

NOTE: The information the condition code register holds is the results of the instruction that last affected the condition code register.

MPU Signal Descriptions

1. **READ/WRITE (R/W)** This output line is used to signal all devices external to the MPU that the MPU is in a read state (R/W = high) or a write state (R/W = low). The normal standby state of this line when no external devices are being accessed is a high state. This line is three state. When three state control goes high, this line enters the high-impedance mode.
2. **VALID MEMORY ADDRESS (VMA)** This output line (when in the high state), tells all devices external to the MPU that there is a valid address in the address bus. This signal *is not three state*.
3. **DATA BUS ENABLE (DBE)** This signal will enable the data bus drives when in the high state. This input is normally the phase 2 (ϕ_2) clock. During the high state, it will permit data to be output during a write cycle. During an MPU read cycle, the data bus drives will be disabled internally.
4. **INTERRUPT REQUEST (IRQ) (Level Sensitive Pin)** This input requests that an interrupt sequence be generated. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the condition code register is not set (interrupt masked), the machine will begin an interrupt sequence. The index register, program counter, accumulators, and condition code register are stored on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations n-6 and n-7. Where n is the highest ROM address. An address loaded at these locations causes the MPU to jump to an interrupt routine in memory.
5. **PHASE ONE (ϕ_1) AND PHASE TWO (ϕ_2) CLOCKS** These two pins are used for a two-phase, non-overlapping clock that runs at the V_{DD} voltage level. These clocks run at a rate up to 2 MHz for MC68B00.
6. **RESTART (\overline{RES})** This input is used to start the MPU from a power-down condition, resulting from a power failure or an initial start-up of the processor. If a positive edge is detected on the input, this will signal the MPU to begin the restart sequence. This will restart the MPU and start execution of a routine to initialize the processor. All the higher order address lines will be forced high.

For the restart, the last two memory locations in the last ROM (n and n-1) will be accessed, whereby an address is stored which is the address to be loaded in the program counter which tells the processor where program execution is to begin.

7. **NONMASKABLE
INTERRUPT (NMI)**
(Edge Sensitive Pin)

This input requests that a nonmask-interrupt sequence be generated within the processor. As with the Interrupt Request signal, the processor will complete the current instruction that is being executed before it recognizes the NMI signal. The interrupt mask bit in the condition code register has no effect on NMI. However, NMI does set the Interrupt Mask bit.

The index register, program counter, accumulators, and condition code register are stored away on the stack. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations n-2 and n-3. An address loaded at these locations causes the MPU to jump to a non-maskable interrupt routine in memory.

8. **GO/HALT (G/H)**

When this input is in the high state, the machine will fetch the instruction addressed by the program counter and start execution. When low, all activity in the machine will be halted. This input is level sensitive. In the halt mode, the machine will stop at the end of instruction. Bus Available will be at a logic "1" level Valid Memory Address will be at a logic "0" and all other three-state lines will be in the three-state mode.

The halt line must go low with the leading edge of phase one to insure single instruction operation. If the halt line does not go low with the leading edge of phase one, one or two instruction operations may result, depending on when the halt line goes low relative to the phasing of the clock.

9. **BUS AVAILABLE
(BA)**

The Bus Available signal will normally be in the low state. When activated, it will go to the high state indicating that the MPU has stopped and that the address bus is available. This will occur if the Go/Halt line is in the halt (low) mode or the MPU is in a "wait" state as the result of some instruction such as the WAI instruction.

- 10. **THREE-STATE CONTROL (TSC)** This input causes all of the address lines and the Read/Write line to go into the off or high-impedance state. The Valid Memory Address and Bus Available signals will be forced low. The data bus is not affected by TSC and has its own enable (Data Bus Enable). In DMA applications, the Three-State Control line should be brought high on the leading edge of the Phase One Clock. The $\phi 1$ clock must be held in the high state for this function to operate properly. The address bus will then be available for other devices to directly address memory. Since the MPU is a dynamic device, it must be refreshed periodically, or destruction of data will occur.

- 11. **ADDRESS BUS (A0/A15)** Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 90 pF at 2 megahertz. When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications.

- 12. **DATA BUS (D0/D7)** Eight pins are used for the data bus. It is bidirectional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load and 130 pF at 2 megahertz.

Now that we have talked about the pins of the MPU package, we will talk about the "heartbeat" of the system—the clock.

The MPU Clock

The MPU clock driver must meet the minimum and/or maximum criteria. These criteria are explained by the table accompanying the timing waveform. (See MPU-10.) The information to remember is that on the falling edge of $\phi 1$, the program counter is advanced, and, on the falling edge of $\phi 2$, the data is latched into the MPU.

1	2	3	4	5
Program Memory Address	Opcode/ Data	Comments	Addressing Modes	Mnemonic
1000	86	LDA A with	Immediate	LDA A #\$5
1001	05	Hex 05		
1002	D6	LDA B with	Direct	LDA B \$F1
1003	F1	the contents of F1=04 (hex)		
1004	1B	Add contents of Accum B to contents of Accum A and store in A	Accumulator	ABA

Up to this point we have physically described the microprocessor, i.e., the accumulators and registers. We will now explain how this microprocessor works. This will be easier if we use a small program to trace the flow of data through the MPU system.

The program shown on MPU-7 will be used; and, in conjunction with MPU-8, a description of the operation of the MPU will be described.

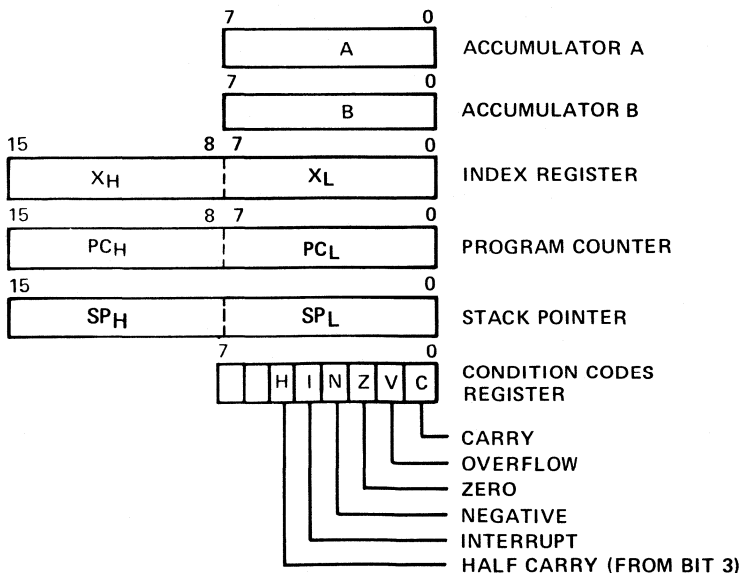
The program shown above is divided into 5 columns. Column 1 is the Program Memory Address column. Here we show the address locations where the program is stored. Column 2 contains the instruction opcode or data. This information is in hexadecimal notation. Column 3 is a comment column to aid you in understanding what task is to be performed. Column 4 describes the addressing mode used. More information is contained in the addressing mode section. The fifth column gives the mnemonic for the particular instruction used.

When Phase 1 (ϕ_1) goes high, the contents of the program counter are transferred to the address bus. While this action is taking place, VMA will go to a logic 1 indicating a valid address. On the falling edge of ϕ_1 , the program counter will be incremented (if required) by one. When Phase 2 (ϕ_2) goes high, data is placed on the data bus; and, during the falling edge of ϕ_2 , the data is latched into the MPU. This sequence occurs every time the MPU addresses a memory location and data is moved.

Let us return to the program and begin with ϕ_1 going to a logic 1. The contents of the program counter (we will assume to be 1000 hex)* will be transferred to the address bus. ϕ_1 goes low and the program counter is incremented to 1001. Memory location 1000 has been selected; and, when ϕ_2 goes to a logic 1, its contents will be transferred to the data bus. Looking at our program on MPU-7, we see that the data stored in memory location 1000 is 86. Therefore, 86 is the data transferred to the data bus. Again, during the falling edge of ϕ_2 , the data on the bus is latched into the MPU.

*All addresses will be given in hexadecimal notation.

PROGRAMMING MODEL OF THE MICROPROCESSING UNIT

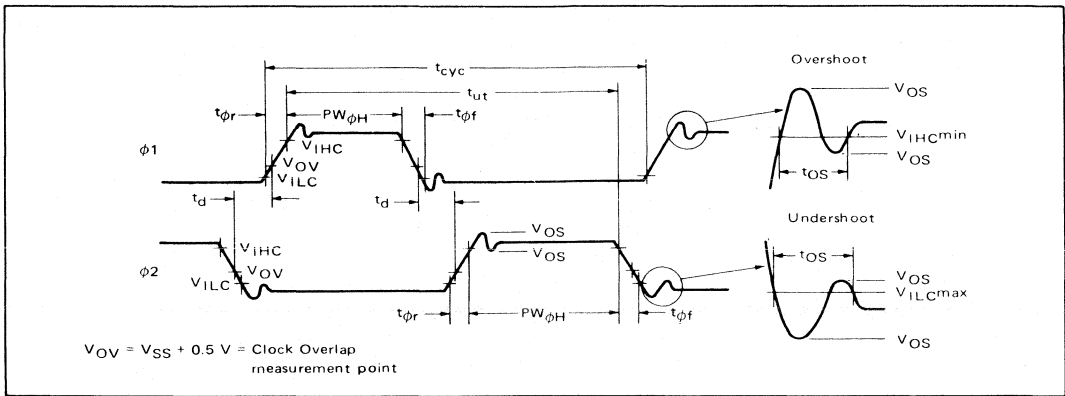


EXECUTABLE INSTRUCTIONS — ALPHABETIC LIST

ABA	ADD ACCUMULATORS	INS	INCREMENT STACK POINTER
ADC	ADD WITH CARRY	INX	INCREMENT INDEX REGISTER
ADD	ADD	JMP	JUMP
AND	LOGICAL AND	JSR	JUMP TO SUBROUTINE
ASL	ARITHMETIC SHIFT LEFT	LDA	LOAD ACCUMULATOR
ASR	ARITHMETIC SHIFT RIGHT	LDS	LOAD STACK POINTER
BCC	BRANCH IF CARRY CLEAR	LDX	LOAD INDEX REGISTER
BCS	BRANCH IF CARRY SET	LSR	LOGICAL SHIFT RIGHT
BEQ	BRANCH IF EQUAL TO ZERO	NEG	NEGATE
BGE	BRANCH IF GREATER OR EQUAL ZERO	NOP	NO OPERATION
BGT	BRANCH IF GREATER THAN ZERO	ORA	INCLUSIVE OR ACCUMULATOR
BHI	BRANCH IF HIGHER	PSH	PUSH DATA
BIT	BIT TEST	PUL	PULL DATA
BLE	BRANCH IF LESS OR EQUAL	ROL	ROTATE LEFT
BLS	BRANCH IF LOWER OR SAME	ROR	ROTATE RIGHT
BLT	BRANCH IF LESS THAN ZERO	RTI	RETURN FROM INTERRUPT
BMI	BRANCH IF MINUS	RTS	RETURN FROM SUBROUTINE
BNE	BRANCH IF NOT EQUAL TO ZERO	SBA	SUBTRACT ACCUMULATORS
BPL	BRANCH IF PLUS	SBC	SUBTRACT WITH CARRY
BRA	BRANCH ALWAYS	SEC	SET CARRY
BSR	BRANCH TO SUBROUTINE	SEI	SET INTERRUPT MASK
BVC	BRANCH IF OVERFLOW CLEAR	SEV	SET OVERFLOW
BVS	BRANCH IF OVERFLOW SET	STA	STORE ACCUMULATOR
CBA	COMPARE ACCUMULATORS	STS	STORE STACK REGISTER
CLC	CLEAR CARRY	STX	STORE INDEX REGISTER
CLI	CLEAR INTERRUPT MASK	SUB	SUBTRACT
CLR	CLEAR	SWI	SOFTWARE INTERRUPT
CLV	CLEAR OVERFLOW	TAB	TRANSFER ACCUMULATORS
CMP	COMPARE	TAP	TRANSFER ACCUMULATORS TO CONDITION CODE REG
COM	COMPLEMENT	TBA	TRANSFER ACCUMULATORS
CPX	COMPARE INDEX REGISTER	TPA	TRANSFER CONDITION CODE REG TO ACCUMULATOR
DAA	DECIMAL ADJUST	TST	TEST
DEC	DECREMENT	TSX	TRANSFER STACK POINTER TO INDEX REGISTER
DES	DECREMENT STACK POINTER	TXS	TRANSFER INDEX REGISTER TO STACK POINTER
DEX	DECREMENT INDEX REGISTER	WAI	WAIT FOR INTERRUPT
EOR	EXCLUSIVE OR		
INC	INCREMENT		

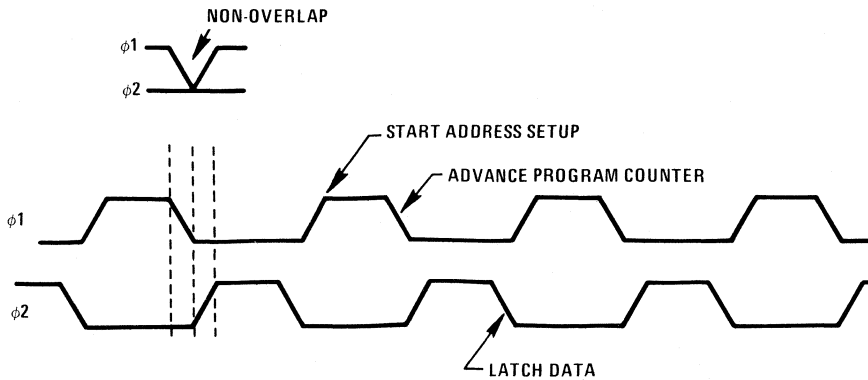
TR 1090

MPU-10 Microprocessing Unit (M6800)



	MC6800	Min	Max		
Clock Timing (Figure 1)					
Cycle Time	t_{cyc}	1.0	—	10	μs
Clock Pulse Width (Measured at $V_{CC} - 0.3\text{ V}$)	$PW_{\phi H}$	400	—	9500	ns
Total $\phi 1$ and $\phi 2$ Up Time	t_{ut}	900	—	—	ns
Rise and Fall Times (Measured between $V_{SS} + 0.3\text{ V}$ and $V_{CC} - 0.3\text{ V}$)	$t_{\phi r}, t_{\phi f}$	—	—	100	ns
Delay Time or Clock Separation (Measured at $V_{OV} = V_{SS} + 0.6\text{ V}$)	t_d	0	—	9100	ns
Overshoot Duration	t_{OS}	0	—	40	ns

MPU CLOCK WAVE FORM



TR1018

This information will be decoded as an opcode versus data, since it will be assumed the last instruction was terminated.

Looking at MPU-7, we see that LDA A requires 2 cycles and 2 bytes of data. The first cycle was used to bring in the first byte of data, 86, into the MPU. On the next $\phi 1$ cycle, the program counter is transferred to the address bus and 1001 is placed on the address bus selecting memory location 1001. $\phi 1$ now falls and the program counter is incremented by 1 to 1002. $\phi 2$ rises to a logic 1 and 05 is the data placed on the data bus. $\phi 2$ goes low and 05 is now latched into the A accumulator. Here we have used the immediate mode of addressing. In the immediate mode of addressing, the data is contained in the second byte of the instruction. $\phi 1$ goes high and the program counter transfers its contents to the address bus which is 1002. $\phi 1$ goes low and the program counter is incremented to 1003 hex. $\phi 2$ goes high and D6 is the data placed on the data bus. Since the MPU finished the previous instruction (2 cycles, 2 bytes), it now knows that the next data D6 is an opcode. To accomplish this instruction we need 3 cycles and 2 bytes of data. We have used 1 cycle and 1 byte of data fetching the opcode. Now $\phi 1$ goes high and the contents of the program counter (1003) is transferred to the address bus. $\phi 2$ goes high and F1 is now placed on the data bus. $\phi 2$ falls and F1 is latched into the MPU. Notice that we have used the direct mode of addressing. This mode of addressing can be used when addressing memory locations 0 through 255 decimal. In this mode, the second byte of data is an address.

To review, in the direct mode, the first byte is the opcode and the second byte is the address. Finally, on the third cycle, the data at memory location F1 is transferred to the B accumulator. On this last cycle, the program counter does not have to and will not increment. $\phi 1$ now goes high and the program counter transfers 1004 hex to the address bus. $\phi 1$ goes low and the program counter is incremented to 1005 hex. $\phi 2$ goes high and 1B is placed on the data bus; and, on the falling edge of $\phi 2$, 1B is latched into the MPU. To complete this instruction we need 2 cycles and 1 byte of data. The mode of addressing we have used here is the accumulator mode. In this mode of address the opcode and operand are contained in the first byte of data. All that that is needed to terminate this instruction is another cycle to add the contents of accumulator B to the contents of accumulator A. After this cycle has been completed, accumulator A has 09 hex and accumulator B has 04.

Any program can be handled in this fashion. Refer to a copy of the instruction set summary chart. By looking at the mnemonic, you can obtain the hex opcode for the addressing mode used. The required number of cycles and bytes are also given so you can "step" through a particular program.

HARDWARE INTERRUPTS

What happens when the MPU gets a hardware interrupt? After it has been determined that the interrupt is not nonmaskable, the MPU checks the status of the mask bit (bit 4 of the condition code register). If the mask bit is set, the main program continues until a CLI (clears bit 4 of condition code register) instruction is executed, after which time the MPU will honor an interrupt by going to the stack pointer (SP) register and will fetch an address which will be the first address in RAM where the status of the MPU registers will be stored during servicing of the interrupt.

SP	:	contents of program counter low
SP-1	:	contents of program counter high
SP-2	:	contents of index register low
SP-3	:	contents of index register high
SP-4	:	contents of accumulator A
SP-5	:	contents of accumulator B
SP-6	:	contents of condition code register

The address in the stack pointer register is determined by the programmer.

After the contents of the MPU registers have been stored in the stack, the mask bit is set, thus preventing any further interrupts from interfering with the MPU until the program executes a CLI instruction. Next the MPU hardware automatically looks at addresses FFF8 (MS) and FFF9 (LS) for the address of the polling routine to find out where the interrupt came from and what action to take.

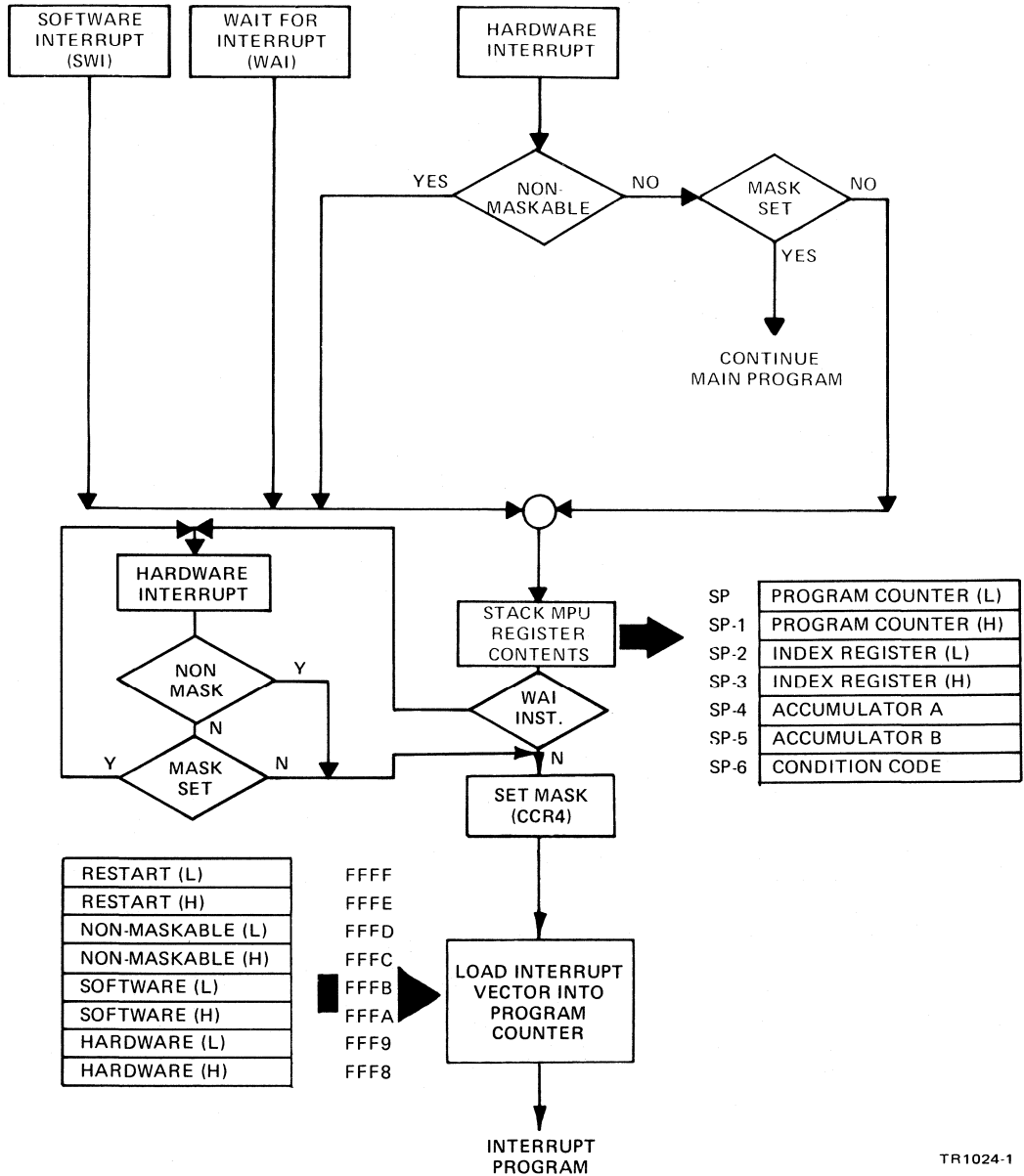
After the interrupt has been serviced and an RTI instruction is executed, the stack—which contains the contents of the registers before the interrupt—is unloaded in reverse order, i.e., the condition code register is loaded first, then accumulator B is restored, etc. When the registers have been restored to their contents before the interrupt, the processor continues as though nothing happened.

The total story of interrupts is shown on the opposite page in the form of a flow chart.

SUMMARY OF MPU OPERATION

The MPU requires a two-phase, symmetrical, nonoverlapping clock. During the first phase of the clock ($\phi 1$ high), an address will be placed on the address bus by the MPU. During the second phase of the clock ($\phi 2$ high), the bidirectional data bus will be active. The first byte of an instruction enters the MPU and is transferred into an internal instruction register and decoded by the MPU. The MPU will then contain the information needed to read in an additional one or two bytes of program as necessary. Once the entire instruction is read into the MPU (one, two, or three bytes), the instruction is then executed. The MPU then reads in the next sequential byte in the program and places it again in the instruction register. The program will sequentially be executed in this manner unless a branch or jump instruction changes the value of the program counter. If this occurs, the next instruction to be executed is determined by the new program counter value.

INTERRUPT FLOW CHART

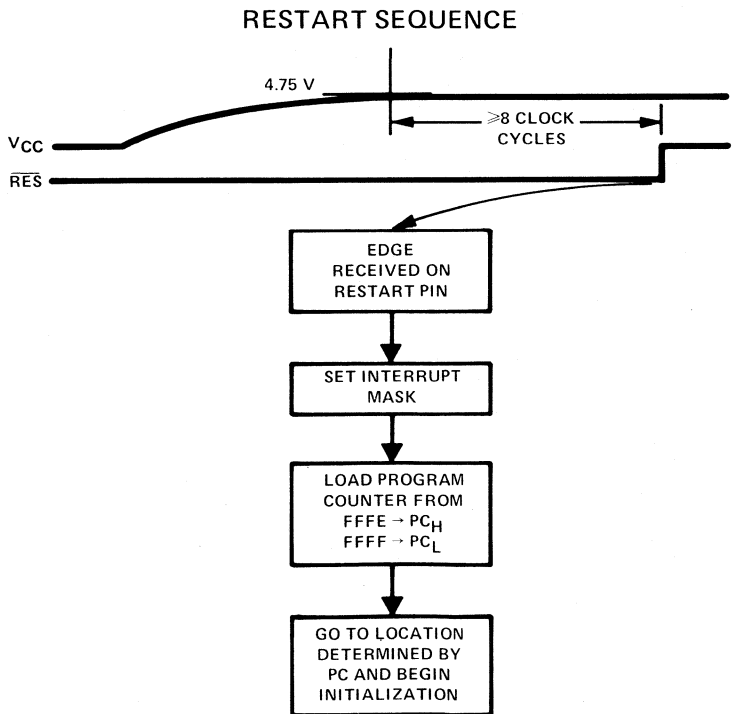


If an interrupt or reset occurs during this process, the program counter value will also be changed. The new program counter value is determined by the highest eight memory locations that are reserved for reset and interrupt vectors.

In the case of interrupt, the stack pointer is used to store the contents of the internal registers necessary to return to the program location prior to the interrupt. This happens when the interrupt program exits by an RTI (Return from Interrupt instruction). Similarly, the stack pointer is used to store the program counter value when a JSR (Jump to Subroutine) or BSR (Branch to Subroutine) instruction occurs. The program counter returns to its original value when an RTS (Return from Subroutine) instruction occurs. The stack pointer value is set by an LDS (Load Stack Pointer) instruction.

RESET SEQUENCE (Figure TR1029, below)

1. While $\overline{\text{HALT}}$ is high, $\overline{\text{Reset}}$ goes low for at least eight cycles of $\phi 1$, $\phi 2$, during which interrupt bit (I) in CC is set.
2. Data at FFFE loads into PCH.
3. Data at FFFF loads into PCL.
4. PC contents go out on address bus during $\phi 1$.



TR1029

5. Contents of memory location addressed enters instruction register during $\phi 2$ and is decoded as first instruction.
6. If two or more byte instruction, additional bytes enter MPU for execution. If not, go to next step.
7. After execution, step 5 is repeated for subsequent instructions.

$\overline{\text{IRQ}}$ SEQUENCE

1. If the I bit in condition code register is not set ($I = 0$) and $\overline{\text{IRQ}}$ goes low, the $\overline{\text{IRQ}}$ sequence will be entered.
2. After completion of the current instruction, internal registers PC, X, A, B, and CC will be stored in RAM at the address indicated by the stack pointer in descending locations (7 bytes in all).
3. The $\overline{\text{IRQ}}$ mask (bit I = 1) is set.
4. Data at FFF8 gets loaded into PCH.
5. Data at FFF9 gets loaded into PCL.
6. PC contents go out on address bus during $\phi 1$.
7. Contents of memory location addressed enters instruction register during $\phi 2$ and is decoded as first instruction of interrupt routine.
8. If it is a more than 1 byte instruction, additional bytes enter MPU for execution. If not, go to next step.
9. After execution, step 7 is repeated for subsequent instructions. This loop is repeated until the instruction "RTI" is executed.

$\overline{\text{NMI}}$ SEQUENCE

1. If $\overline{\text{NMI}}$ goes low, the MPU will wait for completion of current instruction.
2. The internal registers PC, X, A, B, and CC will then be stored in RAM at the address indicated by the stack pointer in descending locations (7 bytes in all).
3. The $\overline{\text{IRQ}}$ (bit I = 1) mask is set.
4. Data at FFFC is loaded into PCH.
5. Data at FFFD is loaded into PCL.
6. PC contents go out on address bus during $\phi 1$.
7. Contents of memory location addressed enters instruction register during $\phi 2$ and is decoded as first instruction of NMI subroutine.

MPU—16 Microprocessing Unit (M6800)

8. If two or more byte instruction, additional bytes enter MPU for execution. If not, go to next step.
9. After execution, step 7 is repeated for subsequent instructions. This loop is repeated until the instruction RTI is executed.

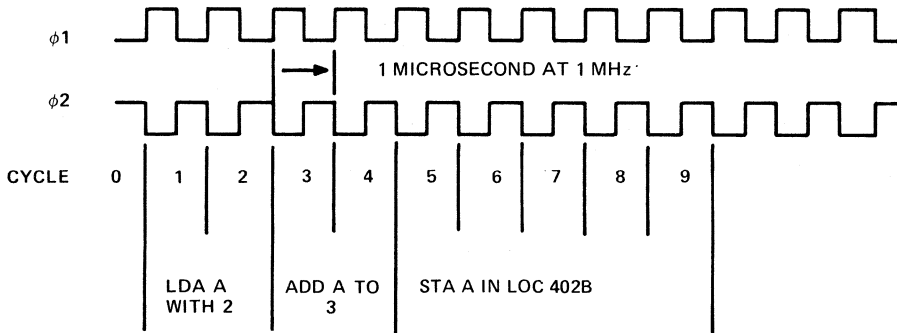
RTI EXECUTION

1. The contents of the stack are loaded back into the MPU (unwinds).
2. The contents of the PC go out on the address bus to fetch the first byte of the next instruction.

SWI INSTRUCTION

1. Contents of the MPU registers PC, X, A, B, and CC are stored in RAM at the address indicated by the stack pointer in descending location (7 bytes in all).
2. The $\overline{\text{IRQ}}$ mask (bit I = 1) is set.
3. Data at FFFA gets loaded into PCH.
4. Data at FFFB gets loaded into PCL.
5. PC contents go out on address bus during $\phi 1$.
6. Contents of memory location addressed enters instruction register during $\phi 2$ and is decoded as first instruction of SWI subroutine.
7. If it is a more than one byte instruction, additional bytes enter MPU for execution. If not, go to next step.
8. After execution, step 6 is repeated for subsequent instructions. This loop is repeated until the instruction RTI is executed.

CYCLE BY CYCLE DESCRIPTION OF SAMPLE PROGRAM



TR1232

ROM Address	ROM Content	Instruction
0018	86	LDA A #2
0019	02	
001A	8B	ADD A #3
001B	03	
001C	B7	STA A \$402B
001D	40	
001E	2B	

indicates immediate mode of addressing.
\$ indicates a hex number.

NOTE: Address 402B may be a RAM, PIA, or ACIA.

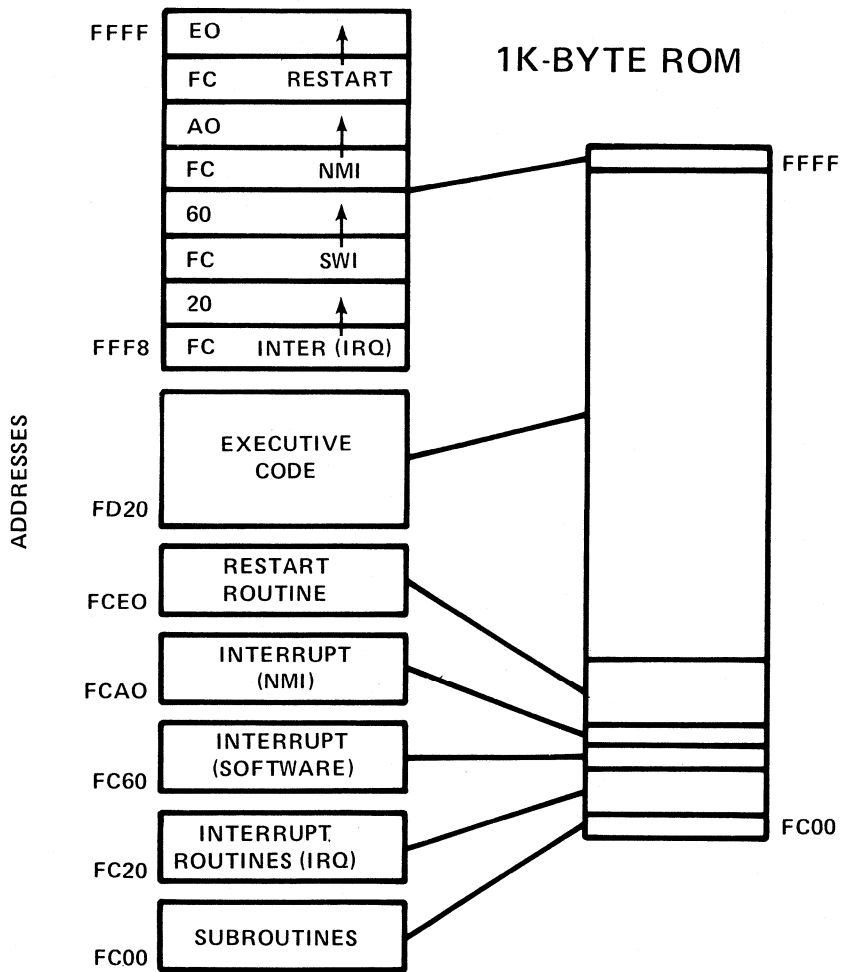
DESCRIPTION OF PROGRAM

The A accumulator is loaded with the number 2. Then the number 3 is added to the 2 in the A accumulator with the result of 5 left in the A accumulator. The 5 in the A accumulator is then stored in location 402B.

MPU-18 Cycle-by-Cycle Description of Sample Program

Cycle-by-Cycle Description of Sample Program

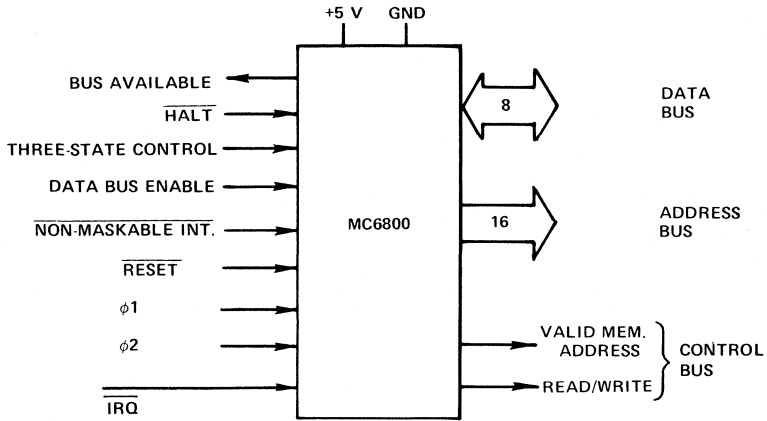
CYCLE	DESCRIPTION
0	The program counter is assumed to be set at 0018.
1	The program counter is gated onto the Address Bus (A0-A15) and the read/write (R/W) line is put in a high state corresponding to a read condition. This results in ROM address 0018 being accessed and the contents of this address (86) being loaded into the instruction register (IR). The program counter has incremented and becomes 0019. The byte "86" in the IR is decoded and interpreted to be a load A immediate (LDA A IMM) instruction.
2	The program counter is gated onto the Address Bus and the R/W line is set high corresponding to a read condition. This accesses ROM address 0019 with the contents of this address (02) being put on the Data Bus (D0-D7). Since the instruction was decoded to be an LDA A immediate, the "02" is loaded into the A accumulator. The program counter has incremented and becomes 001A.
3	The sequence in (1) is repeated except ROM address 001A is accessed resulting in 8B being loaded into the instruction register, and decoded to be an ADD A immediate. The program counter has incremented to 001B.
4	The sequence in (2) is repeated except the data "03" is added to the A accumulator giving a result in the A accumulator of "05". The program counter has incremented to 001C.
5	The sequence in (1) is repeated which results in B7 being loaded into the instruction register. The program counter has incremented to 001D. The instruction register is decoded and determined to be a STA A extended. This causes the MPU to interpret the next two sequential locations in memory (001D and 001E) as a 16-bit address with 001D the most significant and 001E the least significant half of the address.
6	The number (40) in ROM address 001D is read by the MPU and saved. The program counter has incremented to 001E.
7	The contents of ROM address 001E (2B) is read by the MPU and saved. The MPU now has a full 16-bit address saved of 402B. The program counter has incremented to 001F.
8	The extended address of 402B is gated onto the address bus register, and the Data Bus is gated to output the A accumulator to the addressed memory location.
9	Address 402B is accessed and the R/W line is put in a low state, corresponding to a write. The data in the A accumulator is then gated onto the data bus and stored in location 402B.



TR1028

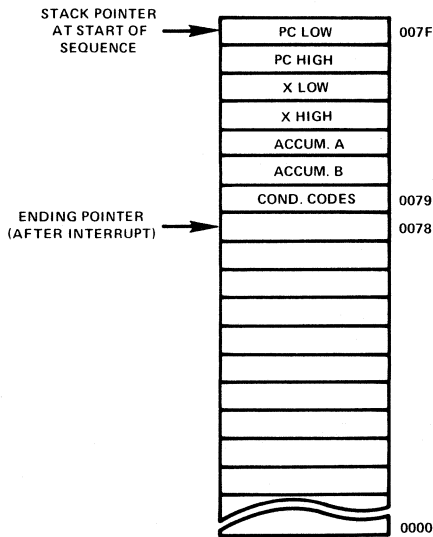
MEMORY		ADDRESS	CONTENT	COMMENTS	MODES	MNEMONIC
ADDRESS	CONTENT					
1000	86			LOAD ACC.A WITH HEX NO. 5 (DATA)	IMMED	LDA #5
1001	05					
1002	F6			LOAD ACC.B WITH DATA STORED AT MEM. LOCATION 2004 (DATA 0A)	EXTND	LDAB \$2004
1003	20					
1004	04					
1005	1B			ADD CONT. OF B ACC TO A ACC AND STORE IN A ACC.	INHER	ABA
1006	97					
1007	50			STORE CONT. OF A ACC. (OF) TO MEMORY LOCATION 50	DIR	STAA \$50
1008	CE					
1009	00			LOAD THE INDEX REGISTER WITH THE HEX NUMBER 80	IMMED	LDX #80
100A	80					
100B	A6					
100C	05			LOAD THE A ACC. WITH THE DATA STORED AT THE MEM. LOC. INDEX REG + 5(85)	INDXD	LDA \$5,X
100D	08			INCREMENT THE INDEX REG BY 1 (80 → 81)	INHER	INX
100E	8C					
100F	00			COMPARE THE INDEX REGISTER WITH THE HEX NUMBER 8A	IMMED	CPX #8A
1010	8A					
1011	26					
1012	F8			IF COMPARISON IS NOT EQUAL BRANCH BACK TO LOAD A ACC (100B)	REL	BNE [LABEL]
1013	8E					
1014	00			OTHERWISE LOAD STACK POINTER WITH HEX NUMBER 7F	IMMED	LDS #7F
1015	7F					

MC6800 BUS & CONTROL SIGNALS



TR1021

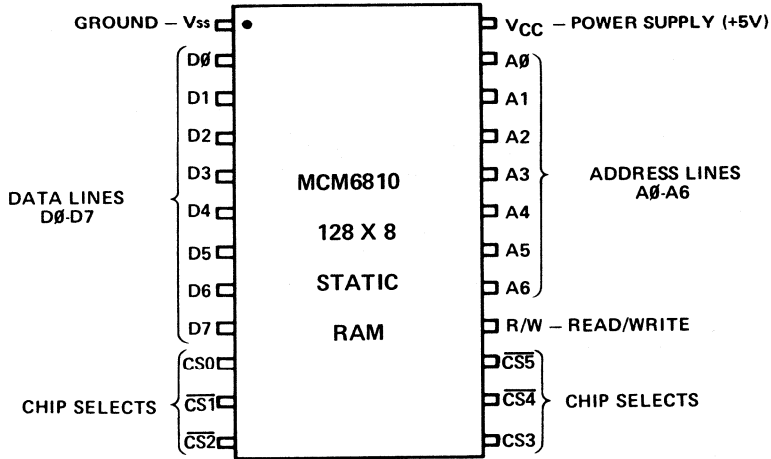
**RAM USED FOR STACK STORAGE
RAM ADDRESS \$0000-\$007F**



TR1025

Memory

RANDOM ACCESS MEMORY



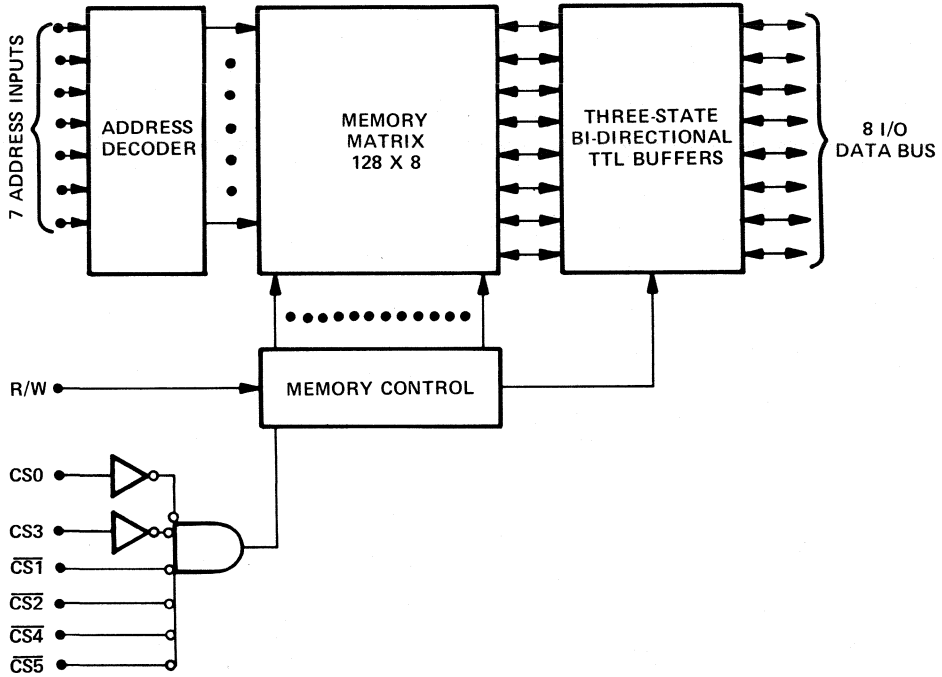
TR1031-2

RANDOM ACCESS MEMORY (RAM)

The MCM6810 is a TTL-compatible, static Random Access Memory (RAM). It is a three-state N-MOS chip containing 128 eight-bit words (128 bytes), housed in a 24-pin package. It has 8 data bus pins, 7 address pins, 6 chip select pins (2 active level high, 4 active level low), 2 power pins (ground and +5 V), and a read/write pin.

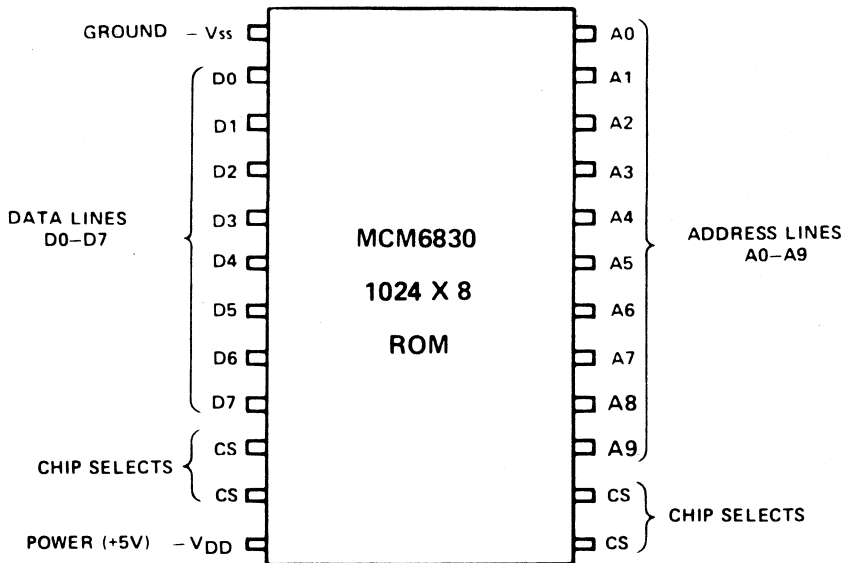
To access the RAM, all six chip selects must be at their proper levels. The R/W pin must be in a high state to read from the RAM, and in a low state to write into the RAM. When not being accessed, the RAM goes three-state, i.e., high impedance to the data bus. A functional block diagram is shown on the next page.

MCM6810 RAM FUNCTIONAL BLOCK DIAGRAM



TR1032-2

READ ONLY MEMORY



TR1033

READ ONLY MEMORY (ROM)

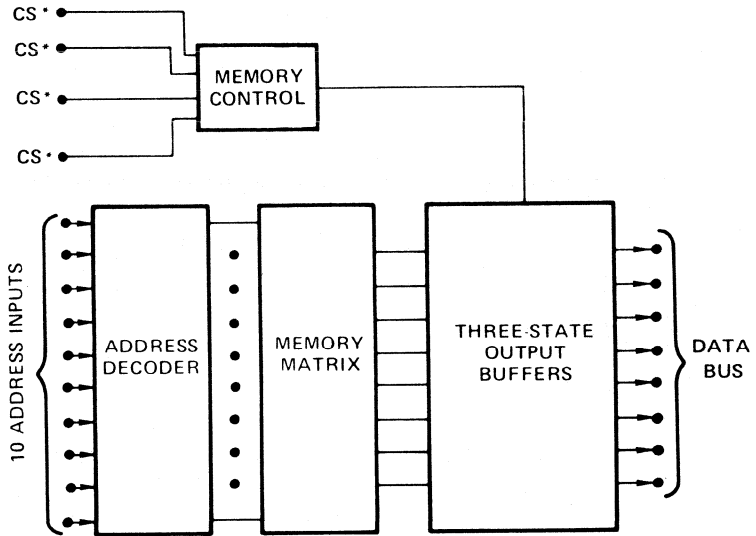
The MCM6830 is a static, TTL-compatible Read Only Memory (ROM). It is a single N-MOS chip containing 1024 eight-bit words (1024 bytes). The ROM is a three-state device housed in a 24-pin package consisting of 10 address pins, 8 data bus pins, 4 chip select pins, and 2 pins for power (+5 V and ground).

The chip selects are defined by the customer (mask programmable) to be either high or low active level.

To access the ROM, all four chip selects must be at their proper levels. When not accessed the device goes three-state, i.e., high input impedance to the data bus.

A functional block diagram is shown on the next page.

MCM6830 ROM FUNCTIONAL BLOCK DIAGRAM



*DEFINED BY THE CUSTOMER

TR1034

M6800 MEMORIES

MASK PROGRAMMABLE ROMS

			tACC
MCM68A30	1KX8	+5v	350ns
MCM6832	2KX8	-5v, +5v, +12v	500ns
MCM68A308	1KX8	+5v	350ns
MCM68A316	2KX8	+5v	350ns
MCM68A332	4KX8	+5v	350ns

UV ERASABLE PROMS

MCM68708	1KX8	-5v, +5v, +12v	450ns
MCM2708	1KX8	-5v, +5v, +12v	450ns
MCM2716	2KX8	-5v, +5v, +12v	450ns

TR1217-2

M6800 MEMORIES

PROMS

				tACC
MC7641	512X8	+5v	24 PIN	85ns
MC7643	1KX4	+5v	18 PIN	85ns

STATIC RAMS

MCM6810	128X8	+5v	24 PIN	450 ns
MCM2114	1KX4	+5v	18 PIN	200ns

DYNAMIC RAMS

MCM6604	4KX1	-5v, +5v, +12v	16 PIN	250 ns
MCM6605	4KX1	-5v, +5v, +12v	22 PIN	200ns
MCM4116	16KX1	-5v, +5v, +12v	16 PIN	150ns
MCM4096	4KX1	-5v, +5v, +12v	16 PIN	250ns
MCM4027	4KX1	-5v, +5v, +12v	16 PIN	150ns

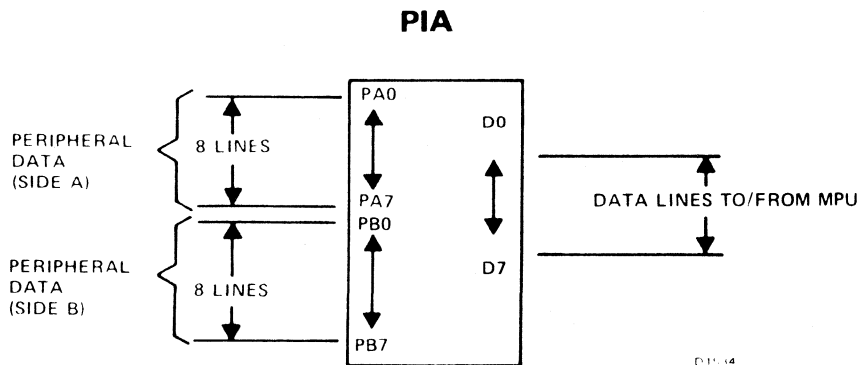
TR1217A-2

PIA

PERIPHERAL INTERFACE ADAPTER (PIA) – MC6821

The Peripheral Interface Adapter (PIA) is a means used to interface peripheral equipment with the microprocessing unit (MPU). The PIA communicates with the MPU via an eight-bit bidirectional data bus, three chip selects, two register selects, two interrupt request lines, one read/write line, an enable line, and a reset line. These will be discussed in detail later.

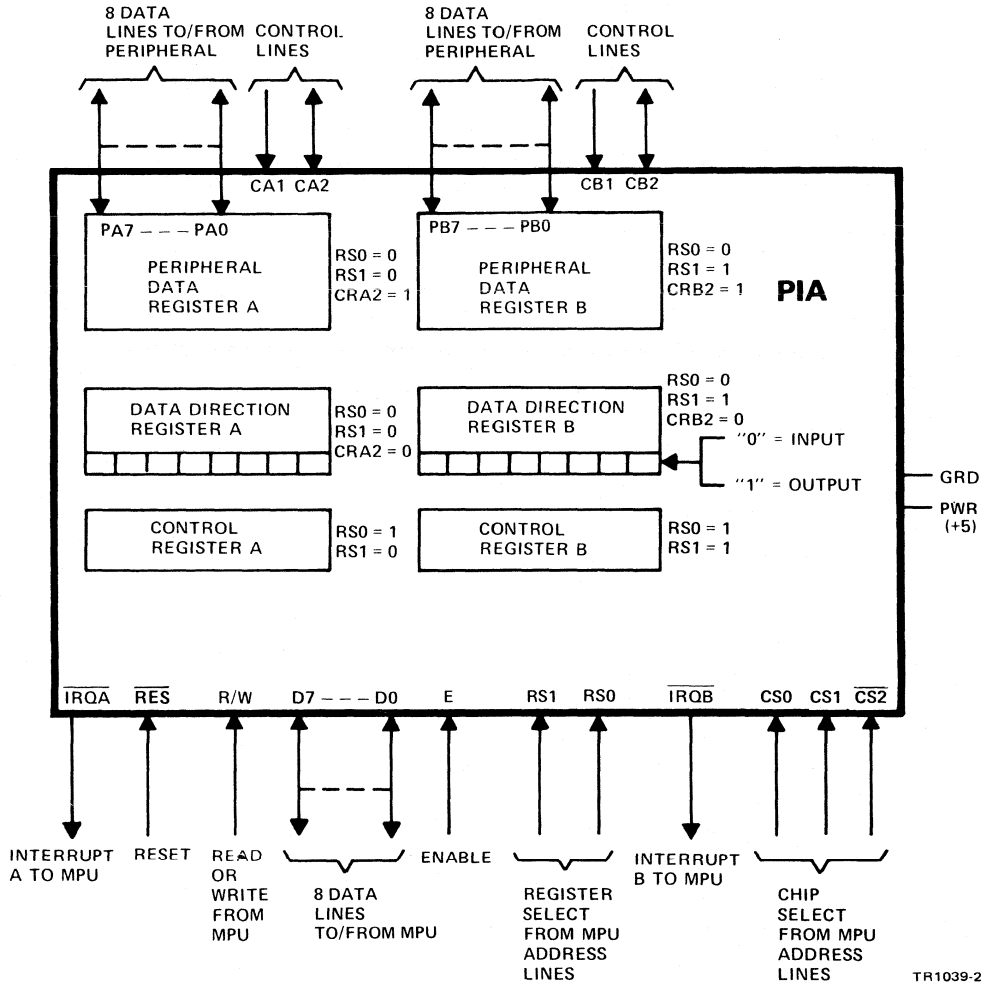
Each PIA has two 8-bit bidirectional peripheral data buses for interfacing with peripheral equipment as shown in Figure 1.



Each peripheral data line may be programmed to act as an input or an output. In addition to the two 8-bit peripheral data buses, peripheral control lines CA2 and CB2 may be programmed to act as a peripheral data line as will be discussed later.

Each PIA consists of two control registers, two data direction registers, and two peripheral interface registers (peripheral data). The control registers and the data direction registers are used to control the data in and out of the PIA.

PIA-2 Peripheral Interface Adapter (MC6821)



A. PERIPHERAL DATA LINES PA0 through PA7

Each of these 8 data lines which interface with the outside world can be programmed to act as either an input or an output. This is accomplished by setting a "1" in the corresponding bit in the Data Direction Register (DDR) if the line is to be an output—or a "0" in the DDR if the line is to be an input. When the data in the peripheral data lines are read into the MPU by a load instruction, those lines which have been designated as input lines (0 in DDR) will be gated directly to the data bus and into the register selected in the MPU. In the input mode, each line represents a maximum of two standard TTL loads for input high current and 1½ standard TTL loads for a low.

On the other hand, when an output data instruction (STA A PIA) is executed, data will be transferred via the data bus to the peripheral data register. A "1" output will cause a "high" on the corresponding data line and a "0" output will cause a "low" on the corresponding data line (two TTL load drive). Data in Peripheral Register A that have been programmed as outputs may be read by an MPU "LDA A from PIA" instruction. If the voltage is above 2 volts for a logic "1" or below 0.8 volts for a logic "0", the data will agree with that data outputted. However, if these output lines have been loaded such that they do not meet the levels for logic "1", the data read back into the MPU may differ from the data stored in the PIA Peripheral Register A.

B. PERIPHERAL DATA LINES PB₀ through PB₇

The 8 data lines which interface with the outside world on the B side may also be programmed to act either as an input or as an output. This is also accomplished by setting a "1" in the corresponding bit in the Data Direction Register (DDR) if the line is to be an output—or a "0" in the DDR if the line is to be an input. The output buffers driving these lines have three-state capability, allowing them to enter a high-impedance state when the peripheral data line is used as an input. Data in Peripheral Register B that have been programmed as outputs may be read by an MPU "LDA A from PIA" instruction even though the lines have been programmed as outputs. If the line has been programmed as an output and a logic "1", reading the line will indicate a logic "1" regardless of the voltage on the pin, due to buffering between the register and the output pin.

C. DATA LINES (D₀ through D₇)

The 8 bidirectional data lines permit transfer of data to/from the PIA and the MPU. The MPU receives data from the outside world from the PIA via these 8 data lines, or sends data to the outside world through the PIAs via the 8 data lines. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs a PIA read operation.

D. CHIP SELECT LINES (CS₀, CS₁, $\overline{\text{CS}}_2$)

These are the lines which are tied to the address lines of the MPU. It is through these lines that a particular PIA is selected (addressed). For selection of a PIA, the CS₀ and CS₁ lines must be high and the $\overline{\text{CS}}_2$ must be low. After the chip selects have been addressed, they must be held in that state for the duration of the E (enable) pulse, which is the only timing signal supplied by the MPU to the PIA. This enable pulse (E) is normally the ϕ_2 clock. One of the address lines should be ANDed with the VMA line with this output tied to a chip select.

PIA-4 Peripheral Interface Adapter (MC6821)

E. ENABLE LINE (E)

The enable pulse (E) is the only timing signal that is supplied to the PIA by the MPU. Timing on all other signals is referenced to the leading or trailing edges of the E pulse.

F. RESET LINE (RS)

This line is used to reset all registers in the PIA to a logical zero. This would be used primarily during a reset or power-on operation. This line is normally in the high state. A low level resets all registers in the PIA.

G. READ/WRITE LINE (R/W)

This signal is generated by the MPU to control the direction of the data transfers on the data bus. A low state on the PIA Read/Write line enables the input buffers and data is transferred from the MPU to the PIA (MPU write) on the falling edge of the E (ϕ_2) signal if the device has been selected. A high on the Read/Write line sets up the PIA for a transfer of data to the data bus (MPU read). The PIA output buffers are enabled when the proper address and the enable pulse are present, thus transferring data to the MPU.

H. INTERRUPT REQUEST LINES ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$)

These lines are used to interrupt the MPU either directly or indirectly through interrupt priority circuitry. These lines are "open source" (no load device on the chip) and are capable of sinking a current of 1.6 mA from an external source. This permits all interrupt request lines to be tied together in a "wire OR" configuration. Interrupts are serviced by a software routine that sequentially reads and tests, on a prioritized basis, the two control registers in each PIA for interrupt flag bits (Bit 6 and 7) that are set. Discussion on the control registers and how the flag bits get set will follow. When the MPU reads the Peripheral Data Register, *the Interrupt Flags (Bit 6 and Bit 7) are cleared and the Interrupt Request is cleared.*

These request lines ($\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$) are active low.

I. INTERRUPT INPUT LINES (CA1 and CB1)

These lines are input only to the PIA and set the interrupt flag (Bit 7) of the control registers in the PIA. Discussion of these lines in conjunction with the control register will follow.

J. PERIPHERAL CONTROL LINE (CA2)

This line can be programmed to act either as an interrupt input or as a peripheral output. As an output, this line is compatible with standard TTL, (2 load drive) and as an input represents two standard TTL loads for input high current and 1½ standard TTL loads for a low. The function of this line is programmed with Control Register A (Bits 3, 4, and 5).

K. PERIPHERAL CONTROL LINE (CB2)

This line may also be programmed to act as an interrupt input or as a peripheral output. As an input, this line has greater than 1 megohm input impedance and is compatible with standard TTL. As an output, it is compatible with standard TTL (2 load drive) and may also be used as a source of up to 1 milliamp at 1.5 volts to directly drive the base of a transistor switch. The function of this line is programmed with Control Register B (Bits 3, 4, and 5).

CONTROL REGISTER A (CRA)





7	6	5	4	3	2	1	0
IRQA1	IRQA2	CA2 Control			DDRA	CA1 Control	

CA 1 Control (Bits 0 and 1)

Peripheral control line CA1 is an input-only line which may be used to cause an interrupt by setting the interrupt flag IRQA1 (Bit 7) of Control Register A (CRA). Bits 0 and 1 of CRA are used to determine how the interrupt is to be handled.

After the MPU reads Peripheral Data Register A, the IRQA1 and IRQA2 (Bits 6 and 7) will be cleared.

PIA-6 Peripheral Interface Adapter (MC6821)

Transition of Interrupt Input Line CA1	Status of Bit 1 in CRA (Edge)	Status of Bit 0 in CRA (Mask)	IRQA1 (Interrupt Flag) Bit 7 of CRA	Status of IRQA Line (MPU Interrupt Request)
	0	0	1	MASKED (Remains High)
	0	1	1	GOES LOW (Processor Interrupted)
	1	0	1	MASKED (Remains High)
	1	1	1	GOES LOW (Processor Interrupted)

All other combinations of CA1 transition and status of Bit 0 and Bit 1 will be ignored.

As shown in the above chart, Bit 1 is the EDGE PROGRAMMING BIT. A logic "0" in Bit 1 programs the Interrupt Flag Bit 7 (IRQA1) to respond to a negative transition (edge) on CA1.

Bit 0 of the control register is the CA1 interrupt MASK PROGRAMMING BIT. If Bit 0 has a logic "0", the setting of the Interrupt Flag Bit 7 (IRQA1) will not allow the interrupt pin IRQA to go low. If Bit 0 contains a logic "1", the IRQA pin will go low when the flag bit 7 goes to a 1.

Data Direction Access Control (DDRA) (Bit 2)

This bit, in conjunction with the register select lines (RS0 and RS1) is used to select either the Peripheral Data Register or the Data Direction Register. To address the A side control register, RS1 is set to a logic "0" and RS0 is set to a logic "1".

RS1	RS0	CRA (Bit 2)	Register Selected
0	0	1	Peripheral Data Register A
0	0	0	Data Direction Register A
0	1	X	Control Register A





CONTROL REGISTER B (CRB)

7	6	5	4	3	2	1	0
IRQB1	IRQB2	CB2 Control			DDRB	CB1 Control	

CB1 Control (Bits 0 and 1)

Peripheral control line CB1 is an input-only line which may be used to cause an interrupt by setting the interrupt flag IRQB1 (Bit 7) of control register B (CRB). Bits 0 and 1 of CRB are used to determine how the interrupt is to be handled.

After the MPU reads Peripheral Data Register B, the IRQB1 (Bit 7) and IRQB2 (Bit 6) will be cleared.

Transition of Interrupt Input Line CB1	Status of Bit 1 in CRB (Edge)	Status of Bit 0 in CRB (Mask)	IRQB1 (Interrupt Flag) Bit 7 of CRB	Status of IRQB Line (MPU Interrupt Request)
	0	0	1	MASKED (Remains High)
	0	1	1	GOES LOW (Processor Interrupted)
	1	0	1	MASKED (Remains High)
	1	1	1	GOES LOW (Processor Interrupted)

All other combinations of CB1 transition and status of Bit 0 and Bit 1 will be ignored.

Bits 1 and 0 of control register B have the same programming use and logic as Bits 1 and 0 of the control register A, that is, Bit 1 is the Edge Programming Bit for CB1 and Bit 0 is the Interrupt Flag Mask Bit for CB1.

Data Direction Access Control (DDRB) (Bit 2)


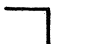


This bit, in conjunction with the register select lines (RS0 and RS1), is used to select either the Peripheral Data Register or the Data Direction Register. To address the B side control register, RS1 is set to a logic "1" and RS0 is set to a logic "1".

RS1	RS0	CRB (Bit 2)	Register Selected
1	0	1	Peripheral Data Register B
1	0	0	Data Direction Register B
1	1	X	Control Register B

PIA-8 Peripheral Interface Adapter (MC6821)

CA2 Control (Bits 3, 4, and 5 of CRA) as an Interrupt Input

Bits 3, 4, and 5 of the control register determine the function of this line.

Transition of Input CA2	Status of Bit 5 in CRA (I/O Control)	Status of Bit 4 in CRA (Edge)	Status of Bit 3 in CRA (Mask)	IRQA2 (Interrupt Flag) Bit 6 of CRA	Status of <u>IRQA</u> Line (MPU Interrupt Request)
	0	0	0	1	MASKED (Remains High)
	0	0	1	1	GOES LOW (Processor Interrupted)
	0	1	0	1	MASKED (Remains High)
	0	1	1	1	GOES LOW (Processor Interrupted)

All other combinations of CA2 transition and status of Bit 3 and Bit 4 will be ignored.

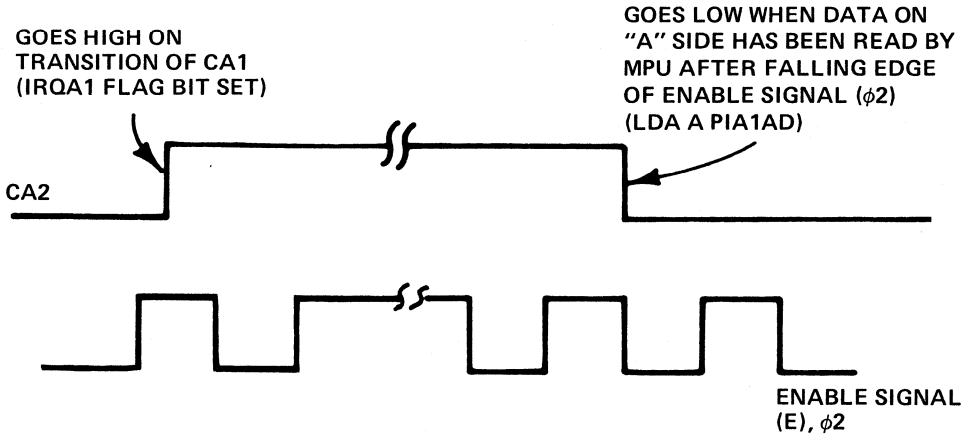
Bit 5 of the control register A is the Input/Output Programming Bit for CA2. If Bit 5 contains a logic "0", CA2 is programmed as an interrupt input line. When programmed as an interrupt input line, the programming of Bits 4 and 3 have the same usage as Bits 1 and 0. Bit 3 is the Interrupt Mask Bit for CA2.

When Bit 5 is a logic "1", CA2 is programmed as an output and Bits 4 and 3 are used to program one of the following three modes of operation. (1) Handshake Mode, (2) Pulse Mode, or (3) Bit 3 Following Mode. These three modes are detailed as follows.

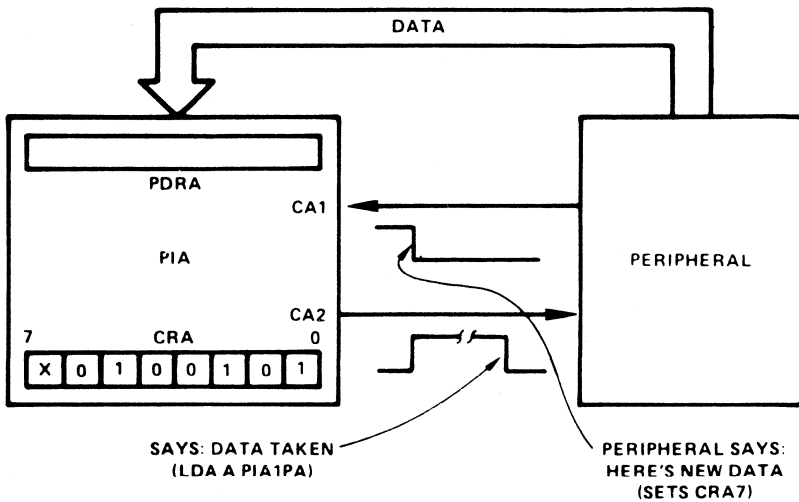
CA2 Used as an Output

If Bit 5 of CRA is set to a logic "1", CA2 is designated as an output. The four options utilizing CA2 as an output are shown below. In all four options the IRQA2 flag (Bit 6 of CRA) remains clear.

Bits 5, 4, 3 of CRA = 100 (Handshake Mode)



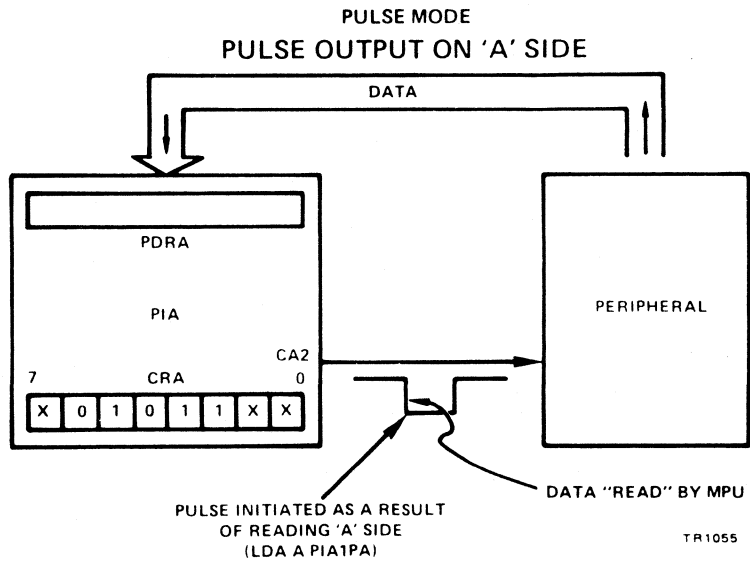
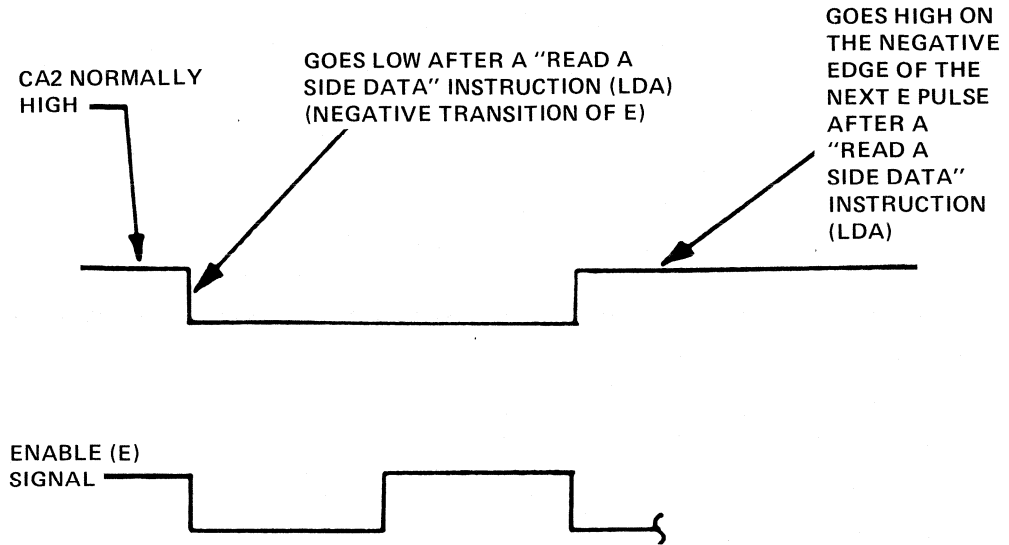
HANDSHAKING WITH PERIPHERAL ON 'A' SIDE



TR1053

PIA-10 Peripheral Interface Adapter (MC6821)

Bits 5, 4, 3 of CRA = 101 (Pulse Mode)



Bits 5, 4, 3 of CRA = 110 (Bit 3 Following Mode)

CA2 will always be low.





Bits 5, 4, 3, of CRA = 111

CA2 will always be high.

b ₅	b ₄	b ₃	CA2
1	1	0	0
1	1	1	1

CB2 Control (Bits 3, 4, and 5 of CRB) as an Interrupt Input

Bits 3, 4, and 5 of the control register determine the function of this line.

Transition of Input CB2	Status of Bit 5 in CRB (I/O Control)	Status of Bit 4 in CRB (Edge)	Status of Bit 3 in CRB (Mask)	IROB2 (Interrupt Flag) Bit 6 of CRB	Status of IROB Line (MPU Interrupt Request)
	0	0	0	1	MASKED (Remains High)
	0	0	1	1	GOES LOW (Processor Interrupted)
	0	1	0	1	MASKED (Remains High)
	0	1	1	1	GOES LOW (Processor Interrupted)

All other combinations of CB2 transition and status of Bit 3 and Bit 4 will be ignored.

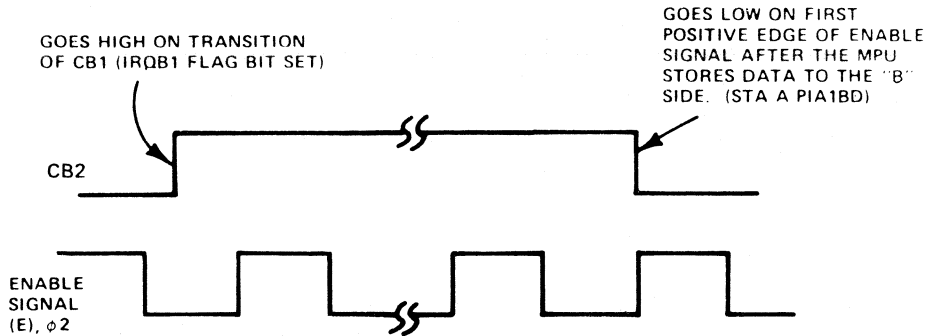
The programming of Bits 3, 4, and 5 in control register B has the same use as Bits 3, 4, and 5 in control register A. Control register B programs CB1 and CB2 while control register A programs CA1 and CA2.

PIA-12 Peripheral Interface Adapter (MC6821)

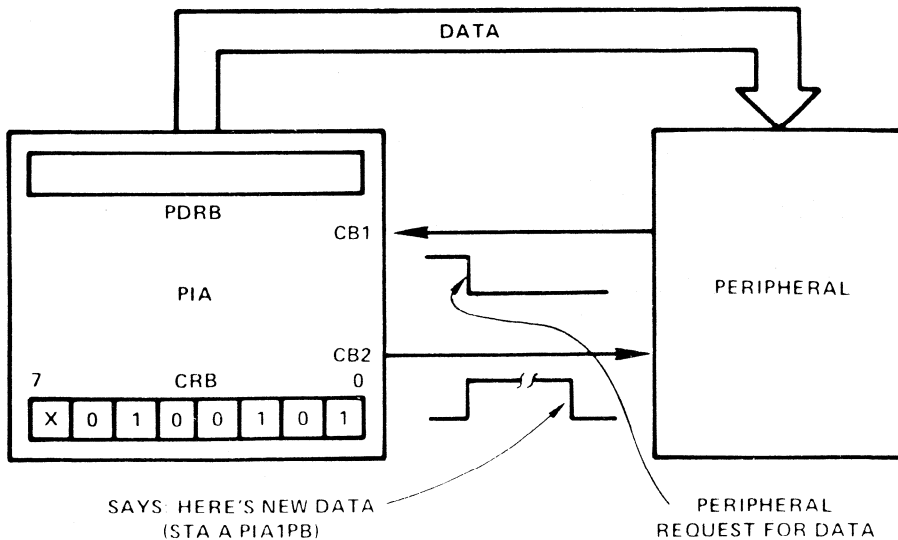
CB2 Used as an Output

If Bit 5 of CRB is set to a logic "1", CB2 is designated as an output. The four options utilizing CB2 as an output are shown below. In all four options, the IRQB2 flag (Bit 6 of CRB) remains clear and the IRQB interrupt request line remains high.

Bits 5, 4, 3 of CRB = 100 (Handshake Mode)

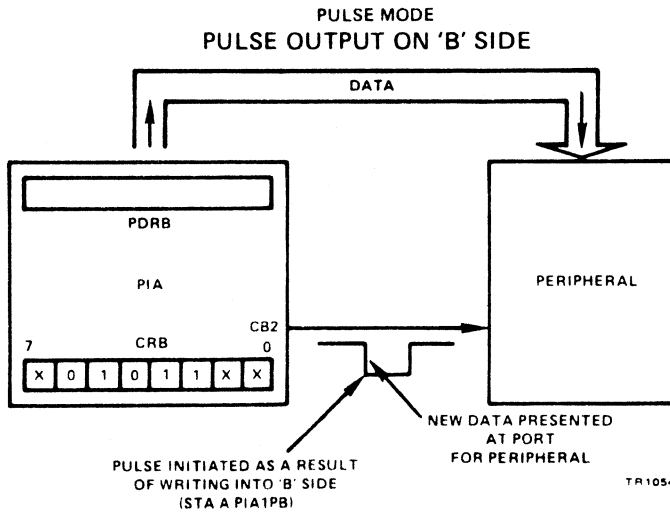
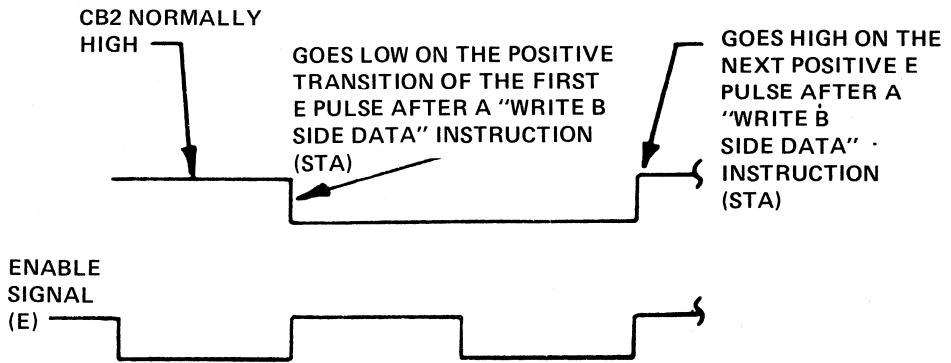


HANDSHAKING WITH PERIPHERAL ON 'B' SIDE



TR1052

Bits 5, 4, 3 of CRB = 101 (Pulse Mode)



PIA—14 Peripheral Interface Adapter (MC6821)

Bits 5, 4, 3 of CRB = 110 (Bit 3 Following Mode)

CB2 will always be low.

Bits 5, 4, 3 of CRB = 111

CB2 will always be high.

b ₅	b ₄	b ₃	CB2
1	1	0	0
1	1	1	1

SUMMARY OF PIA CONTROL REGISTERS

A. REGISTER SELECTS RS₀ AND RS₁

If RS₁ is set to a logic "0", then "A" side is selected.

If RS₁ is set to a logic "1", then the "B" side is selected.

If RS₀ is set to a logic "0", and CRA (or CRB) Bit 2 is set to a logic "1", the peripheral data register is selected.

If RS₀ is set to a logic "0", and CRA (or CRB) Bit 2 is set to a logic "0", then the data direction register is selected.

If RS₀ is set to a logic "1", the control register is selected.

B. CA1 OR CB1 INTERRUPT LINE

If Bit 0 of CRA (or CRB) is set to a logic "0", all interrupts caused by CA1 (or CB1) are disallowed by the PIA. However, the respective flag bits will be set by CA1 and/or CB1.

C. CA2 OR CB2 INTERRUPT LINE

If Bit 3 of CRA (or CRB) is set to a logic "0", all interrupts caused by CA2 (or CB2) are disallowed by the PIA. If Bit 5 of CRA (or CRB) is set to a logic "1", then the CA2 (or CB2) line is used as an output line per previous table and the respective flag bits CRA₆ (CRB₆) are reset to a "0".

D. The interrupt flag bits (6, 7 of both the A and B control registers) are read only bits. The MPU *cannot* write into Bits 6 and 7 of either control register. Only interrupt inputs from the “outside world” can set Bits 6 or 7 of the control registers. The MPU can reset the flag bits by reading the respective peripheral data registers. When the read peripheral data register is performed, both flag bits (6 and 7) are cleared.

SUMMARY OF CONTROL REGISTERS CRA AND CRB

Control Registers CRA and CRB have total control of CA1, CA2, CB1, and CB2 lines. The status of eight bits of the control registers may be read into the MPU. However, the MPU can only write into Bit 0 through Bit 5 (6 bits), since Bit 6 and Bit 7 are set only by CA1, CA2, CB1, or CB2.

ADDRESSING PIAs

Before addressing PIAs, the Data Direction (DDR) must first be loaded with the bit pattern that defines how each line is to function, i.e., as an input or an output. A logic “1” in the Data Direction Register defines the corresponding line as an output while a logic “0” defines the corresponding line as an input. Since the DDR and the Peripheral Data Lines have the same address, the control register bit 2 determines which register is being addressed. If Bit 2 in the control register is a logic “0”, then the DDR is addressed. If Bit 2 in the control register is a logic “1”, the Peripheral Data Register is addressed. Therefore, it is essential that the DDR be loaded first before setting Bit 2 of the control register.

The above sequence of setting up the PIA assumes that the data outputs of the PIA are active high (True \geq 2.4 volts). If all the outputs at a given PIA port are active low (True \leq 0.4 volts), see the section on ACTIVE LOW OUTPUTS.

EXAMPLE

Given a PIA with an address of 4004, 4005, 4006, and 4007. 4004 is the address of the A side Peripheral Interface Register. 4005 is the address of the A side control register. 4006 is the address of the B side Peripheral Interface Register. 4007 is the address of the B side control register. On the A side, Bits 0, 1, 2, and 3 will be defined as inputs, while Bits 4, 5, 6, and 7 will be used as outputs. On the B side, all lines will be used as outputs.

PIA—16 Peripheral Interface Adapter (MC6821)

The program to accomplish the above is as follows.

```
PIA1AD = 4004
PIA1AC = 4005
PIA1BD = 4006
PIA1BC = 4007

1. LDA A #%11110000      (4 outputs, 4 inputs)
2. STA A PIA1AD          (Loads A DDR)
3. LDA A #%11111111      (All outputs)
4. STA A PIA1BD          (Loads B DDR)
5. LDA A #%0000100       (Sets Bit 2)
6. STA A PIA1AC          (Bit 2 set in A control register)
7. STA A PIA1BC          (Bit 2 set in B control register)
```

Statement 2 addresses the DDR, since the control register (Bit 2) has not been loaded. Statements 6 and 7 load the control registers with Bit 2 set, so addressing PIA1AD or PIA1BD accesses the Data Register.

PIA PROGRAMMING VIA THE INDEX REGISTER

The program shown in the previous section can be accomplished using the Index Register.

```
1. LDX #$F004
2. STX PIA1AD           $F0 → PIA1AD; $F0 → PIA1AC
3. LDX #$FF04
4. STX PIA1BD           $FF → PIA1BD; $04 → PIA1BC
```

Using the Index Register in this example has saved six bytes of program memory as compared to the program shown in the previous section.

ACTIVE LOW OUTPUTS

When all the outputs of given PIA port are to be active low (True \leq 0.4 volts), the following procedure should be used.

- Set Bit 2 in the control register.
- Store all 1s (\$FF) in the peripheral data register.
- Clear Bit 2 in the control register.
- Store all 1s (\$FF) in the data direction register.
- Store control word (Bit 2 = 1) in control register.

EXAMPLE

The B side of PIA1 is set up to have all active low outputs. CB1 and CB2 are set up to allow interrupts in the HANDSHAKE MODE and CB1 will respond to positive edges (low-to-high transitions). Assume reset conditions. Addresses are set up and equated to the same labels as previous example.

1. LDA A #4
2. STA A PIA1BC Set Bit 2 in PIA1BC (Control Register)
3. LDA B #\$FF
4. STA B PIA1BD All 1s in Peripheral Data Register
5. CLR PIA1BC Clear Bit 2
6. STA B PIA1BD All 1s in Data Direction Register
7. LDA A #\$27
8. STA A PIA1BC 00100111 →→ Control Register

The above procedure is required in order to avoid outputs going low, to the active low TRUE STATE, when all 1s are stored to the Data Direction Register as would be the case if the normal configuration procedure were followed.

INTERCHANGING RS0 AND RS1

Some system applications may require movement of 16 bits of data to or from the "outside world" via two PIA ports (A side + B side). When this is the case it is an advantage to interconnect RS1 and RS0 as follows.

RS0 to A1 (Address Line A1)
RS1 to A0 (Address Line A0)

This will place the peripheral data registers and control registers side by side in the memory map as follows.

Table	Example Address
PIA1AD	\$4004
PIA1BD	\$4005
PIA1AC	\$4006
PIA1BC	\$4007

The index register or stackpointer may be used to move the 16-bit data in two 8-bit bytes with one instruction. As an example:

LDX PIA1AD PIA1AD → IX_H; PIA1BD → IX_L

PIA—18 Peripheral Interface Adapter (MC6821)

PIA – AFTER RESET

When the RS (Reset Line) has been held low for a minimum of one microsecond, all registers in the PIA will be cleared.

Because of the reset conditions, the PIA has been defined as follows.

1. All I/O lines to the "outside world" have been defined as inputs.
2. CA1, CA2, CB1, and CB2 have been defined as interrupt input lines that are negative edge sensitive.
3. All the interrupts on the control lines are masked. Setting of interrupt flag bits *will not* cause \overline{IRQA} or \overline{IRQB} to go low.

SUMMARY OF CA1-CB1 PROGRAMMING

Bits 1 and 0 of the respective control registers are used to program the interrupt input control lines CA1 and CB1.

b ₁	b ₀	
0	0	b ₁ = Edge (0 = -, 1 = +)
0	1	b ₀ = Mask (0 = Mask, 1 = Allow)
1	0	
1	1	

Note that this is the same logic as Bits 4 and 3 for CA2-CB2 when CA2-CB2 are programmed as inputs.

SUMMARY OF CA2-CB2 PROGRAMMING

Bits 5, 4, and 3 of the control registers are used to program the operation of CA2-CB2.

	b ₅	b ₄	b ₃	
CA2-CB2 INPUT MODE	0	0(-)	0 (Mask)	CA2-CB2 Input Mode
	0	0(-)	1 (Allow)	b ₄ = Edge (0 = -, 1 = +)
	0	1(+)	0 (Mask)	b ₃ = Mask (0 = Mask, 1 = Allow)
	0	1(+)	1 (Allow)	
CA2-CB2 OUTPUT MODE	1	0	0	0 – HANDSHAKE MODE
	1	0	1	1 – PULSE MODE
	1	1	0	
	1	1	1	b ₃ FOLLOWING MODE

PIA CONFIGURATION PROBLEM

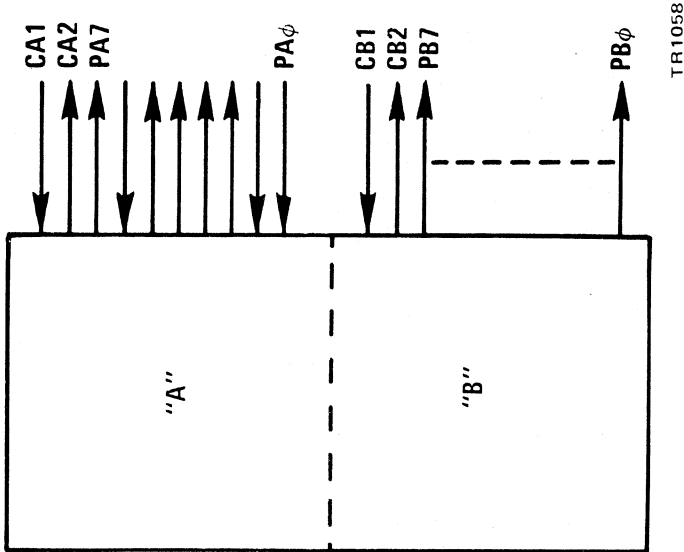
I/O AS FOLLOWS:

CONTROL LINES:

- CA1 – POSITIVE EDGE, ALLOW INTERRUPT
- CA2 – PULSE MODE
- CB1 – NEGATIVE EDGE, MASK INTERRUPT
- CB2 – HAND SHAKE MODE

ASSUME RESET CONDITION

- PIA1AD
- PIA1AC
- PIA1BD
- PIA1BC



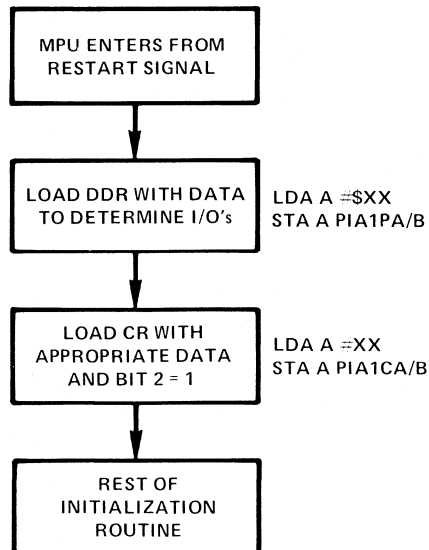
TR1058

PIA CONFIGURATION SOLUTION

```
LDA A #$B0    10111100
STA A PIA1AD  I/O TO DDRA
LDA A #$FF    1111 1111
STA A PIA1BD  I/O TO DDRB
LDA A #$2F    0010 1111
STA A PIA1AC  TO "A" CONTROL
LDA A #$24    0010 0100
STA A PIA1BC  TO "B" CONTROL
```

TR1059

INITIALIZING PIA ON POWER-UP



TR1061

For convenience the following table summarizes the improvements incorporated in the MC6821 vs. the MC6820.

Summary of Differences — 6820 vs. 6821

	6820	6821
RESET	Although RESET clears all the registers, if a negative edge comes in on CA1, CA2, CB1, and CB2, the appropriate flag bit can set even though the RESET pin is zero.	All the bits including the flag bits are held to zero.
Enable Input	<ol style="list-style-type: none"> 1. $C_{in} = 20$ pf max $V_{IH\ min} = V_{SS} + 2.4$ volts $V_{IL\ max} = V_{SS} + 0.4$ volts 2. Refreshes the data bus drivers drivers. 3. Clocks in interrupts 	<ol style="list-style-type: none"> 1. $C_{in} = 7.5$ pf max $V_{IH\ min} = V_{SS} + 2.0$ volts $V_{IL\ mzx} = V_{SS} + 0.8$ volts 2. No data bus driver refresh required. 3. Does not clock in interrupts, however one E pulse is required while the input is at the inactive level prior to receiving the active transition.
Output Drive Capability (A and B side and IRQA and IRQB)	1 TTL load	2 TTL load
PD Max	650 mw	550 mw

TR1231

ACIA

SERIAL DATA COMMUNICATIONS

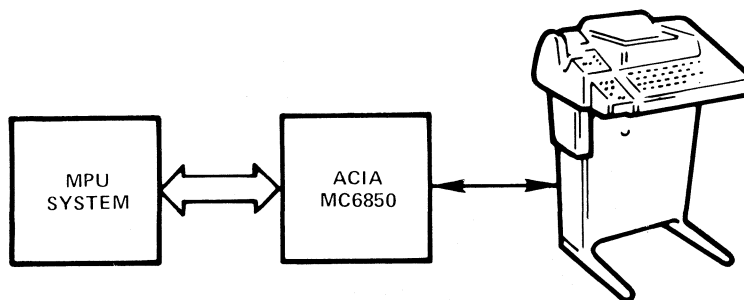
As has been shown in previous sections of this manual, the M6800 MPU works with 8 bits per byte and the bytes are moved in a parallel format, i.e., all bits of a given byte move at the same time.

There are times when data must move over a communications channel or media that is expensive per unit length. When this is the case, it is desirable to trade off communications channel cost for time. Rather than have 8 channels provided in order to move data in a parallel format, it is far less expensive to provide a single communications channel and move the data in a serial format. When data is moved in a serial format, it takes more time to send the byte for a given communications rate or channel bandwidth, but a lesser number of channels are required.

The very simplest example would be where a system on the ground floor of a building must send data to a system on the fifth floor. It may be less expensive (less cabling to the fifth floor) and will be more reliable if a single channel is used for the communications.

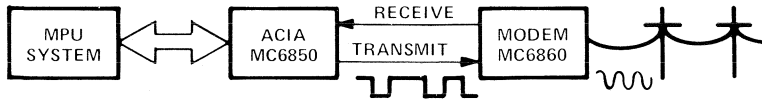
The MC6850 Asynchronous Communications Interface Adapter provides the M6800 MPU-based systems designer with a means of implementing serial communications. The main function of the ACIA is to provide a hardware means of parallel-to-serial and serial-to-parallel data conversion and communications control via software programming.

MPU TO TTY INTERFACE



DR1155

MPU TO REMOTE SITE VIA MODEM



TR1156

Asynchronous Communications Interface Adapter (ACIA) —MC6850

The Asynchronous Communications Interface Adapter (ACIA) is a means used to receive and transmit up to eight bits of data for serial data communications. The ACIA communicates with the MPU via an eight-bit bidirectional data bus, three chip select lines, one register select line, one interrupt request line, an enable line, and one read/write line.

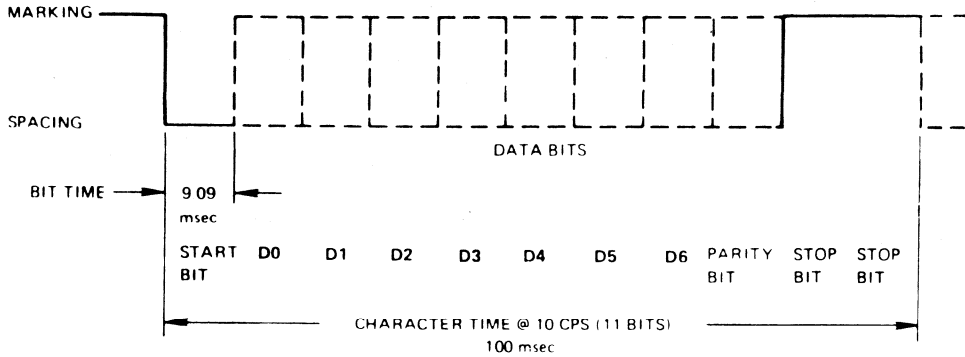
The ACIA has four registers which may be addressed by the MPU. The Status Register (SR) and the Receiver Data Register (RDR) are "read only" registers in that the MPU cannot write into two registers. The Transmit Data Register (TDR) and the Control Register (CR) are "write only" registers in that the MPU cannot read from these registers.

MPU Interface Lines

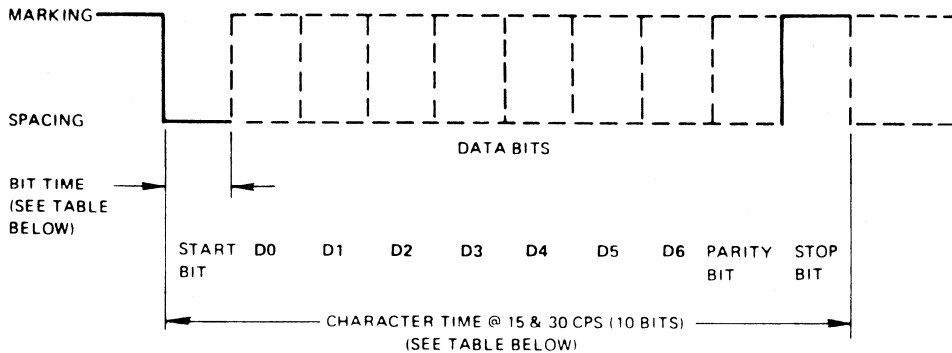
A. BIDIRECTIONAL DATA LINES (D0-D7)

The eight bidirectional data lines permit transfer of data to and from the ACIA and the MPU. The MPU receives data from the outside world from the ACIA via these eight data lines or sends data to the outside world through the ACIA via the eight data lines. The data bus output drivers are three-state devices that remain in the high-impedance (off) state except when the MPU performs an ACIA read operation.

**110 BAUD
SERIAL ASCII DATA TIMING**



TR1159



BAUD RATE	150	300
CHARACTERS/SEC	15	30
BIT TIME (msec)	6.67	3.33
CHARACTER TIME (msec)	66.7	33.3

$$\text{BIT TIME} = \frac{\text{SEC}}{\text{BAUD RATE}}$$

150 & 300 BAUD SERIAL ASCII DATA TIMING

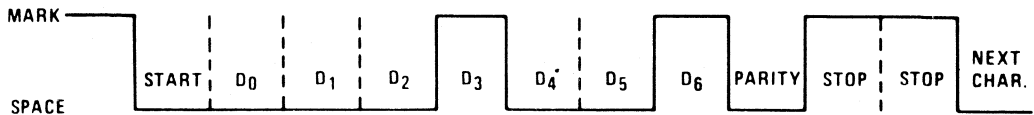
TR1160

ASCII CODE

BITS 4 THRU 6		0	1	2	3	4	5	6	7
BITS 0 THRU 3	0	NUL	DLE	SP	0	@	P		'p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	\$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	'	7	G	W	g	w
	8	BS	CAN	(8	H	X	h	x
	9	HT	EM)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[k	}
	C	FF	FS	,	<	L	/	l	/
	D	CR	GS	-	=	M]	m	}
	E	SO	RS	.	>	N	↑	n	≈
	F	SI	US	/	?	O	←	o	DEL

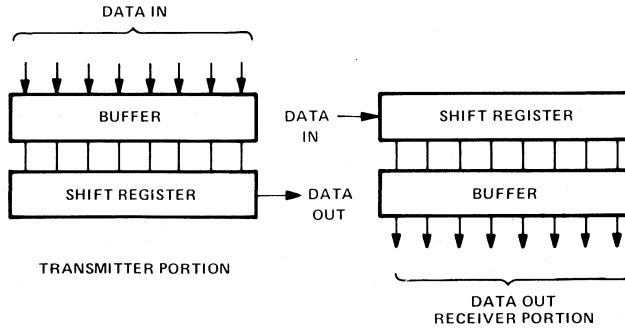
TR1097

SEND A 7 BIT ASCII CHAR. "H"
EVEN PARITY — 2 STOP BITS
 $H = 48_{16} = 1001000_2$



TR1161

**PARALLEL TO SERIAL CONVERTER
AND
SERIAL TO PARALLEL CONVERTER**



TR1162

B. CHIP SELECT LINES (CS0, CS1, CS2)

These are the lines which are tied to the address lines of the MPU. It is through these lines that a particular ACIA is selected (addressed). For selection of an ACIA, the CS0 and CS1 lines must be high and the CS2 must be low. After the chip selects have been addressed, they must be held in that state for the duration of the E enable pulse, which is the only timing signal supplied by the MPU to the ACIA.

C. ENABLE SIGNAL (E)

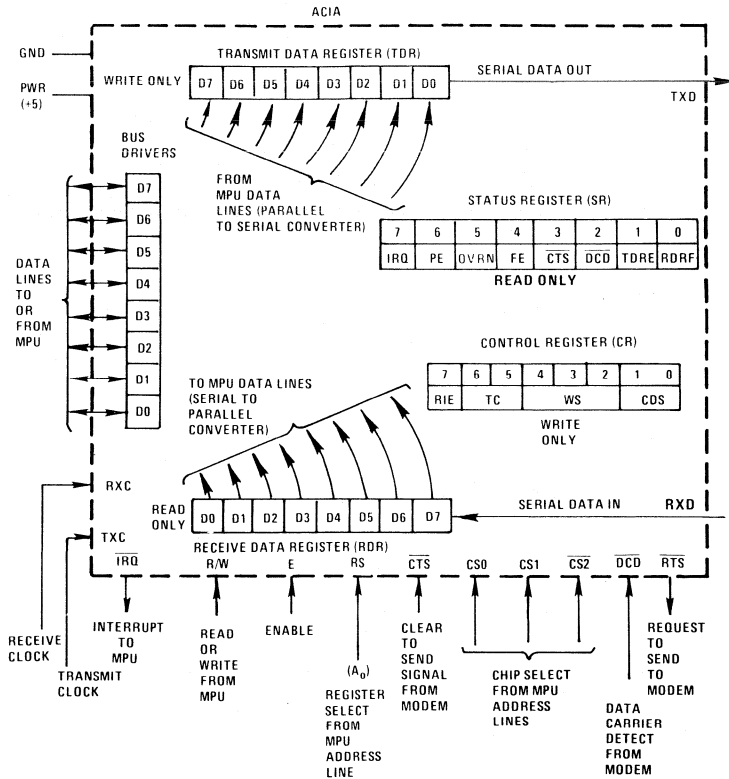
The enable pulse is a high-impedance, TTL-compatible input from the MPU that enables the ACIA input or output buffers, and clocks data to or from the ACIA.

D. READ/WRITE LINE (R/W)

The Read/Write line is a high-impedance, TTL-compatible input that is used to control the direction of data flow between the ACIA's eight-bit parallel data bus and the MPU. When Read/Write is high (MPU read), the ACIA output driver is turned on and a selected register is read by the MPU. When the Read/Write line is low (MPU write), the ACIA output driver is turned off and the MPU writes into a selected register. Thus, the Read/Write signal, in conjunction with the register select line, is used to select the registers within the ACIA that are read only.

Register Select (RS)	Read/Write (R/W)	ACIA Register Selected	MPU Read or Write
0	0	Control	Write
0	1	Status	Read
1	0	Transmit Data	Write
1	1	Receive Data	Read

ACIA-6 Serial Data Communications



TR1163

E. REGISTER SELECT (RS)

The Register Select line is a high-impedance, TTL-compatible input from the MPU that is used to select, in conjunction with the Read/Write line, either the Transmit/Receiver Data register or the Control/Status register in the ACIA as shown in paragraph D of this section.

F. INTERRUPT REQUEST LINE ($\overline{\text{IRQ}}$)

The Interrupt Request Line is a TTL-compatible output line to the MPU that is used to interrupt the MPU upon the occurrence of certain events. This line is active in the low state and remains low as long as the interrupt is present and the appropriate interrupt enable within the ACIA is set.

ACIA Registers

A. STATUS REGISTER (READ ONLY)

The Status Register can *only* be read by the MPU. This register is selected when the Register Select (RS) line is low and the Read/Write (R/W) line is high ($\text{RS} \cdot \text{R/W} = 01$).

STATUS REGISTERS (SR)

7	6	5	4	3	2	1	ϕ
IRQ	PE	OVRN	FE	$\overline{\text{CTS}}$	$\overline{\text{DCD}}$	TDRE	RDRF

Bit 0 – Receiver Data Register Full (RDRF)

“1” – The Receiver Data Register is full. This bit is cleared when the RDR is read by the MPU.

“0” – The Receiver Data Register has been read by the MPU. The non-destructive read cycle clears the RDRF bit, although the data in the Receiver Data Register is retained. If the $\overline{\text{DCD}}$ line goes high indicating loss of carrier, the RDRF bit is clamped at logic “0” indicating the contents of the Receiver Data Register are not current.

Bit 1 – Transmit Data Register Empty (TDRE)

“1” – The Transmit Data Register is empty and new data may be transferred. This bit is cleared by a write from the MPU to the TDR.
– IRQ (bit 7 gets set).

“0” – The Transmit Data Register is full.

When a logic “1” is present on the $\overline{\text{CTS}}$ pin a 1 will be present in bit 3 of the Status Register indicating it is not clear to Send. This condition will clamp bit 1 of the Status Register (TDRE) to a logic “0” and inhibit interrupts due to a Transmit Register Empty collection. See Bit 7 – IRQ.

Bit 2 – Data Carrier Detect (DCD)

“1” – There is no carrier from the modem. This also clamps bit 0 (RDRF) to a logic “0”, thus inhibiting further interrupts from RDRF. See Bit 7-IRQ.

“0” – The carrier from the modem is present.

Bit 3 – Clear to Send ($\overline{\text{CTS}}$)

“1” – The Clear to Send line from the modem is high, thus inhibiting the Transmit Data Register Empty (TDRE) bit. Modem is not ready for data.

“0” – The Clear to Send line from the modem is low. Modem is ready for data.

Bit 4 – Framing Error (FE)

“1” – Framing Error indicates that the received character is improperly framed by the start and stop bit and is detected by the absence of the first or second stop bit. This error indicates a synchronization error, faulty transmission, or a break condition. This error flag is set or reset during the receiver data transfer time. Therefore, this error indicator is present throughout the time that the associated character is available.

“0” – The received character is properly framed.

Bit 5 – Receiver Overrun (OVRN)

“1” – Overrun is an error flag that indicates that one or more characters in the data stream were lost. That is, a character or a number of characters were received but not read from the Receiver Data Register (RDR) prior to subsequently being received. The overrun condition begins at the midpoint of the last bit of the second character received in succession without a read of the RDR having occurred. The Overrun does not occur in the Status Register until the valid character prior to Overrun has been read. Character synchronization is maintained during the Overrun condition. The Overrun indication is reset after the reading of data from the Receive Data Register. Overrun is also reset by the Master Reset.

“0” – No Receiver Data Overruns have occurred.

Bit 6 – Parity Error (PE)

“1” – The parity error flag indicates that the number of highs (ones) in the character does not agree with the preselected odd or even parity. Odd parity is defined to be when the total number of ones is odd. The parity error indication will be present as long as the data character is in the RDR. If no parity is selected, then both the transmitter parity generator output and the receiver parity check results are inhibited.

“0” – No parity error occurred.

Bit 7 – Interrupt Request (IRQ)

“1” – There is an interrupt in the ACIA. This bit being high causes the IRQ output line to be low. This will be cleared by reading the Status Register and writing into the Transmit Data Register or reading the Receiving Data Register.

Interrupts (IRQ=1) can also be caused by loss of receive carrier (DCD=1). The respective interrupts—*a)* Transmitter Data Register Empty, *b)* Receiver Data Register Full, *c)* Data Carrier Loss—will only occur if the respective interrupts are enabled, i.e., bit 7 of the Control Register set to a 1 for receive interrupts and Bit 6=0 and Bit 5=1 of the Control Register for the transmit interrupts.

“0” – No interrupt present.

B. CONTROL REGISTER (WRITE ONLY)

The Control Register can only be written into by the MPU. This register is selected when the Register Select (RS) line and the Read/Write line are both low (RS · R/W = 00).

CONTROL REGISTER (CR)

7	6	5	4	3	2	1	ϕ
R I E	Transmitter Control		Word Select			Counter Divide	

Receiver Interrupt Enable

Bits 0 and 1 – Counter Divide Select Bits (CDS)

CR1	CR0	Function	Maximum Data Clock Rate
0	0	÷1	500 KHz
0	1	÷16	800 KHz
1	0	÷64	800 KHz
1	1	Master Reset	

Bits 2, 3, and 4 – Word Select Bits (WS)

B4	B3	B2	Word Length	+	Parity	+	Stop Bits
0	0	0	7		Even		2
0	0	1	7		Odd		2
0	1	0	7		Even		1
0	1	1	7		Odd		1
1	0	0	8		None		2
1	0	1	8		None		1
1	1	0	8		Even		1
1	1	1	8		Odd		1

Bits CR5 and CR6 have the following system application:

CR6	CR5	
0	0	The $\overline{\text{RTS}}$ pin is <i>low</i> and Transmit Interrupts are inhibited. This is the code used when requesting that the communications channel be set up. It is not clear to send data yet.
0	1	The $\overline{\text{RTS}}$ pin is <i>low</i> and the communications channel has been set up. Therefore, this code is used to generate IRQs via the TDRE bit in the Status Register.
1	0	The $\overline{\text{RTS}}$ pin is <i>high</i> and transmit interrupts are inhibited. This code can be used to "knock down" the communications channel.
1	1	The $\overline{\text{RTS}}$ pin is <i>low</i> (keep up communications channel), a break signal (low level on transmit data out line) is transmitted. This is used to interrupt the remote system.

Bit 7 – Receiver Interrupt Enable (RIE)

"1" – Enables interrupts caused by:

- a) Receiver Data Register Full going high;
- b) A low to high transition on the Data Carrier Detect signal line.

"0" – Inhibits interrupts due to Receive Data Register Full or loss of Receive Data Carrier.

Clock Inputs

Separate high-impedance, TTL-compatible inputs are provided for clocking of transmitted and received data. Clock frequencies of 1, 16, or 64 times the data rate may be selected.

A. TRANSMIT CLOCK (TXC)

The transmit clock input is used for the clocking of transmitted data. The transmitter initiates data on the negative transition of the clock.

B. RECEIVE CLOCK (RXC)

The Receive Clock input is used for synchronization of received data. The receiver strobes the data on the positive transition of the clock. (In the $\div 1$ mode, the clock and data must be synchronized externally.)

Modem Control

The ACIA includes several functions that permit limited control of a data modem. The functions included are Clear-to-Send, Request-to-Send, and Data Carrier Detect.

A. CLEAR-TO-SEND ($\overline{\text{CTS}}$)

This high-impedance, TTL-compatible input provides automatic control of the transmitting end of a communications link via the modem's "clear-to-send" active low output.

B. REQUEST-TO-SEND ($\overline{\text{RTS}}$)

The Request to Send output enables the MPU to control a modem via the data bus. The active state is low.

C. DATA CARRIER DETECTED ($\overline{\text{DCD}}$)

This high-impedance, TTL-compatible input provides automatic control of the receiving end of a communication link by means of the modem "Data Carrier Detect" or "Received-Line-Signal Detect" output. The $\overline{\text{DCD}}$ input inhibits and initializes the receiver section of the ACIA when high. A low-to-high transition of the Data Carrier Detect may initiate an interrupt to the MPU to indicate the occurrence of a loss carrier.

Received Data Line (RX)

The Received Data Line is a high-impedance, TTL-compatible input through which data is received in a serial NRZ (Nonreturn to Zero) format. Synchronization with a clock for detection of data is accomplished internally when clock rates of 16 or 64 times the bit rate are used. Data rates are in the range of 0 to 500 Kbps when external synchronization is utilized.

Transmitted Data Lines (TX)

The Transmit Data Output Line transfers serial NRZ data to a modem or other peripheral at the same range of rates as the received data.

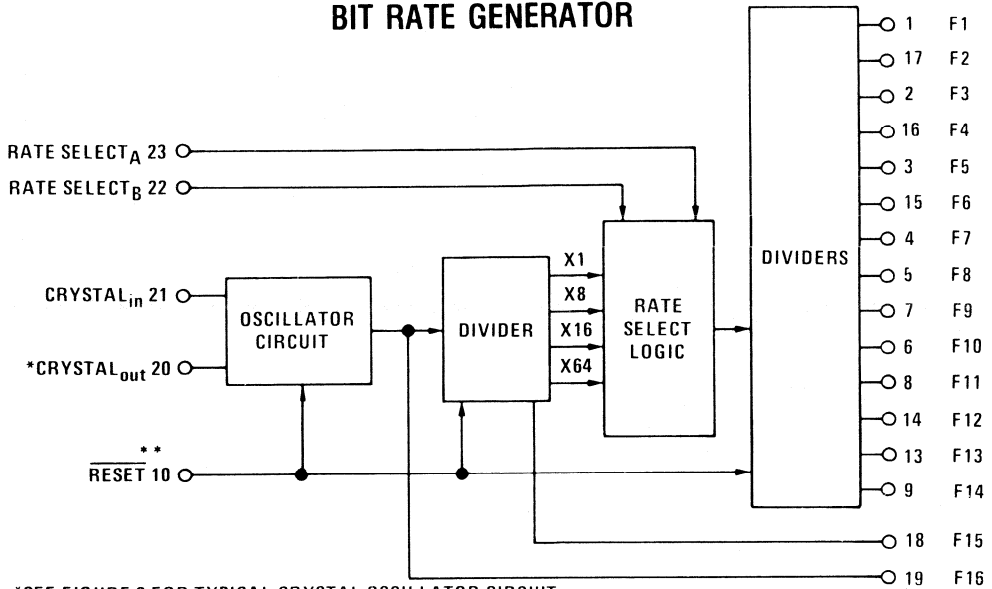
PROBLEM: SET UP ACIA, ÷64, 8B, OP, 1S. OPERATE WITH MODEM.
 WRITE THE CODE TO SET UP THE ACIA AND BE READY
 TO RECEIVE AND TRANSMIT DATA.

CONTROL REG = ACIAC
 STATUS REG = ACIAC
 RCVR REG = ACIAD
 XMIT REG = ACIAD

ANS: LDA A #S03 00000011 RESET
 STA A ACIAC
 LDA A #S1E 00011110 ÷64, 8B, OP, IS
 STA A ACIAC
 CHK LDA A ACIAC LOAD STATUS REG
 AND A #SOC 00001100 CHECK \overline{CTS} & \overline{DCD}
 BNE CHK
 LDA A #SBE 10111110 READY TO
 STA A ACIAC RECEIVE & XMIT

TR1176-1

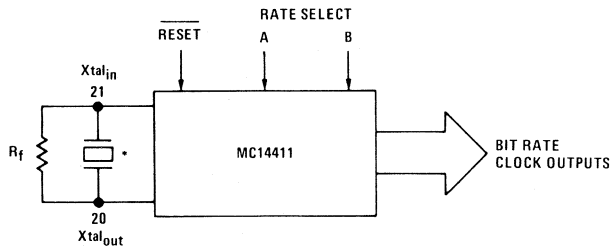
MC14411 BIT RATE GENERATOR



*SEE FIGURE 2 FOR TYPICAL CRYSTAL OSCILLATOR CIRCUIT.
 **OUTPUTS GO TO "1" LEVEL UPON RESET.

TR1181

MC14411 BIT RATE GENERATOR



$R_f = 15 \text{ M}\Omega \pm 10\%$

CRYSTAL SPECIFICATION

CRYSTAL MODE	PARALLEL
FREQUENCY	1.8432 MHz $\pm 0.05\%$ @ 13 pF
R_S	540 Ω MAX
C_0	7.0 pF MAX
TEMPERATURE RANGE	0 to 70°C

TR1182

Addressing Modes

72 MNEMONIC INSTRUCTIONS

6 ADDRESSING MODES

- INHERENT/ACCUMULATOR/IMPLIED
- IMMEDIATE
- DIRECT
- EXTENDED
- INDEXED
- RELATIVE

TR1063-1

ADDRESSING MODES

The MC6800 Microprocessor has six addressing modes available to the programmer. They are Inherent/Accumulator, Immediate, Direct, Extended, Indexed, and Relative.

A. INHERENT/ACCUMULATOR/IMPLIED

These addressing modes have one-byte instructions and therefore either do not require addressing a memory location or the addressing information is contained in the instruction. An example of an *inherent* instruction would be to execute a "clear carry bit" instruction and would look like this in memory:

Memory Location	Memory Contents (Hex)
0100	0C (CLC opcode)

0C (in hex) is the CLC instruction. The result of this instruction would be to load a zero in the carry bit in the MPU condition code register.

An example of an *accumulator* instruction would be to execute an "arithmetic shift left of accumulator A" instruction and would look like this in memory:

Memory Location	Memory Contents (Hex)
0100	48 (ASL A opcode)

48 (in hex) is the ASL A instruction. The result of this instruction will have the contents of accumulator A shifted one place to the left. Bit 0 (LSB) of the accumulator will be loaded with a zero, and bit 7 (MSB) will be loaded into the carry bit of the condition code register.

An example of an inherent memory addressing instruction would be to execute a "Push Data" instruction and would look like this in memory:

Memory Location	Memory Contents (Hex)
0100	36 (PSH A opcode)

36 (in hex) is the PSH A instruction. Execution of this instruction will cause the contents of accumulator A to be loaded into memory at the address contained in the stack pointer register. The stack pointer register is then decremented by one.

AD-2 Addressing Modes

Source input coding to an assembler, written in mnemonics, for the above three instructions would appear as follows:

```
CLC
ASL A
PSH A
```

B. IMMEDIATE

In this mode of addressing, the *operand* is found in the next one or two memory locations following the opcode. For example, to "load accumulator A with the hex number 55", it would look like this in memory:

Memory Location	Memory Contents (Hex)
0100	86 (LDA A immediate opcode)
0101	55 (Data)

86 (in hex) is the LDA A immediate opcode. 55 (in hex) is the data. The result after execution of the above is that hex number 55 has been loaded into accumulator A.

Source input coding would be: LDA A #\$55
signifies the immediate mode of addressing.

C. DIRECT

In this mode of addressing, the *address* is found in the next memory location following the opcode. This enables direct addressing of the first 256 bytes of memory (0000 to 00FF in hex). As an example, to load accumulator A with the contents of memory location 67 (in hex), consecutive memory locations would look like this:

Memory Location	Memory Contents (Hex)
0100	96 (LDA A direct opcode)
0101	67 (Address of memory containing the data)

96 (in hex) is the LDA A direct opcode. 67 (in hex) is the address where the data is to be fetched from. So, whatever is in memory location 0067 would be loaded into accumulator A.

Source input coding would be: LDA A \$67

D. EXTENDED

This mode of addressing is used to address memory locations above 00FF. The second memory location of the instruction contains the high-order 8 bits of the address, and the third memory location contains the low-order 8 bits of the address. For example, to load accumulator A with the contents of memory location 4057 (in hex), the consecutive program memory locations would look like this:

Memory Location	Memory Contents (Hex)
0100	B6 (LDA A extended opcode)
0101	40 (Address high byte)
0102	57 (Address low byte)

B6 (in hex) is the LDA A Extended opcode. 40 (in hex) is the most significant half of the address and 57 (in hex) is the least significant half of the address where data is stored. After execution of the above instruction, the contents of memory location 4057 would have been loaded into accumulator A.

Source input coding would be: LDA A \$4057.

E. INDEXED

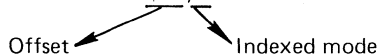
In this mode of addressing, the *number* (offset) found in the second memory location of the instruction is added to the contents of the index register to form a new "effective address". The new "effective address" is the location in memory which contains the data for the operation or is the destination for data.

The effective address is held in a temporary address register so the content of the index register is not destroyed or altered. As an example, if the index register contains the hex A014 and a load accumulator A, indexed with offset of hex 21 is executed, the offset of hex 21 is added to the contents of the index register (A014) to form a new "effective address" of hex A035.

Memory Location	Memory Contents (Hex)
0100	A6 (LDA A indexed opcode)
0101	21 (Offset)

A6 (in hex) is the LDA A Indexed opcode. 21 (in hex) is the offset. To the index register, the offset of 21 is added to form a new "effective address" of hex A035 (A014 + 21). After execution of the above instruction, the *contents* of memory location A035 will have been loaded into accumulator A.

Source input coding would be: LDA A \$21,X



AD-4 Addressing Modes

F. RELATIVE

In this mode of addressing, program control may be transferred to someplace other than the next sequential memory location. It is used for *BRANCH* instructions only. Transfer is limited to 126 memory locations back or 129 memory locations forward from the present location. Since this is a two-byte instruction (two memory locations), transfer is always referenced from the next instruction which the MPU would execute if it did not transfer control. This reference point would be the present value of the program counter after reading the two-byte instruction, or the present location +2.

The number of memory locations to branch over is called the "offset" and is expressed as an 8-bit 2's complement number.

All branches forward are given in a positive 2's complement number from 0 to 7F (in hex). All branches back from the present location are represented as a negative number on 2's complement from FF to 80 (in hex).

TRANSFER FORWARD FROM PRESENT LOCATION

Assume it is desired to branch from the present location of 0100 (in hex) to location 0147 (in hex). First, it should be verified that the branch is not beyond the allowable range of +129 locations from the present location.

Final destination	=	0147 (hex)
Present location +2	=	<u>0102 (hex)</u>
Number of locations to branch forward over	=	45 (hex)

45 (hex) is within our allowable range. The 8-bit 2's complement representation of a positive number in the range of 0 to 7F (hex) is the number itself (MSB bit 7 = 0).

Memory Location	Memory Contents (Hex)
0100	20 (BRA opcode)
0101	45 (Offset)
0102	XX (Present value of program counter)
.	.
.	.
0147	XX (Opcode of next instruction that will be executed)

20 (in hex) is the BRA (Branch Always) opcode. 45 (in hex) is the offset or number of locations which will be branched over starting with 0102. Therefore, the next instruction the MPU will execute will be located at 0102 + 45 or location hex 0147.

Source input coding would normally be: BRA LEVEL,

where "LABEL" is the unique label given to the opcode mnemonic at location 0147.

TRANSFER BACK FROM PRESENT LOCATION

Assume it is desired to branch from the present location of 0100 back to memory location 0090 (hex). This is accomplished in a similar manner as the forward branch, except the number of locations is a negative number expressed in 2's complement from the present location +2. The 2's complement form of a negative number places a 1 in bit 7 (MSB) which, in effect, tells the MPU to branch back rather than forward.

Present location +2	=	0102 (hex)
Final location	=	<u>0090 (hex)</u>
Number of locations to branch back over	=	72 (hex)

To represent -72 in 2's complement, first write the binary representation of 72 (hex):

72 (hex) = 01110010

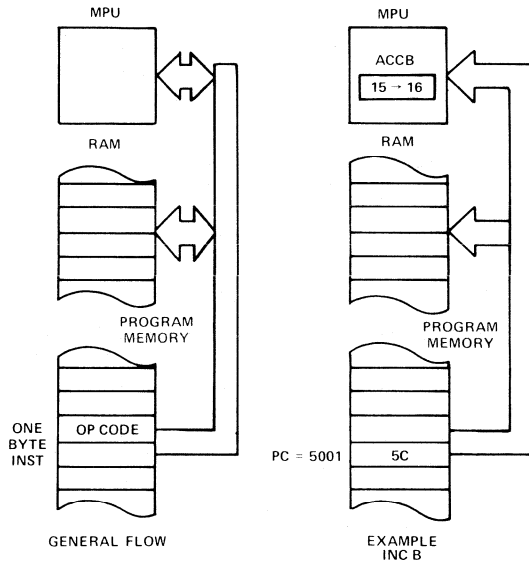
Then take the 1's complement:	10001101
and add 1:	<u>0000001</u>
To give the 2's complement:	10001110

-72 (hex) = 8E (2's complement in hex)

Memory Location	Memory Contents (Hex)
0090	XX (Opcode of next instruction after branch instruction)
.	
.	
.	
0100	20 (BRA opcode)
0101	8E (Offset)
0102	XX (Present value of program counter)

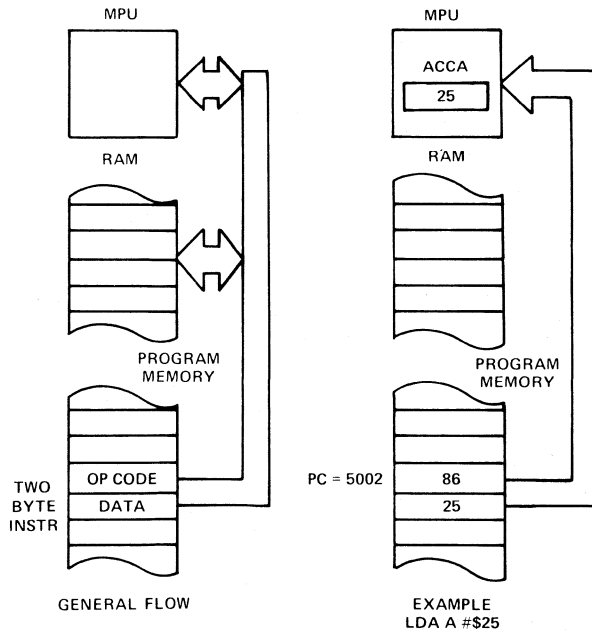
20 (in hex) is the BRA (Branch Always) opcode. 8E is the offset or number of locations which will be branched back over starting from 0102. Therefore, the next instruction the MPU will execute will be located at memory location 0090 (hex).

ACCUMULATOR/INHERENT ADDRESSING



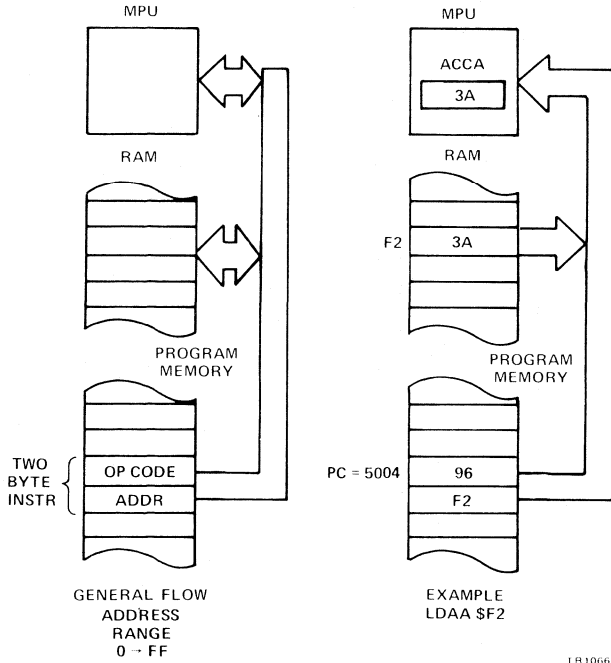
TR1064

IMMEDIATE ADDRESSING

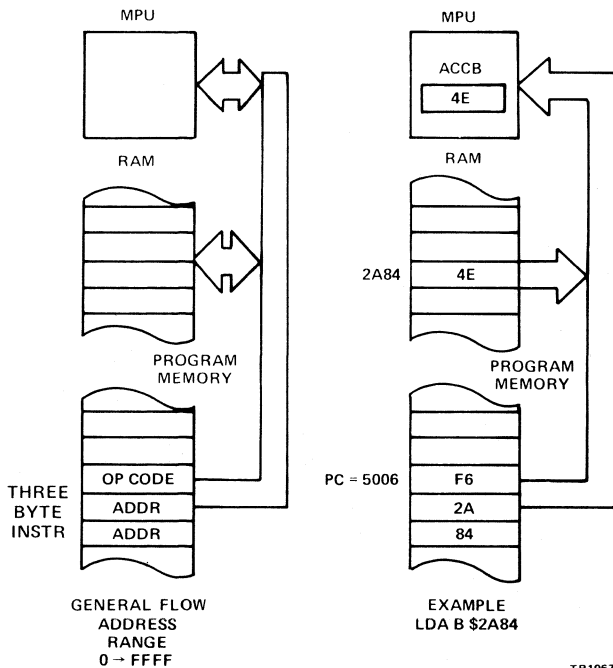


TR1065

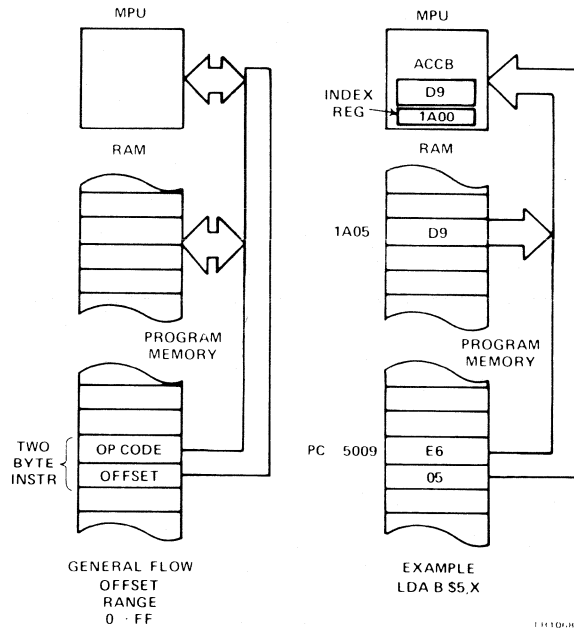
DIRECT ADDRESSING



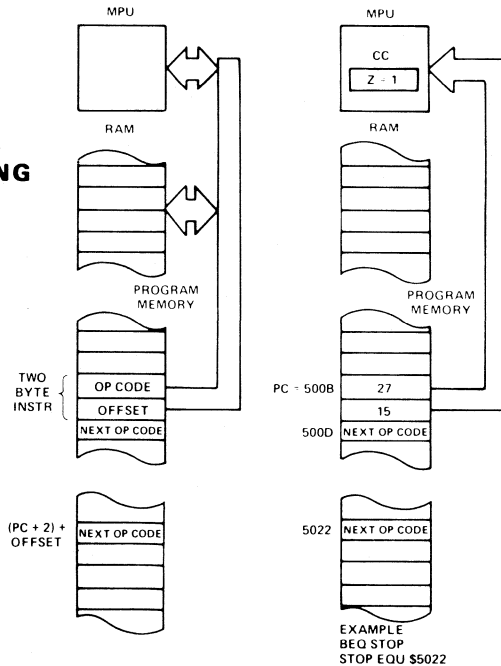
EXTENDED ADDRESSING



INDEXED ADDRESSING



RELATIVE ADDRESSING



IMMEDIATE	DIRECT	INDEXED	EXTENDED
<u>LDAA #SAB</u> 86 AB	<u>LDAA SFE</u> 96 FE	<u>LDAA SA,X</u> A6 0A	<u>LDAA \$4004</u> B6 40 04
<u>LDX #S1F85</u> CE 1F 85	<u>LDX S1F</u> DE 1F	<u>LDX SA,X</u> EE 0A	<u>LDX \$FD20</u> FE FD 20

TR1070

INHERENT/ACCUMULATOR ADDRESSING

INC B

5C

PSH A

36

RELATIVE ADDRESSING

BRA START

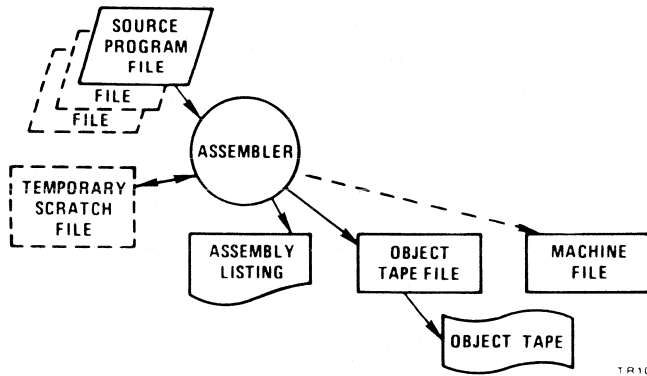
20 OP CODE

16 OFFSET

TR1071

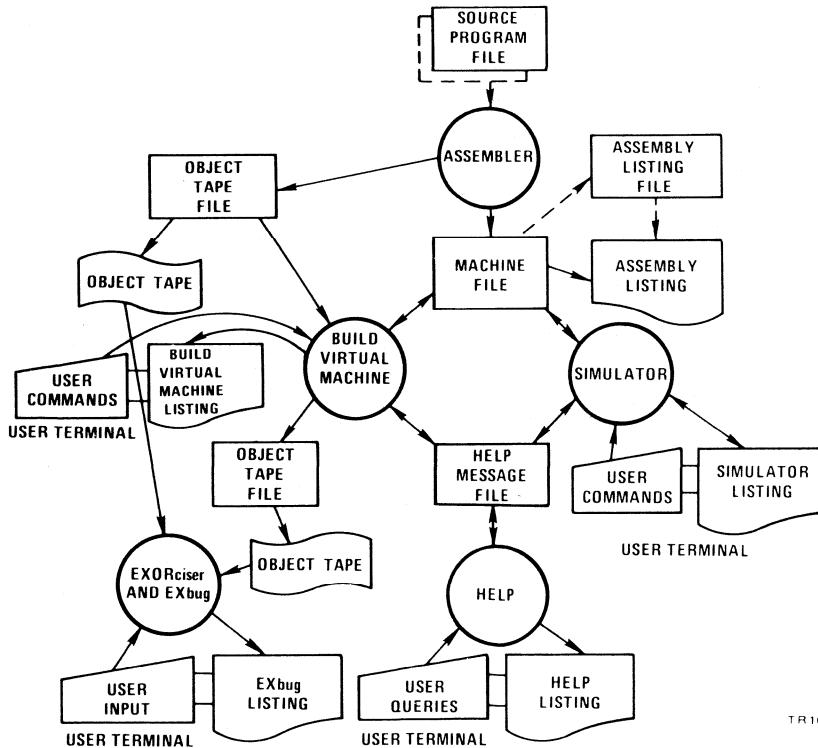
Assembler Techniques

ASSEMBLER BLOCK DIAGRAM



TR1091

SUPPORT SOFTWARE — FUNCTIONAL BLOCK DIAGRAM



TR1092

SOURCE LANGUAGE

FOR THE M6800 MPU SYSTEM

- 72 MNEMONIC INSTRUCTIONS
- 12 ASSEMBLER DIRECTIVES

TR1093

ASSEMBLER CHARACTER SET

- ALPHABET A-Z
- INTEGERS 0-9
- ARITHMETIC OPERATORS + - * /
- PREFIX CHARACTERS
- SUFFIX CHARACTERS
- SEPARATOR CHARACTERS

TR1094

*

- DEFINE COMMENT LINE
- PRESENT VALUE OF LOCATION COUNTER
- USED AS AN ARITHMETIC OPERATOR

```

*THIS IS A TEST PROGRAM.
BRA *+7
LDA A SS*NB-3,X

```

TR1095-1

ASSEMBLER PREFIX CHARACTERS

- DECIMAL NUMBER
- # SPECIFIES IMMEDIATE ADDRESSING MODE
- \$ HEXADECIMAL NUMBER
- @ OCTAL NUMBER
- % BINARY NUMBER
- ASCII LITERAL CHARACTER (20-5F)

```
LDA A #$A5  
LDA B #'C
```

TR1096

ASSEMBLER SUFFIX CHARACTERS

- B BINARY NUMBER
- H HEXADECIMAL NUMBER
- O OCTAL NUMBER
- Q OCTAL NUMBER
- ,X SPECIFIC INDEXED ADDRESSING MODE

```
|| LDA A #0A5H
```

TR1098-1

ASSEMBLER SEPARATOR CHARACTERS

- SPACE
- CR (CARRIAGE RETURN)
- (COMMA)

TR1099

DEFINITION OF THE ASSEMBLER DIRECTIVES

ALPHABETIC LIST OF ASSEMBLER DIRECTIVES

END	END OF PROGRAM
EQU	EQUATE SYMBOL
FCB	FORM CONSTANT BYTE
FCC	FORM CONSTANT CHARACTERS
FDB	FORM DOUBLE CONSTANT BYTE
MON	RETURN TO MONITOR CONSOLE
NAM	NAME PROGRAM
OPT	OPTION
ORG	ORIGIN
PAGE	ADVANCE LISTING TO TOP OF PAGE
RMB	RESERVE MEMORY BYTES
SPC n	SPACE n LINES

TR1102

NAM

1. THE "NAM" DIRECTIVE:
 - A) NAMES THE PROGRAM - HEADING TEXT
 - B) PROVIDES TOP OF FILE AND PAGE 1 FOR PAGING
2. DO NOT USE LABEL

```
          COMMENT
      _____
NAM CLPROB 1-1-75 REV001
      _____
      PRINTED OUT
      AT TOP OF
      ASSEM LISTING
```

TR1103-1

OPT

1. THE "OPT" DIRECTIVE ALLOWS THE USER TO SELECT OR CONTROL VARIOUS OPERATIONS (OUTPUT) OF THE CROSS-ASSEMBLER
2. DO NOT USE LABEL WITH "OPT"
3. NO OBJECT CODE RESULTS FROM "OPT"

```
NAM CLPROB
OPT M,S,NOP
```

TR1104-1

FDB (FORM DOUBLE BYTE)

1. THE "FDB" DIRECTIVE IS USED TO FORM TABLES IN MEMORY WITH DOUBLE BYTES OR 16-BIT DATA

```

ORG $3F00
TABLE FDB0, $AABB, 10000,@55552, TABLE
    
```

MEM ADR	MEM CONTENTS
3F00	00
3F01	00
3F02	AA
3F03	BB
3F04	27
3F05	10
3F06	5B
3F07	6A
3F08	3F
3F09	00

TR1111-1

FCC (FORM CONSTANT CHARACTERS)

1. THE "FCC" DIRECTIVE IS USED TO FORM TABLES IN MEMORY WITH ASCII CHARACTERS

```

ORG $8A00
MESS1 FCC / ERR 208 /
MESS2 FCC 16, TEST ROUTINE 6
    
```

RESULTING OBJECT CODE			
8A00	20	8A15	45
1	45	6	20
2	52	7	36
3	52	8	20
4	20		
5	32		
6	30		
7	38		
8	20		
9	20		
A	54		
B	45		
C	53		
D	54		
E	20		
F	52		
8A10	4F		
1	55		
2	54		
3	49		
4	4E		

TR1112-1

<u>CODE</u>	<u>SUMMARY DEFINITION</u>	<u>FUNCTION</u>
ORG	ASSIGN ORIGIN OF PROGRAM COUNTER	DEFINES THE NUMERICAL ADDRESS OF THE FIRST BYTE OF A SUBSEQUENT SEGMENT OF THE CODED PROGRAM.
EQU	EQUATE A SYMBOL TO AN OPERAND	EQUATES A SYMBOL TO A NUMERICAL VALUE, ANOTHER SYMBOL, OR AN EXPRESSION.
FCB	FORM CONSTANT BYTE	} ASSIGN VALUES AND ADDRESSES OF DATA, AND ASSIGN ADDRESSES OF SCRATCH AREAS OF MEMORY.
FCC	FORM CONSTANT CHARACTERS	
FDB	FORM DOUBLE CONSTANT BYTE	
RMB	RESERVE MEMORY BYTES	
END	DEFINE END OF SOURCE PROGRAM	} CONTROL THE SEQUENCING OF SOURCE PROGRAMS THROUGH THE ASSEMBLER.
MON	RETURN TO CONSOLE	
NAM	NAME THE PROGRAM OR INSERT TEXT	} FORMAT CONTROL (SOURCE PROGRAM AND/OR ASSEMBLER LISTING)
OPT	ASSEMBLER CONTROL OPTIONS	
PAGE	MOVE PAPER TO TOP OF FORM	
SPC	VERTICAL SPACING OF PROGRAM LISTING	

TR1115

READY
 OLD DEF123

READY
 LIST

```

DEF123      31:48EDT      03 12 74

100 NAM ITEM2
110 OPT MEM
120 * ADDITION OF TWO MULTIPLE-PRECISION
130 * BINARY-CODED-DECIMAL NUMBERS.
140 *
150 NB EQU 8      UNDEFINED 8-BYTE OPERAND.
160 *
170 * BEGIN SUBROUTINE.
180 OPG $1000
190 BCD LDA B #NB
200 LDM ADDR      LOADS DATA ADDRESS.
210 CLC
220 NEXT LDA A #B-1    START LOOP
230 ADD A 2*NB-1
240 DAA
250 STA A 3*NB-1
260 DEC
270 DEC B
280 BNE NEXT      END OF LOOP
290 RTI          END OF BCD SUBROUTINE.
300 *
310 *
320 * BEGIN MAIN PROGRAM.....
330 * TEST OF SUBROUTINE BCD.
340 OPG $1100
350 LDC #B10F      INITIALIZE ITCN PTR.
360 LDM #P         LOADS ADDRESS OF P.
370 ITC ADDR
380 JCR BCD
390 NOP
400 BRR #-1      END OF MAIN PROGRAM.
410 *
420 *
430 * ALLOCATE A DATA AREA IN
440 * READ-WRITE MEMORY.
450 OPG $0100
460 *      1) FOR THE SUBROUTINE.
470 ADDR PMB 2
480 *      2) FOR THE MAIN PROGRAM.
490 P PMB NB
500 O PMB NB
510 RES PMB NB
520 END
530 MON

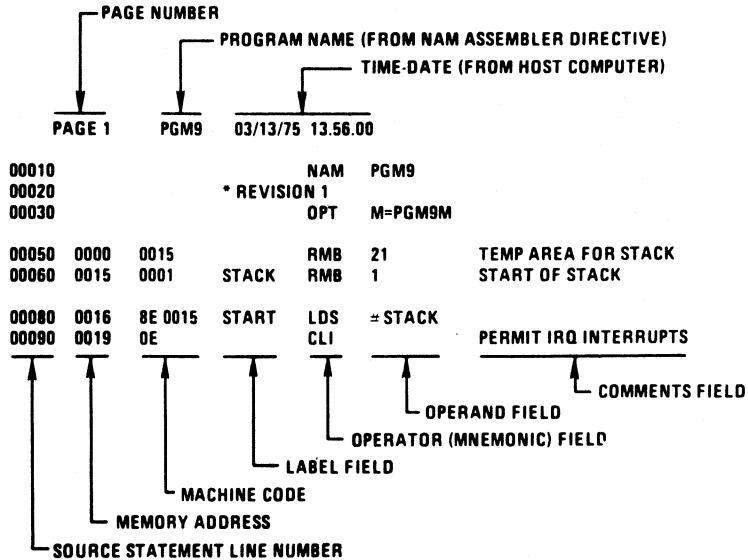
```

READY

Listing of the Source Program "ITEM2"

TR1116

ASSEMBLY LISTING FORMAT



TR1117

Object Tape File

The assembler when instructed by an "OPT OT=filename" will generate an Object Tape File which can be used to generate a paper tape or cassette of the machine language object code generated. The object tape and/or object tape file can be used as input to the machine file for use by the Simulator. The object tape format is also accepted by the EXbug firmware to load the object program into an EXORciser.

Format Object Tape File

```

000600004844521E
011301008E0132FE01370603960AA1022705095A59
0111011026F63EBD01197E010016BA013339F0
010C013380100405013953455400
00030000FC
    
```



M68000

GE

SUPPORT SOFTWARE

PROGRAMMABLE LOGIC — *the easy way*

Motorola software for the M6800 microcomputer family is currently operating on the General Electric Information Services International Network:

- MPCASM** — M6800 Cross Assembler converts symbolic source code to machine-language with listing.
- MPSSIM** — M6800 Interactive Simulator duplicates the execution of machine-language instructions assembled with the cross assembler.
- HELP** — The HELP Program provides the user of Motorola support software with real time documentation of the software. This documentation includes abbreviated operating procedures.
- MPBVM** — Build Virtual Machine program simplifies the file management problems associated with developing microprocessor programs.

TO ACCESS THE SOFTWARE:

1. Contact your local G E Information Services Sales Department and request service under the NSS (Network Software Services) catalog "AQ36."
2. If you are a new user also ask the G E salesman for "Command System Manual" and "Editing Commands Manual" for Mark III Foreground System.
3. For detailed programming and support software information order your copy of the "M6800 Microprocessor Programming Manual" Motorola Semiconductor Literature Distribution Center, P. O. Box 20924, Phoenix, Arizona 85036.
4. Sign on with your TTY (or other terminal) and you will be up and running.



MOTOROLA Semiconductor Products Inc.

M6800 SUPPORT SOFTWARE



The sample program displayed on this and the next three pages used the GE Mark III Service system to give the new user a capsule view of the procedure for using Motorola's M6800 Support Software.

Item **1** describes preparation of the sample program using the edit features of timesharing.

Item **2** demonstrates how the user can: a) create a machine file, b) change the machine file's size, and c) alter the label to a meaningful message about the application being developed.

Item **3** shows the conversation to assemble the sample program, and the listing of the program generated by the assembler.

Item **4** explains the simulator conversation and a step by step simulation of the sample program.

Item **5** describes the technique to punch a paper tape of the Object Tape File. This tape is compatible with the Motorola EXORciser.TM

1 CREATE A SAMPLE PROGRAM

HHH _____	Enter HHHH for learning character so the computer can calculate your terminals speed (30 cps, 120 cps, etc.)
UP=NA035005+ _____	Enter your user number and password
PASSWORD _____	
XXXXXXXXXXXXXSR } _____	Optional feature (enter carriage return to bypass)
ID: _____	Assign FORTRAN system
SYSTEM- FIV _____	Create new file with filename "PGM"
NEW OR OLD- _____	
NEW PGM _____	
READY _____	Ready indicates system is ready to accept data or command
100 NAM PGM _____	NOTE: The line number includes the first space following the number; allow for this space character while entering the program.
110 OPT M=MEMF1 SPECIFY MACHINE FILENAME _____	The first record should be a NAM assembler directive: the first six characters of operand will appear in the assembler listing header.
120 OPT D=TAPEF1 OBJECT TAPE FILENAME _____	
130 OPT SYMBOLS SELECT PRINTING OF SYMBOLS _____	The ORG assembler directive sets the program counter.
140 ORG 256 _____	
150 COUNT EQU 03 3 INDICATES OCTAL NUMBER _____	Enter the program: only one space between a line number and a label; otherwise the assembler accepts one or more spaces separating the fields.
160 START LDS #STACK INZ STACK POINTER _____	
170 LDX ADDR _____	<p>The END assembler directive informs the assembler this is the last record of this assembly.</p> <p>The MON assembler directive informs the assembler this is the last file to be assembled.</p> <p>SAV is the command to save the new file just created.</p> <p>Examples of program changes; by overwriting and by inserting a new line.</p> <p>EDI RES is the command to resequence the program starting with line number 100 and sequencing by 10.</p> <p>REP is the command to replace (or update) an existing file.</p>
180 LDA B #COUNT _____	
190 BACK LDA 10 DIRECT ADDRESSING _____	
200 CMP A 2,X INDEXED ADDRESSING _____	
210 BEQ FOUND RELATIVE ADDRESSING _____	
220 DEX IMPLIED ADDRESSING _____	
230 DEC B ACCUMULATOR ONLY ADDRESSING _____	
240 BNE BACK _____	
250 WHI WAIT FOR INTERRUPT _____	
260 SPC 1 _____	
270 FOUND JSR SUBRTN JUMP TO SUBROUTINE _____	
280 JMP START EXTENDED ADDRESSING _____	
290 * COMMENT STATEMENT NOTE TRUNCATION 01234567890123456789 _____	
300 SUBRTN TAB COMMENT FIELD TRUNCATION0123456789 _____	
310 ADDA BYTE SET MOST SIGNIFICANT BIT _____	
320 RTS RETURN FROM SUBROUTINE _____	
330 SPC 2 _____	
340 RMB 20 SCRATCH AREA FOR STACK _____	
350 STACK RMB 1 START OF STACK _____	
360 BYTE FCB #30 FORM CONSTANT BYTE _____	
370 FCB \$10,\$4 \$ INDICATES HEXADECIMAL _____	
380 ADDR FDB DATA FORM CONSTANT DOUBLE BYTE _____	
390 DATA FDC /SET/ FORM DATA STRING (ASCII) _____	
400 END _____	
410 MON _____	
SAV _____	
READY _____	
310 ORAA BYTE SET MOST SIGNIFICANT BIT _____	
105 * REVISION 01 _____	
EDI RES _____	REP is the command to replace (or update) an existing file.
READY _____	
REP _____	
READY _____	

3 ASSEMBLE THE SAMPLE PROGRAM

```

RUN MPCASM
-----
MPCASM      01:40EDT      04/30/75
---

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPCASM
COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU CROSS ASSEMBLER, RELEASE 1.4

ENTER SI FILENAME
?PGM

FILE'S LABEL:
SOURCE FILENAME: PGM
AUTHOR: JOHN DOE
---
```

Call the cross assembler and cause it to run.

The release number is changed as the cross assembler is updated with improvements.

Enter SI (source input) filename of the program to be assembled.

The contents of the Label Buffer Area from the machine file MEMFI.

Line number.

Program counter (hexadecimal).

Hexadecimal instruction, data, or value.

An asterisk (*) may be used as the first character of a comment statement.

The # indicates immediate addressing.

The missing line 270 was a SPC 1 assembler directive.

The \$ indicates a hexadecimal value follows.

NOTE: For more detailed information as to specific meaning of mnemonics and the details of each program refer to the M6800 MICROPROCESSOR PROGRAMMING MANUAL.

```

PAGE 1 PGM      04/30/75 01:40.00

00100
00110
00120
00130
00140
00150 0100
00160      0003 COUNT EQU 93          * INDICATES OCTAL NUMBER
00170 0100 8E 0132 START LDS #STACK  INZ STACK POINTER
00180 0103 FE 0136 LDX ADDR
00190 0106 C6 03 LDA B #COUNT
00200 0108 96 0A BACK LDA A 10 DIRECT ADDRESSING
00210 010A A1 02 CMP A 2*X INDEXED ADDRESSING
00220 010C 27 05 BEQ FOUND RELATIVE ADDRESSING
00230 010E 09 DEX IMPLIED ADDRESSING
00240 010F 5A DEC B ACCUMULATOR ONLY ADDRESSING
00250 0110 26 F6 BNE BACK
00260 0112 3E WAI WAIT FOR INTERRUPT

00280 0113 8D 0119 FOUND JSR SUBRTN JUMP TO SUBROUTINE
00290 0116 7E 0100 JMP START EXTENDED ADDRESSING
00300
00310 0119 16 SUBRTN TAB * COMMENT STATEMENT NOTE TRUNCATION 01234567890123
00320 011A BA 0133 DRB A BYTE COMMENT FIELD TRUNCATION012
00330 011D 39 RTS RETURN FROM SUBROUTINE

00350 011E 0014 RMB 20 SCRATCH AREA FOR STACK
00360 0132 0001 STACK RMB 1 START OF STACK
00370 0133 80 BYTE FCB $80 FORM CONSTANT BYTE
00380 0134 10 FCB $10,$4 $ INDICATES HEXADECIMAL
00390 0135 04
00390 0136 0138 ADDR FDB DATA FORM CONSTANT DOUBLE BYTE
00400 0138 53 DATA FCC /SET/ FORM DATA STRING (ASCII)
00410 0139 45
00410 013A 54
00410 END

SYMBOL TABLE

ADDR 0136 BACK 0108 BYTE 0133 COUNT 0003 DATA 0138
FOUND 0113 STACK 0132 START 0100 SUBRTN 0119
```

4 SIMULATE THE SAMPLE PROGRAM

RUN MPSSIM

MPSSIM 01:42EDT 04/30/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPSSIM
 COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU SIMULATOR, RELEASE 1.3

ENTER MF FILENAME

?MEMF1

FILE'S LABEL:

SOURCE FILENAME: PGM

AUTHOR: JOHN DOE

HH IA DC EA P X A B C S T

0000 *** 0000 0000 0000 00 00 000000 0000 00000000

?SM 0A,54,SR P 100,T 0C

```

*0100 LDS *0102*0103 0000 00 00 000000*0132 00000003
*0103 LDX *0137*0105*0138 00 00 000000 0132 00000008
*0105 LDA B*0107*0108 0138 00*03 000000 0132 00000010
*0108 LDA A*000A*010A 0138*54 03 000000 0132 00000013
*010A CMP A*013A*010C 0138 54 03 000200 0132 00000018
*010C BEQ *010D*0113 0138 54 03 000200 0132 00000022
*0113 JSR *0131*0119 0138 54 03 000200*0130 00000031
*0119 TAB *0119*011A 0138 54*54 000000 0130 00000033
*011A ORA A*0133*011D 0138*D4 54 000000 0130 00000037
HH IA DC EA P X A B C S T
*011D RTS *0132*0116 0138 D4 54 000000*0132 00000042
*0116 JMP *0118*0100 0138 D4 54 000000 0132 00000045
*0100 LDS *0102*0103 0138 D4 54 000000 0132 00000048
?DM 100,3B
    
```

```

0100 3E 01 32 FE 01 36 C6 03 96 0A A1 02 27 05 09 5A ..2..6.....?..Z
0110 26 F6 3E BD 01 19 7E 01 00 16 BA 01 33 39 00 00 &.>.....39..
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0130 00 01 16 30 10 04 01 33 53 45 54 .....3SET
    
```

?RS.D

0000 *** 0000 0000 0000 00 00 000000 0000 00000000

?EX

PROGRAM STOP AT 0

5 PUNCH A PAPER TAPE

OLD TAPEF1

READY

LISTNH

S00600004844521B

S11301008E0132FE0136C603960AA1022705095A5A

S111011026F63EBD01197E010016BA013339F0

S10B01338010040138534554F4

S9030000FC

BYE

Enter the MF (machine file) filename of the machine file to be simulated.

The contents of the Label Buffer Area from the machine file MEMF1.

Register heading:

- HH Input and output base hexadecimal
- IA Instruction address
- OC Operator mnemonic code
- EA Effective operand address
- P Program counter
- X Index register
- A Accumulator A
- B Accumulator B
- C Condition code register
- S Stack pointer
- T Time cycles (always decimal)

Simulator commands separated by period

- SM 0A,54 Set memory location A to contain 54
- SR P 100 Set register (Program Counter) equal 100
- T 0C Trace C instructions

Simulator command Display Memory; beginning with location 100, display 3B (hex) bytes (note the right margin contains the literal equivalent of the printable characters; the periods show nonprintable characters).

Simulator commands: RS restore registers; D display registers.

Simulator command EX exit simulator

NOTE: Hexadecimal input to the simulator requires the first character be numeric (i.e. to enter the hex. value "C" enter "0C")

OLD TAPEF1 calls the formatted tape image so it may be punched and listed.

List the Object Tape File TAPEF1 without heading. Turn on punch device before entering carriage return.

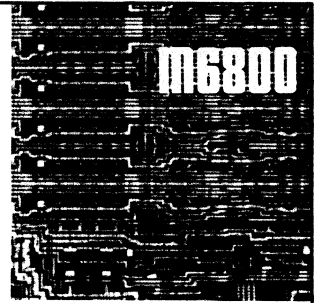
S0 ... indicates a header record

S1 ... indicates a data record

S9 ... indicates an end-of-file record

Sign-off system (enter goodBYE)

Once a machine file has been created and configured the Build Virtual Machine program need not be run until the configuration needs changing.



2 CREATE A MACHINE FILE

RUN MPBVM _____ Call the Build Virtual Machine program and cause it to run.

MPBVM 01:38EDT 04/30/75 _____ Name of program running.

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPBVM
COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU BUILD VIRTUAL MACHINE, RELEASE 1.4

ENTER "HP HP B" FOR MORE HELP

?MF MEMF1 _____ Fetch the machine file named MEMF1

◆◆◆ATTN: 92 _____

FILE'S LABEL:

◆◆ DEFAULT VIRTUAL MACHINE FILE USED ◆◆
THIS FILE IS SETUP FOR 4K (4096) WORDS OF
MEMORY AND A LAST WORD ADDRESS OF \$0FFF.

Indication that the machine file MEMF1 did not exist and that a file was created assuming the default parameters.

USE MPBVM'S TI AND LW COMMANDS TO CHANGE THE
FILE'S LABEL OR IT'S LAST WORD ADDRESS (SIZE
OF MEMORY). THE COMMAND "HP TI,B" WILL EXPLAIN
THE FORMAT OF THE MPBVM TI COMMAND.

The contents of the default Label Buffer Area.

?LW 01FF _____

Change this machine file's size by setting a new Last Word address (hexadecimal value 1FF was entered).

?TI _____

ENTER TITLE TEXT

? SOURCE FILENAME: PGM

ENTER TITLE TEXT

? AUTHOR: JOHN DOE

Set new information into the Label Buffer Area for this machine file (MEMF1).

ENTER TITLE TEXT

?

?MO _____

Display the Machine File Organization.

VIRTUAL MACHINE FILE MEMF1

FILE'S LABEL:

SOURCE FILENAME: PGM

AUTHOR: JOHN DOE

LAST WORD ADDRESS 1FF

MACRO LIBRARY LISTING

1248 REMAINING CHARACTERS

?EX _____

EX is the command to exit the Build Virtual Machine program.

PROGRAM STOP AT 0

LANGUAGE OF THE M6800 MICROPROCESSOR



MICROPROCESSOR INSTRUCTION SET ALPHABETIC SEQUENCE

ABA	Add Accumulators
ADC	Add with Carry
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to Zero
BGE	Branch if Greater or Equal Zero
BGT	Branch if Greater than Zero
BHI	Branch if Higher
BIT	Bit Test
BLE	Branch if Less or Equal
BLS	Branch if Lower or Same
BLT	Branch if Less than Zero
BMI	Branch if Minus
BNE	Branch if Not Equal to Zero
BPL	Branch if Plus
BRA	Branch Always
BSR	Branch to Subroutine
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CBA	Compare Accumulators
CLC	Clear Carry
CLI	Clear Interrupt Mask
CLR	Clear
CLV	Clear Overflow
CMP	Compare
COM	Complement
CPX	Compare Index Register
DAA	Decimal Adjust
DEC	Decrement
DES	Decrement Stack Pointer
DEX	Decrement Index Register
EOR	Exclusive OR
INC	Increment
INS	Increment Stack Pointer
INX	Increment Index Register
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator
LDS	Load Stack Pointer
LDX	Load Index Register
LSR	Logical Shift Right
NEG	Negate
NOP	No Operation
ORA	Inclusive OR Accumulator
PSH	Push Data
PUL	Pull Data
ROL	Rotate Left
ROR	Rotate Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBA	Subtract Accumulators
SBC	Subtract with Carry
SEC	Set Carry
SEI	Set Interrupt Mask
SEV	Set Overflow
STA	Store Accumulator
STS	Store Stack Register
STX	Store Index Register
SIJB	Subtract
SWI	Software Interrupt
TAB	Transfer Accumulators
TAP	Transfer Accumulators to Condition Code Reg.
TBA	Transfer Accumulators
TPA	Transfer Condition Code Reg. to Accumulator
TST	Test
TSX	Transfer Stack Pointer to Index Register
TXS	Transfer Index Register to Stack Pointer
WAI	Wait for Interrupt

INSTRUCTION ADDRESSING MODES AND ASSOCIATED EXECUTION TIMES

(in microseconds assuming a 1 MHz clock)

	(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative
ABA		2
ADC x		.	2	3	4	5	.	.
ADD x		.	2	3	4	5	.	.
AND x		.	2	3	4	5	.	.
ASL	2	.	.	.	6	7	.	.
ASR	2	.	.	.	6	7	.	.
BCC		4
BCS		4
BEQ		4
BGE		4
BGT		4
BHI		4
BIT x		.	2	3	4	5	.	.
BLE		4
BLS		4
BLT		4
BMI		4
BNE		4
BPL		4
BRA		4
BSR		8
BVC		4
BVS		4
CBA		2
CLC		2
CLI		2
CLR	2	.	.	.	6	7	.	.
CLV		2
CMP x		.	2	3	4	5	.	.
COM	2	.	.	.	6	7	.	.
CPX		.	3	4	5	6	.	.
DAA		2
DEC	2	.	.	.	6	7	.	.
DES		4
DEX		4
EOR x		.	2	3	4	5	.	.
INC	2	.	.	.	6	7	.	.
INS		4
INX		4
JMP		3
JSR		.	.	.	9	8	.	.
LDA x		.	2	3	4	5	.	.
LDS		.	3	4	5	6	.	.
LDX		.	3	4	5	6	.	.
LSR	2	.	.	.	6	7	.	.
NEG	2	.	.	.	6	7	.	.
NOP		2
ORA x		.	2	3	4	5	.	.
PSH		4
PUL		4
ROL	2	.	.	.	6	7	.	.
ROR	2	.	.	.	6	7	.	.
RTI		10
RTS		5
SBA		2
SBC x		.	2	3	4	5	.	.
SEC		2
SEI		2
SEV		2
STA x		.	.	4	5	6	.	.
STS		.	.	.	5	6	7	.
STX		.	.	.	5	6	7	.
SUB x		.	2	3	4	5	.	.
SWI		12
TAB		2
TAP		2
TBA		2
TPA		2
TST	2	.	.	.	6	7	.	.
TSX		4
TXS		4
WAI		9

LIST OF ASSEMBLER DIRECTIVES

END	End of Program
EQU	Equate Symbol
FCB	Form Constant Byte
FCC	Form Constant Characters
FDB	Form Double Constant Byte
MON	Return to Console
NAM	Name
OPT	Option
ORG	Origin
PAGE	Top of Form
RMB	Reserve Memory Byte
SPC	Space Lines

ACCX (accumulator only) Addressing

In accumulator only addressing, either accumulator A or accumulator B is specified. These are one-byte instructions.

Immediate Addressing

In immediate addressing, the operand is contained in the second byte of the instruction. No further addressing of memory is required. The MPU addresses this location when it fetches the immediate instruction for execution. These are two/three-byte instructions.

Direct Addressing

In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes in the machine; i.e., locations zero through 255. That part of the memory should be used for temporary data storage and intermediate results. In most configurations, it should be a random access memory. These are two-byte instructions.

Extended Addressing

In extended addressing, the value contained in the second byte of the instruction is used as the higher eight-bits of the address of the operand. The third byte of the instruction is used as the lower eight-bits of the address of the operand. This gives one a 16-bit address for the operand. This is an absolute address in memory. These are three-byte instructions.

Indexed Addressing

In indexed addressing, the value contained in the second byte of the instruction is added to the index register lower eight-bits in the MPU. The carry is then added to the higher order eight-bits of the index register. This result is then used to address memory. The modified address is held in a temporary address register so there is no change to the index register. These are two-byte instructions.

Implied Addressing

In the implied addressing mode the instruction gives the address (i.e., stack pointer, index register, etc.). These are one-byte instructions.

Relative Addressing

In relative addressing, the value contained in the second byte of the instruction is added to the program counters lowest eight-bits plus two. The carry or borrow is then added to the high eight-bits. This allows the user to address data within a range of -126 to +129 bytes of the present instruction. These are two-byte instructions.



MOTOROLA
Semiconductor Products Inc.

3005 EAST McDOWELL ROAD, PHOENIX, ARIZONA 85008



M6800

SUPPORT SOFTWARE

PROGRAMMABLE LOGIC — *the easy way*



Motorola's M68SAM Cross Assembler for the M6800 microcomputer family is currently available as a remote batch program operating on the General Electric Information Services International Network. Remote batch processing provides substantial cost savings to users who do not need assembly results immediately.

TO ACCESS THE SOFTWARE:

1. Contact your local GE Information Services Sales Department and request service under the NSS (Network Software Services) catalog "AQ36".
2. Ask the GE salesman for
 - Command System Manual
 - Editing Commands Manual
 - Foreground-Background Interface Reference Manual
 - Foreground-Background Interface User's Guide
3. For detailed programming and support software information, order your copy of the "M6800 Microprocessor Programming Manual" from Motorola Semiconductor Literature Distribution Center, P. O. Box 20924, Phoenix, Arizona 85036.
4. Sign on with your teletype (or other terminal) and you will be up and running.



MOTOROLA Semiconductor Products Inc.

M6800 SUPPORT SOFTWARE



The M68SAM is a subset of the MPCASM assembler. M68SAM always generates a list file and an object tape file. The M68SAM supports the following OPT assembler directives:

- DB8
- DB10
- DB16 (default)
- LIST (Default)
- NOLIST

Any other OPT assembler directive will cause an error message:

****ERROR 217

If the assembler output is to be simulated, the foreground program MPBVM is recommended to load the object tape file into the machine file.

Item **1** illustrates the preparation of a source program. Although none are shown, edit commands may be used to modify the file.

Item **2** shows the use of the input driver program to initiate a background assembly.

Item **3** demonstrates a status check for an initiated background job; and cost of a completed job.

Item **4** shows the use of the output driver in retrieving assembly results.

Item **5** explains the generation of an object tape. The format of this tape is compatible with Build Virtual Machine and Motorola provided loaders.

Item **6** is an annotated assembly listing.

CREATE A SAMPLE PROGRAM

```
H#H# _____ Enter a series of H's for learning characters so the computer
UR#M#036005. } can calculate your terminal's speed (30 cps, 120 cps, etc.)
P#03M#0P#D }
#0000#0#0#0#P } Enter your user number and password
ID: _____ Optional feature (enter carriage return to bypass)
SYSTEM= FIV _____ Assign FORTRAN system
NEW OP OLD= _____
NEW PGM _____ Create new file with filename "PGM"

READY _____ Ready indicates system is ready to accept data or
command.

100 NAM PGM _____ NOTE: The line number includes the first space follow-
110 ♦ REVISION 01.A ing the number; allow for this space character while
120 ORG 256 _____ entering the program.
130 COUNT EQU 99 9 INDICATES OCTAL NUMBER The first record should be a NAM assembler directive:
140 START LDS #STACK INZ STACK POINTER the first six characters of operand will appear in the
150 LDX ADDR assembler listing header.
160 LDA B ACOUNT The ORG assembler directive sets the program counter.
170 BACK LDAR 10 DIRECT ADDRESSING
180 CMP A 24X INDEXED ADDRESSING
190 BEQ FOUND RELATIVE ADDRESSING
200 DEX IMPLIED ADDRESSING
210 DEC B ACCUMULATOR ONLY ADDRESSING
220 BNE BACK
230 WAI WAIT FOR INTERRUPT
240 SPC 1
250 FOUND JIR SUBRTN JUMP TO SUBROUTINE
260 JMP START EXTENDED ADDRESSING
270 ♦ COMMENT STATEMENT NOTE TRUNCATION 01234567890123456789
280 SUBRTN TAB COMMENT FIELD TRUNCATION0123456789
290 ORAR BYTE SET MOST SIGNIFICANT BIT
300 RTS RETURN FROM SUBROUTINE
310 SPC 2
320 RMB 20 SCRATCH AREA FOR STACK
330 STACK RMB 1 START OF STACK
340 BYTE FC# 80 FORM CONSTANT BYTE
350 FC# 810.84 8 INDICATES HEXADECIMAL
360 ADDR FC# DATA FORM CONSTANT DOUBLE BYTE
370 DATA FC# /SET/ FORM DATA STRING (ASCII)
380 END _____ The END assembler directive informs the assembler this
390 MON _____ is the last record of this assembly.
SAV _____ The MON assembler directive informs the assembler this
is the last file to be assembled.
SAV is the command to save the new file just created.

READY
```



2 INITIATE BACKGROUND ASSEMBLY

RUN M68SAMI _____ Call the background assembler and cause it to run.

M68SAMI 11:09EST 10/30/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR M68SAMI
INITIATE BACKGROUND ASSEMBLER, RELEASE 1.0

SOURCE FILE NAME: ?PGM _____

PRIORITY: ? _____

CONTROL FILE NAME: M68S9996 _____
JOB ID = NG17 _____

PROGRAM STOP AT 1190

BYE _____

Filename of foreground source program to be assembled in background.

Three priorities are available:

H - High Priority. Job will be initiated within 15 minutes. (Highest cost).

(C/R) - Normal Priority. Job will be initiated within 3 hours.

L - Low Priority. Job may be deferred for overnight processing. (Lowest cost).

The control file includes foreground-background interface commands and control for the background program.

ID of background job. This will be needed later to retrieve results.

Sign off and allow time for job to run in background.

3 CHECK STATUS OF BACKGROUND JOB

BST NG17 _____ Background Job ID.

NG17 DONE _____ Status indicates job is complete

00015 RETURNED
00003 SUBMITTED
SYSOUT REPORTS

NG1701\$\$-RETURNED
NG170174-RETURNED
NG170103-RETURNED
NG170102-RETURNED
NG170212-RETURNED

OBJECT TAPE File

ASSEMBLY LISTING File

The other files are system-generated reports not normally referenced.

READY

Job's status is returned by the system; 'DONE' indicates the job is complete and output may be retrieved. Other status messages are explained in the GE Foreground/Background Interface Reference Manual.

BRES NG17 _____ Check the cost of the Background job.

ACTIVITY APPROXIMATE CENTS



4 RETRIEVE OUTPUT

RUN M68SAMO _____ Retrieve the output from background assembler.

M68SAMO 11:54EST 10/30/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR M68SAMO
RETRIEVE BACKGROUND ASSEMBLER OUTPUT, RELEASE 1.0

JOB ID: ?M617 _____ Enter ID of Background Job

LIST FILE: ?PREMLIST _____ Name of file in which assembly listing will be saved; a carriage return indicates the listing is not to be retrieved.

LIST FILE SAVED

OBJECT FILE: ?TAPEF1 _____ Name of file in which object output will be saved; a carriage return indicates the object output is not to be retrieved.

OBJECT FILE SAVED
CONTROL FILE TO DELETE: ?M6809996 _____ Control file is no longer needed, and should be purged.

M6809996 PURGED
PURGE JOB - YES/NO /YES _____ Purge all output from background job after desired files have been saved. If output is not purged at this time, it will be done automatically after 36 hours.

M617 PURGED

PROGRAM STOP AT 580

NOTE: The drivers M68SAMI and M68SAMO were written to simplify the use of background processing. Additional flexibility and, in some cases additional cost savings, may be achieved with user-supplied interface commands. Information about the foreground-background command interface is available from the GE manuals listed on the cover or contact your GE Account Representative.

5 PUNCH A PAPER TAPE

LISTINH TAPEF1 _____ List the Object Tape File without heading. Turn on punch device before entering carriage return.

S00600004844521E _____ S0... indicates a header record

S11301008E0132FE0136C603960AA1022705095A5A } _____ S1:... indicates a data record

S111011028F63EBD01197E010016BA013339F0

S1080133801004013853455407

S9030000FC _____ S9... indicates an end-of-file record

6 ASSEMBLY LISTING

LIST ASMLIST _____ Name of list file created by M68SAM0
(see Step 4... Retrieve output)

ASMLIST 11:56EST 10/30/75

SNUMB = M6G17, ACTIVITY # = 01, REPORT CODE = 03, RECORD COUNT = 00052

1 MOTOROLA M68SAM CROSS-ASSEMBLER PAGE 1

M68SAM IS THE PROPERTY OF MOTOROLA SPD, INC.
COPYRIGHT 1974 BY MOTOROLA INC

MOTOROLA M6800 CROSS ASSEMBLER, RELEASE 1.1

The release number is changed as the cross assembler is updated with improvements.

Line number.

Program counter (hexadecimal).

Hexadecimal instruction, data, or value.

```

00100
00110
00120 0100
00130 0003
00140 0100 8E 0132 START
00150 0103 FE 0136
00160 0106 C6 03
00170 0108 96 0A BACK
00180 010A A1 02
00190 010C 27 05
00200 010E 09
00210 010F 5A
00220 0110 26 F6
00230 0112 3E

```

An asterisk (*) may be used as the first character of a comment statement.

The # indicates immediate addressing.

```

00250 0113 BD 0119 FOUND JSP SUBRTN JUMP TO SUBROUTINE
00260 0116 7E 0100 JMP START EXTENDED ADDRESSING
00270
00280 0119 14 SUBRTN TAB COMMENT STATEMENT NOTE TRUNCATION 0123456789012345
00290 011A EA 0133 DPA A BYTE COMMENT FIELD TRUNCATION 01234
00300 011D 39 RTS RETURN FROM SUBROUTINE

```

The missing line 240 was an SPC 1 assembler directive.

The missing line 310 was an SPC 2 assembler directive.

```

00320 011E 0014 RMB 20 SCRATCH AREA FOR STACK
00330 0132 0001 STACK RMB 1 START OF STACK
00340 0133 80 BYTE FCB $80 FORM CONSTANT BYTE
00350 0134 10 FCB $10*$4 $ INDICATES HEXADECIMAL
00350 0135 04
00360 0136 0138 * ADDR FDB DATA FORM CONSTANT DOUBLE BYTE
00370 0138 53 DATA FCC <SET> FORM DATA STRING (ASCII)
00370 0139 45
00370 013A 54
00390 END

```

The \$ indicates a hexadecimal value follows.

SYMBOL TABLE

```

COUNT 0003 START 0100 BACK 0108 FOUND 0113 SUBRTN 0119
STACK 0132 BYTE 0133 ADDR 0136 DATA 0138

```

M68SAM does not sort the symbol table

READY

NOTE: For more detailed information as to specific meaning of mnemonics and the details of each program refer to the M6800 MICROPROCESSOR PROGRAMMING MANUAL.

LANGUAGE OF THE M6800 MICROPROCESSOR

M6800

MICROPROCESSOR INSTRUCTION SET ALPHABETIC SEQUENCE

ABA	Add Accumulators
ADC	Add with Carry
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to Zero
BGE	Branch if Greater or Equal Zero
BGT	Branch if Greater than Zero
BHI	Branch if Higher
BIT	Bit Test
BLE	Branch if Less or Equal
BLS	Branch if Lower or Same
BLT	Branch if Less than Zero
BMI	Branch if Minus
BNE	Branch if Not Equal to Zero
BPL	Branch if Plus
BRA	Branch Always
BSR	Branch to Subroutine
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CBA	Compare Accumulators
CLC	Clear Carry
CLI	Clear Interrupt Mask
CLR	Clear
CLV	Clear Overflow
CMP	Compare
COM	Complement
CPX	Compare Index Register
DAA	Decimal Adjust
DEC	Decrement
DES	Decrement Stack Pointer
DEX	Decrement Index Register
EOR	Exclusive OR
INC	Increment
INS	Increment Stack Pointer
INX	Increment Index Register
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator
LDS	Load Stack Pointer
LDX	Load Index Register
LSR	Logical Shift Right
NEG	Negate
NOP	No Operation
ORA	Inclusive OR Accumulator
PSH	Push Data
PUL	Pull Data
ROL	Rotate Left
ROR	Rotate Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBA	Subtract Accumulators
SBC	Subtract with Carry
SEC	Set Carry
SEI	Set Interrupt Mask
SEV	Set Overflow
STA	Store Accumulator
STS	Store Stack Register
STX	Store Index Register
SUB	Subtract
SWI	Software Interrupt
TAB	Transfer Accumulators
TAP	Transfer Accumulators to Condition Code Reg.
TBA	Transfer Accumulators
TPA	Transfer Condition Code Reg. to Accumulator
TST	Test
TSX	Transfer Stack Pointer to Index Register
TXS	Transfer Index Register to Stack Pointer
WAI	Wait for Interrupt

INSTRUCTION ADDRESSING MODES AND ASSOCIATED EXECUTION TIMES

(in microseconds assuming a 1 MHz clock)

(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative
ABA	*	*	*	*	*	2	*
ADC x	*	2	3	4	5	*	*
ADD x	*	2	3	4	5	*	*
AND x	*	2	3	4	5	*	*
ASL	2	*	*	6	7	*	*
ASR	2	*	*	6	7	*	*
BCC	*	*	*	*	*	4	*
BCS	*	*	*	*	*	4	*
BEQ	*	*	*	*	*	4	*
BGE	*	*	*	*	*	4	*
BGT	*	*	*	*	*	4	*
BHI	*	*	*	*	*	4	*
BIT	*	*	*	*	*	4	*
BLE	*	*	*	*	*	4	*
BLS	*	*	*	*	*	4	*
BLT	*	*	*	*	*	4	*
BMI	*	*	*	*	*	4	*
BNE	*	*	*	*	*	4	*
BPL	*	*	*	*	*	4	*
BRA	*	*	*	*	*	4	*
BSR	*	*	*	*	*	8	*
BVC	*	*	*	*	*	4	*
BVS	*	*	*	*	*	4	*
CBA	*	*	*	*	*	2	*
CLC	*	*	*	*	*	2	*
CLI	*	*	*	*	*	2	*
CLR	2	*	*	6	7	*	*
CLV	*	*	*	*	*	2	*
CMP x	*	2	3	4	5	*	*
COM	2	*	*	6	7	*	*
CPX	*	3	4	5	6	*	*
DAA	*	*	*	*	*	2	*
DEC	2	*	*	6	7	*	*
DES	*	*	*	*	*	4	*
DEX	*	*	*	*	*	4	*
EOR x	*	2	3	4	5	*	*
INC	2	*	*	6	7	*	*
INS	*	*	*	*	*	4	*
INX	*	*	*	*	*	4	*
JMP	*	*	*	3	4	*	*
JSR	*	*	*	9	8	*	*
LDA x	*	2	3	4	5	*	*
LDS	3	4	5	6	*	*	*
LDX	3	4	5	6	*	*	*
LSR	2	*	*	6	7	*	*
NEG	2	*	*	6	7	*	*
NOP	*	*	*	*	*	2	*
ORA x	*	2	3	4	5	*	*
PSH	4	*	*	*	*	*	*
PUL	4	*	*	*	*	*	*
ROL	2	*	*	6	7	*	*
ROR	2	*	*	6	7	*	*
RTI	*	*	*	*	*	10	*
RTS	*	*	*	*	*	5	*
SBA	*	*	*	*	*	2	*
SBC x	*	2	3	4	5	*	*
SEC	*	*	*	*	*	2	*
SEI	*	*	*	*	*	2	*
SEV	*	*	*	*	*	2	*
STA x	*	4	5	6	*	*	*
STS	*	5	6	7	*	*	*
STX	*	5	6	7	*	*	*
SUB x	*	2	3	4	5	*	*
SWI	*	*	*	*	*	12	*
TAB	*	*	*	*	*	2	*
TAP	*	*	*	*	*	2	*
TBA	*	*	*	*	*	2	*
TPA	*	*	*	*	*	2	*
TST	2	*	*	6	7	*	*
TSX	*	*	*	*	*	4	*
TXS	*	*	*	*	*	4	*
WAI	*	*	*	*	*	9	*

LIST OF ASSEMBLER DIRECTIVES

END	End of Program
EQU	Equate Symbol
FCB	Form Constant Byte
FCC	Form Constant Characters
FDB	Form Double Constant Byte
MON	Return to Console
NAM	Name
OPT	Option
ORG	Origin
PAGE	Top of Form
RMB	Reserve Memory Byte
SPC	Space Lines

ACCX (accumulator only) Addressing

In accumulator only addressing, either accumulator A or accumulator B is specified. These are one-byte instructions.

Immediate Addressing

In immediate addressing, the operand is contained in the second byte of the instruction. No further addressing of memory is required. The MPU addresses this location when it fetches the immediate instruction for execution. These are two/three-byte instructions.

Direct Addressing

In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes in the machine; i.e., locations zero through 255. That part of the memory should be used for temporary data storage and intermediate results. In most configurations, it should be a random access memory. These are two-byte instructions.

Extended Addressing

In extended addressing, the value contained in the second byte of the instruction is used as the higher eight-bits of the address of the operand. The third byte of the instruction is used as the lower eight-bits of the address of the operand. This gives one a 16-bit address for the operand. This is an absolute address in memory. These are three-byte instructions.

Indexed Addressing

In indexed addressing, the value contained in the second byte of the instruction is added to the index register lower eight-bits in the MPU. The carry is then added to the higher order eight-bits of the index register. This result is then used to address memory. The modified address is held in a temporary address register so there is no change to the index register. These are two-byte instructions.

Implied Addressing

In the implied addressing mode the instruction gives the address (i.e., stack pointer, index register, etc.). These are one-byte instructions.

Relative Addressing

In relative addressing, the value contained in the second byte of the instruction is added to the program counters lowest eight-bits plus two. The carry or borrow is then added to the high eight-bits. This allows the user to address data within a range of -126 to +129 bytes of the present instruction. These are two-byte instructions.

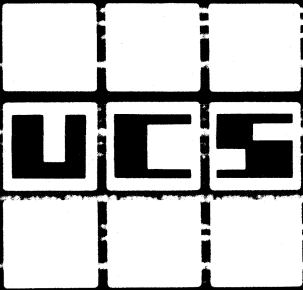


MOTOROLA
Semiconductor Products Inc.

5005 EAST McDOWELL ROAD, PHOENIX, ARIZONA 85008



M6800



UCS

SUPPORT SOFTWARE

PROGRAMMABLE LOGIC — *the easy way*

Motorola software for the M6800 microcomputer family is currently available on United Computing's Multiple Access Remote Computing Service.

- MPCASM** — M6800 Cross Assembler converts symbolic source code to machine-language with listing.
- MPSSIM** — M6800 Interactive Simulator duplicates the execution of machine language instructions assembled with the cross assembler.
- HELP** — The HELP Program provides the user of Motorola support software with real time documentation of the software. This documentation includes abbreviated operating procedures.
- MPBVM** — Build Virtual Machine program simplifies the file management problems associated with developing microprocessor programs.

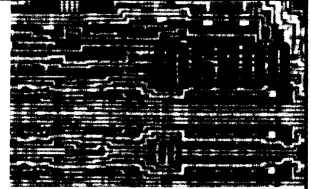
TO ACCESS THE SOFTWARE:

1. Contact your local UCS sales representative and request service for Motorola M6800 Software System under user catalog M437.
2. If you are a new user also request the UCS System Guide and the UNEDIT (Editor) manuals from the UCS sales representative. You will also need to obtain the appropriate telephone numbers to access UCS's time-sharing service.
3. For detailed programming and support software information order your copy of the "M6800 Microprocessor Programming Manual" Motorola Semiconductor Literature Distribution Center, P. O. Box 20924, Phoenix, Arizona 85036.
4. Sign on with your TTY (or other terminal) and you will be up and running.



MOTOROLA Semiconductor Products Inc.

M6800 SUPPORT SOFTWARE



The sample program displayed on this and the next three pages used the U.C.S. Timesharing system to give the new user a capsule view of the procedure for using Motorola's M6800 Support Software.

Item 1 describes preparation of the sample program using the edit features of timesharing.

Item 2 demonstrates how the user can: a) create a machine file, b) change the machine file's size, and c) alter the label to a meaningful message about the application being developed.

Item 3 demonstrates the procedure for assembling in background. A batch job is created and then submitted to background (batch).

OR

Item 4 shows the procedure for assembling the sample program in foreground (timesharing) and the listing of the program generated by the assembler.

Item 5 explains the simulator dialog and a step by step simulation of the sample program.

Item 6 describes the technique to punch a paper tape of the Object Tape File. This tape is compatible with the Motorola EXORciser.TM

CREATE A SAMPLE PROGRAM

L?T61

```
UCS 05/01/76. 11.59.38. T144
USER NUMBER: M437XXX
MMMMMM
*RDY-FOR*
NEW, PGM
*RDY-FOR*
AUTO
00100 NAM PGM
00110 OPT M=MEMF1 SPECIFY MACHINE FILENAME
00120 OPT O=TAPEF1 OBJECT TAPE FILENAME
00130 OPT SYMBOL SELECT PRINTING OF SYMBOLS
00140 ORG 256
00150 COUNT EQU 3
00160 START LDS #STACK INZ STACK POINTER
00170 LDH ADDR
00180 LDA B #COUNT IMMEDIATE ADDRESSING
00190 BACK LDA A 10 DIRECT ADDRESSING
00200 CMP A 2, X INDEXED ADDRESSING
00210 BEQ FOUND RELATIVE ADDRESSING
00220 DEX IMPLIED ADDRESSING
00230 DEC B ACCUMULATOR ONLY ADDRESSING
00240 BNE BACK
00250 WAI WAIT FOR INTERRUPT
00260 SPC 1
00270 FOUND JSR SUBRTH JUMP TO SUBROUTINE
00280 JMP START EXTENDED ADDRESSING
00290 * COMMENT STATEMENT NOTE TRUNCATION 01234567890123456789
00300 SUBRTH TAB COMMENT FIELD TRUNCATION0123456789
00310 ADDA BYTE SET MOST SIGNIFICANT BIT
00320 RTS RETURN FROM SUBROUTINE
00330 SPC 2
00340 RMB 20 SCRATCH AREA FOR STACK
00350 STACK RMB 1 START OF STACK
00360 BYTE FCB $80 FORM CONSTANT BYTE
00370 FCB $10, $4 $ INDICATES HEXADECIMAL
00380 ADDR FDB DATA FORM CONSTANT DOUBLE BYTE
00390 DATA FCC /SET/ FORM DATA STRING (ASCII)
00400 END
00410 NON
00420 *DEL*
SAV
*RDY*
310 ORAA BYTE SET MOST SIGNIFICANT BIT
105 * REVISION 1
RES
*RDY-FOR*
REP
*RDY*
```

Enter response so computer can determine your terminal's speed.

if 10 CPS enter ?61
if 15 CPS enter 861
if 30 CPS enter T61

UCS log-on sequence where XXX is your assigned user number.

Enter your password.

FORTTRAN system automatically assigned.

Create new file with filename "PGM."

Ready indicates system is ready to accept data or command.

NOTE: The line number includes the first space following the number; allow for this space character while entering the program.

Automatic line number assignment.

The first record should be a NAM assembler directive; the first six characters of operand will appear in the assembler listing header.

The ORG assembler directive sets the program counter.

Enter the program: only one space between a line number and a label; otherwise the assembler accepts one, or more spaces separating the fields.

The END assembler directive informs the assembler this is the last record of this assembly.

The MON assembler directive informs the assembler this is the last file to be assembled.

Escape key, delete key, or control-x cause exit from auto mode.

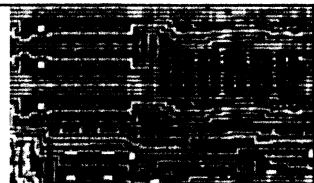
SAV is the command to save the new file just created.

Examples of program changes; by overwriting and by inserting a new line.

RES is the command to resequence the program starting with line number 100 and sequencing by 10.

REP is the command to replace (or update) an existing file.

4 ASSEMBLE IN FOREGROUND



```
EXE, OLD, MPCASH(M437***) }
*RDY-EXE*
RUN, M=24000
```

Call the cross assembler and cause it to run.

```
05/03/76 1.32.14
PROGRAM MPCASH
```

```
MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPCASH
COPYRIGHT 1974 BY MOTOROLA INC
```

The release number is changed as the cross assembler is updated with improvements.

```
MOTOROLA MPU CROSS ASSEMBLER, RELEASE 1.4A
```

```
ENTER SI FILENAME
? PGH
FILE'S LABEL:
SOURCE FILENAME: PGH
AUTHOR JOHN DOE
---
```

Enter SI (source input) filename of the program to be assembled.

The contents of the Label Buffer Area from the machine file MEMF1.

```
1 PAGE 1 PGH 05/03/76 13:32.14
```

Line number.

Program counter (hexadecimal).

Hexadecimal instruction, data, or value.

```
00100          * REVISION 1
00110          OPT M-MEMF1 SPECIFY MACHINE FILENAME
00120          OPT O=TAPEF1 OBJECT TAPE FILENAME
00130          OPT SYMBOL SELECT PRINTING OF SYMBOLS
00140          OPT
00150 0100     ORG 256
00160          EQU 3
00170 0100 0E 0132 START LBS #STACK INZ STACK POINTER
00180 0103 FE 0136       LDX ADDR
00190 0106 C6 03       LDA B #COUNT IMMEDIATE ADDRESSING
00200 0100 96 0A     BACK LDA A 10 DIRECT ADDRESSING
00210 010A A1 02     CMP A 2,X INDEXED ADDRESSING
00220 010C 27 05     BEQ FOUND RELATIVE ADDRESSING
00230 010E 09       BEX IMPLIED ADDRESSING
00240 010F 5A       DEC B ACCUMULATOR ONLY ADDRESSING
00250 0110 26 F6     BNE BACK
00260 0112 3E       WAI WAIT FOR INTERRUPT
```

An asterisk (*) may be used as the first character of a comment statement.

The # indicates immediate addressing.

```
00280 0113 BD 0119 FOUND JSR SUBRTH JUMP TO SUBROUTINE
00290 0116 7E 0100     JMP START EXTENDED ADDRESSING
00300          * COMMENT STATEMENT NOTE TRUNCATION 01234567890123
00310 0119 16     SUBRTH TAB COMMENT FIELD TRUNCATION 0123
00320 011A 8A 0133 ORA A BYTE SET MOST SIGNIFICANT BIT
00330 011D 39     RTS RETURN FROM SUBROUTINE
```

The missing line 270 was a SPC 1 assembler directive.

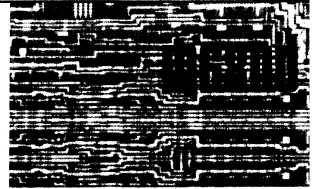
```
00350 011E 0014     RMB 20 SCRATCH AREA FOR STACK
00360 0132 0001     STACK RMB 1 START OF STACK
00370 0133 00     BYTE FCB $00 FORM CONSTANT BYTE
00380 0134 10     FCB $10.44 $ INDICATES HEXADECEIMAL
00390 0135 04
00390 0136 0130 ADDR FDB DATA FORM CONSTANT DOUBLE BYTE
00400 0138 53 DATA FCC /SET/ FORM DATA STRING (ASCII)
00410 0139 45
00410 013A 54
00410          END
```

The \$ indicates a hexadecimal value follows.

SYMBOL TABLE

```
ADDR 0136 BACK 0100 BYTE 0133 COUNT 0003 DATA 0130
FOUND 0113 STACK 0132 START 0100 SUBRTH 0119
```

NOTE: For more detailed information as to specific meaning of mnemonics and the details of each program refer to the M6800 MICROPROCESSOR PROGRAMMING MANUAL.



6 SIMULATE THE SAMPLE PROGRAM

```
EXE, OLD, MPSSIM(M437***)
*RDY-EXE*
RUN, M=24000
```

Call the simulator and cause it to run.

```
05/03/76. 16.25.55.
PROGRAM MPSSIM
```

Enter the MF (memory file) filename of the memory file to be simulated.

```
MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPSSIM
COPYRIGHT 1975 BY MOTOROLA INC
```

The contents of the Label Buffer Area from the machine file MEMF1.

```
MOTOROLA MPU SIMULATOR, RELEASE 1.3A
```

Register heading:

- HH Input the output base hexadecimal
- IA Instruction address
- OC Operator mnemonic code
- EA Effective operand address
- P Program counter
- X Index register
- A Accumulator A
- B Accumulator B
- C Condition code register
- S Stack pointer
- T Time cycles (always decimal)

```
ENTER MF FILENAME
? MEMF1
```

```
FILE'S LABEL:
SOURCE FILENAME: PGM
AUTHOR JOHN DOE
```

```
HH IA OC EA P X A B C S T
0000 *** 0000 0000 0000 00 00 000000 0000 00000000
? SM 0A,54. SR P 100. T OC
*0100 LDS *0102*0103 0000 00 00 000000*0132 00000003
*0103 LDX *0137*0106*0138 00 00 000000 0132 00000008
*0106 LDA B*0107*0108 0138 00*03 000000 0132 0000010
*0108 LDA A*000A*010A 0138*54 03 000000 0132 0000013
*010A CMP A*013A*010C 0138 54 03 000200 0132 0000018
*010C BEQ *010D*0113 0138 54 03 000200 0132 0000022
*0113 JSR *0131*0119 0138 54 03 000200*0130 0000031
*0119 TAB *0119*011A 0138 54*54 000000 0130 0000033
*011A ORA A*0133*011D 0138*D4 54 00N000 0130 0000037
HH IA OC EA P X A B C S T
*011D RTS *0132*0116 0138 D4 54 00N000*0132 0000042
*0116 JMP *0118*0100 0138 D4 54 00N000 0132 0000045
*0100 LDS *0102*0103 0138 D4 54 000000 0132 0000048
```

Simulator commands separated by period
SM 0A,54 Set memory location A to contain 54

SR P 100 Set register (Program Counter) equal 100
T OC Trace C instructions

Simulator command Display Memory; beginning with location 100, display 3B (hex) bytes (note the right margin contains the literal equivalent of the printable characters; the periods show nonprintable characters).

```
? DM 100,3B
0100 8E 01 32 FE 01 36 C6 03 96 0A A1 02 27 05 09 5A ... 2...6..... Z
0110 26 F6 3E 0D 01 19 7E 01 00 16 BA 01 33 39 00 00 ... 4...>..... 39.
0120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0130 00 01 16 00 10 04 01 38 53 45 54 ..... 8SET
```

Simulator commands: RS restore registers; D display registers.

```
? RS, D
0000 *** 0000'0000 0000 00 00 000000 0000 00000000
```

Simulator command EX exit simulator

```
? EX
```

NOTE: Hexadecimal input to the simulator requires the first character be numeric (i.e.: to enter the hex. value "C" enter "0C")

6 PUNCH A PAPER TAPE

```
OLD, TAPEF1
*RDY-FOR*
LNH
S0060000040445210
S11301000E0132FE0136C603960AA1022705095A5A
S111011026F63E0D01197E010016BA013339F0
S10B0133801004013853455407
S9030000FC
*RDY*
BYE
CT = HH, 88
08KCPU = T, TT
16KCPU = T, TT
24KCPU = T, TT
M437XXX LOG OFF. 17.38.42.
```

OLD, TAPEF1 calls the formatted tape image so it may be punched and listed.

List the Object Tape File TAPEF1 without heading. Turn on punch device before entering carriage return.

S0... indicates a header record

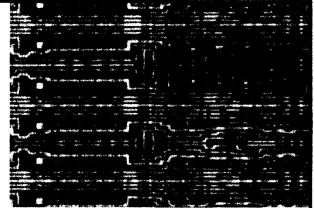
S1... indicates a data record

S9... indicates an end-of-file record

Sign-off system (enter goodBYE)

Accounting statistics for the current session.

Once a machine file has been created and configured the Build Virtual Machine program need not be run until the configuration needs changing.



2 CREATE A MACHINE FILE

```

EXE, OLD, MPBVM(M437***) }
*RDY-EXE*
RUN, M=24800 }
05/01/76. 12.43.10. }
PROGRAM MPBVM }
MOTOROLA SPB, INC. OWNS AND IS RESPONSIBLE FOR MPBVM
COPYRIGHT 1974 BY MOTOROLA INC
MOTOROLA MPU BUILD VIRTUAL MACHINE, RELEASE 1.4A
ENTER 'HP HP 0' FOR MORE HELP
? MF MEMF1
***ATTN, 92
FILE'S LABEL:
DEFAULT VIRTUAL MACHINE FILE (4K OF MEMORY)
? LW 01FF
? TI
ENTER TITLE TEXT
? SOURCE FILENAME, PGM
ENTER TITLE TEXT
? AUTHOR JOHN DOE
ENTER TITLE TEXT
?
? NO
VIRTUAL MACHINE FILE MEMF1
FILE'S LABEL:
SOURCE FILENAME, PGM
AUTHOR JOHN DOE
LAST WORD ADDRESS 1FF
MACRO LIBRARY LISTING
1072 REMAINING CHARACTERS
? EX

```

Call the Build Virtual Machine program and cause it to run.

Date and Time provided by U.C.S. system.

Name of program running.

Fetch the machine file named MEMF1.

Indication that the machine file MEMF1 did not exist and that a file was created assuming the default parameters.

The contents of the default Label Buffer Area.

Change this machine file's size by setting a new Last Word address (hexadecimal value 1FF was entered).

Set new information into the Label Buffer Area for this machine file (MEMF1).

Display the Machine File Organization.

EX is the command to exit the Build Virtual Machine program.

3 ASSEMBLE IN BACKGROUND

```

NEW, CONTROL
*RDY-FOR*
AUTO
00100 JOB.
00110 ACCOUNT, M437XXX.
00120 GET, MPCASH(M437***)
00130 RFL, 70000.
00140 MPCASH.
00150 GOEXIT.
00160 EXIT.
00170 SAVE, OUTPUT=OUTFILE.
00180 DFD, DAYFIL.
00190 SAVE, DAYFIL.
00200 EOR.
00210 PGM
00220 EOF.
00230 *DEL*
SAVE
*RDY*
RJE
RJE COMPLETE, ID = RJEJJJ
*RDY*

```

Create new file with filename "CONTROL."

Automatic line number assignment.

Job and account card for batch job "CONTROL." XXX is your assigned user number.

Get the cross assembler program and cause it to run.

Direct assembler listings to a "new" file called "OUTFILE" and SAVE it.

File DAYFIL contains the system operating messages associated with job processing.

Name of M6800 assembly language source file.

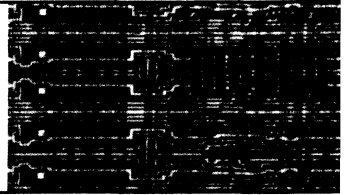
Escape key, delete key, or control-x must be hit to escape auto mode.

Save file "CONTROL."

Submit batch job "CONTROL" to background.

System response to RJE command—JJJ is the job-id assigned to this job.

LANGUAGE OF THE M6800 MICROPROCESSOR



MICROPROCESSOR INSTRUCTION SET ALPHABETIC SEQUENCE

ABA	Add Accumulators
ADC	Add with Carry
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to Zero
BGE	Branch if Greater or Equal Zero
BGT	Branch if Greater than Zero
BHI	Branch if Higher
BIT	Bit Test
BLE	Branch if Less or Equal
BLS	Branch if Lower or Same
BLT	Branch if Less than Zero
BMI	Branch if Minus
BNE	Branch if Not Equal to Zero
BPL	Branch if Plus
BRA	Branch Always
BSR	Branch to Subroutine
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CBA	Compare Accumulators
CLC	Clear Carry
CLI	Clear Interrupt Mask
CLR	Clear
CLV	Clear Overflow
CMP	Compare
COM	Complement
CPX	Compare Index Register
DAA	Decimal Adjust
DEC	Decrement
DES	Decrement Stack Pointer
DEX	Decrement Index Register
EOR	Exclusive OR
INC	Increment
INS	Increment Stack Pointer
INX	Increment Index Register
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator
LDS	Load Stack Pointer
LDX	Load Index Register
LSR	Logical Shift Right
NEG	Negate
NOP	No Operation
ORA	Inclusive OR Accumulator
PSH	Push Data
PUL	Pull Data
ROL	Rotate Left
ROR	Rotate Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBA	Subtract Accumulators
SBC	Subtract with Carry
SEC	Set Carry
SEI	Set Interrupt Mask
SEV	Set Overflow
STA	Store Accumulator
STS	Store Stack Register
STX	Store Index Register
SUB	Subtract
SWI	Software Interrupt
TAB	Transfer Accumulators
TAP	Transfer Accumulators to Condition Code Reg.
TBA	Transfer Accumulators to Accumulator
TPA	Transfer Condition Code Reg. to Accumulator
TST	Test
TSX	Transfer Stack Pointer to Index Register
TXS	Transfer Index Register to Stack Pointer
WAI	Wait for Interrupt

INSTRUCTION ADDRESSING MODES AND ASSOCIATED EXECUTION TIMES

(in microseconds assuming a 1 MHz clock)

	(Dual Operand)	Execution Times						
		ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative
ABA		*	*	*	*	*	*	*
ADC x		*	2	3	4	5	*	*
ADD x		*	2	3	4	5	*	*
AND x		*	2	3	4	5	*	*
ASL		*	2	*	6	7	*	*
ASR		*	2	*	6	7	*	*
BCC		*	*	*	*	*	4	*
BCS		*	*	*	*	*	4	*
BEQ		*	*	*	*	*	4	*
BGE		*	*	*	*	*	4	*
BGT		*	*	*	*	*	4	*
BHI		*	*	*	*	*	4	*
BIT x		*	2	3	4	5	*	*
BLE		*	*	*	*	*	4	*
BLS		*	*	*	*	*	4	*
BLT		*	*	*	*	*	4	*
BMI		*	*	*	*	*	4	*
BNE		*	*	*	*	*	4	*
BPL		*	*	*	*	*	4	*
BRA		*	2	3	4	5	*	*
BSR		*	*	*	*	*	8	*
BVC		*	*	*	*	*	4	*
BVS		*	*	*	*	*	4	*
CBA		*	*	*	*	*	2	*
CLC		*	*	*	*	*	2	*
CLI		*	*	*	*	*	2	*
CLR		2	*	*	6	7	*	*
CLV		*	*	*	*	*	2	*
CMP x		*	2	3	4	5	*	*
COM		2	*	*	6	7	*	*
CPX		*	3	4	5	6	*	*
DAA		*	*	*	*	*	2	*
DEC		2	*	*	6	7	*	*
DES		*	*	*	*	*	4	*
DEX		*	*	*	*	*	4	*
EOR x		*	2	3	4	5	*	*
INC		2	*	*	6	7	*	*
INS		*	*	*	*	*	4	*
INX		*	*	*	*	*	4	*
JMP		*	*	*	*	*	8	*
JSR		*	*	*	*	*	9	*
LDA x		*	2	3	4	5	*	*
LDS		*	3	4	5	6	*	*
LDX		*	3	4	5	6	*	*
LSR		2	*	*	6	7	*	*
NEG		2	*	*	6	7	*	*
NOP		2	*	*	6	7	*	*
ORA x		*	2	3	4	5	*	*
PSH		4	*	*	*	*	*	*
PUL		4	*	*	*	*	*	*
ROL		2	*	*	6	7	*	*
ROR		2	*	*	6	7	*	*
RTI		*	*	*	*	*	10	*
RTS		*	*	*	*	*	5	*
SBA		*	2	3	4	5	*	*
SBC x		*	2	3	4	5	*	*
SEC		*	*	*	*	*	2	*
SEI		*	*	*	*	*	2	*
SEV		*	*	*	*	*	2	*
STA x		*	4	5	6	*	*	*
STS		*	*	5	6	7	*	*
STX		*	*	5	6	7	*	*
SUB x		*	2	3	4	5	*	*
SWI		*	*	*	*	*	12	*
TAB		*	*	*	*	*	2	*
TAP		*	*	*	*	*	2	*
TBA		*	*	*	*	*	2	*
TPA		*	*	*	*	*	2	*
TSX		2	*	*	6	7	*	*
TXS		*	*	*	*	*	4	*
TXS		*	*	*	*	*	4	*
WAI		*	*	*	*	*	9	*

LIST OF ASSEMBLER DIRECTIVES

END	End of Program
EQU	Equate Symbol
FCB	Form Constant Byte
FCC	Form Constant Characters
FDB	Form Double Constant Byte
MON	Return to Console
NAM	Name
OPT	Option
ORG	Origin
PAGE	Top of Form
RMB	Reserve Memory Byte
SPC	Space Lines

ACCX (accumulator only) Addressing

In accumulator only addressing, either accumulator A or accumulator B is specified. These are one-byte instructions.

Immediate Addressing

In immediate addressing, the operand is contained in the second byte of the instruction. No further addressing of memory is required. The MPU addresses this location when it fetches the immediate instruction for execution. These are two/three-byte instructions.

Direct Addressing

In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes in the machine; i.e., locations zero through 255. That part of the memory should be used for temporary data storage and intermediate results. In most configurations, it should be a random access memory. These are two-byte instructions.

Extended Addressing

In extended addressing, the value contained in the second byte of the instruction is used as the higher eight-bits of the address of the operand. The third byte of the instruction is used as the lower eight-bits of the address of the operand. This gives one a 16-bit address for the operand. This is an absolute address in memory. These are three-byte instructions.

Indexed Addressing

In indexed addressing, the value contained in the second byte of the instruction is added to the index register lower eight-bits in the MPU. The carry is then added to the higher order eight-bits of the index register. This result is then used to address memory. The modified address is held in a temporary address register so there is no change to the index register. These are two-byte instructions.

Implied Addressing

In the implied addressing mode the instruction gives the address (i.e., stack pointer, index register, etc.). These are one-byte instructions.

Relative Addressing

In relative addressing, the value contained in the second byte of the instruction is added to the program counters lowest eight-bits plus two. The carry or borrow is then added to the high eight-bits. This allows the user to address data within a range of -126 to +129 bytes of the present instruction. These are two-byte instructions.



MOTOROLA
Semiconductor Products Inc.

8005 EAST McDOWELL ROAD, PHOENIX, ARIZONA 85008



M6800

SUPPORT SOFTWARE

PROGRAMMABLE LOGIC — *the easy way*

Motorola's MPL Compiler for the M6800 microcomputer family is currently available on United Computing's Multiple Access Remote Computing Service.

TO ACCESS THE SOFTWARE:

1. Contact your local UCS sales representative and request service for Motorola M6800 Software System under user catalog M437.
2. If you are a new user also request the UCS System Guide and the UNEDIT (Editor) manuals from the UCS sales representative. You will also need to obtain the appropriate telephone numbers to access UCS's time-sharing service.
3. For detailed programming and support software information order your copy of the "MPL Language Reference Manual" from Motorola Semiconductor Literature Distribution Center, P.O. Box 20924, Phoenix, Arizona 85036.
4. Sign on with your teletype (or other terminal) and you will be up and running.



MOTOROLA Semiconductor Products Inc.

M6800 SUPPORT SOFTWARE



The sample program displayed on this and the following pages used the UCS Timesharing system to give the new user a capsule view of the procedure for using Motorola's MPL Compiler.

Item 1 describes preparation for the sample program using the edit features of timesharing.

Item 2 shows the conversation to compile the sample program in timeshare mode.

Item 3 shows the conversation to assemble the output from the compilation, and a partial listing of the program generated by the assembler.

Item 4 describes the preparation of a control file for executing "M68MPL" in remote job entry (RJE) mode.

1 CREATE A SAMPLE PROGRAM

L??761

```

UCS 05/14/76. 16.28.02 L051
USER NUMBER:M437XXX
MMMMMM
♦RDY-FOR♦
NEW:MPLTST1
♦RDY-FOR♦
AUTO
00100 $ OPT MF=MTEST,NOP
00110 ♦
00120 SAMPLE MPL PROGRAM
00130 ♦
00140 ♦♦START THE MAIN PROGRAM.♦♦
00150 DCL ASIZ,BB(10), B(2) INIT(6000,333,300,4,99,5403,666,44,32,24);
00160 DCL C(5), B(2) INIT("99","F0","FFF0","300","66");
00170 ORIGIN "100"
00180 PROCEDURE OPTIONS(MAIN)
00190 ASIZ=10
00200 ♦♦CALL SORT AND SEND ADDRESS AND SIZE OF ARRAY (BB) WITH CALL.♦♦
00210 CALL SORT (<ASIZ,<BB>);
00220 ♦♦EXAMPLE OF IN-LINE CODE♦♦
00230 $ LDA A,BB
00240 $ STA A,ASIZ
00250 ♦♦CALL SORT AND SEND ADDRESS AND SIZE OF ARRAY (C) WITH CALL.♦♦
00260 CALL SORT (<ASIZ,<C>);
00270 ♦♦ STOP PROGRAM EXECUTION WITH BRANCH TO SELF ♦♦
00280 STOP: GO TO STOP;
00290 END
00300 ♦♦ SUBROUTINE SORT ♦♦
00310 SORT: PROCEDURE(<NN,RESETP>)
00320 DCL RA B(2) BASED,PTR B(2), PTR2 B(2);
00330 DCL I B(2),SWITCH,TEMP B(2),RESETP B(2);
00340 DCL NN;
00350 PTR2=RESETP+NN;
00360 S1: SWITCH=2
00370 DO I = RESETP TO PTR2 BY 2;
00380 PTR=I+2;
00390 IF RA:I LE RA:PTR THEN GO TO S2;
00400 SWITCH = 1;
00410 TEMP = RA:I;
00420 RA:I=RA:PTR;
00430 RA:PTR = TEMP;
00440 S2: END
00450 GO TO (<S1,<S3>,<SWITCH
00460 S3: RETURN
00470 ♦♦ NOTHING WILL BE LISTED WHEN ASSEMBLED
00480 AFTER THIS STATEMENT ♦♦
00490 $ OPT NOL
00500 END
00510 ♦DEL♦
SAV
    
```

Enter response so computer can determine your terminal's speed.

if 10 CPS enter ?61
if 15 CPS enter 861
if 30 CPS enter T61

UCS log-on sequence where XXX is your assigned user number.

Enter your password.

FORTRAN system automatically assigned.

Create new file with filename "MPLTST1."

Ready indicates system is ready to accept data or command.

NOTE: The line number includes the first space following the number; allow for this space character while entering the program.

Automatic line number assignment.

Escape key will exit from auto mode.

SAV is the command to save the new file just created.

2 COMPILE THE SAMPLE PROGRAM

```
EXE,OLD,M68MPL(M437◆◆◆)
◆RDY-EXE◆
RUN,M=24000
```

Call the Compiler and initiate execution.

```
05/14/76. 15.51.02.
PROGRAM M68MPL
```

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR M68MPL
COPYRIGHT 1975 AND 1976 BY MOTOROLA INC.

MOTOROLA'S M6800 MPL COMPILER, RELEASE 1.2

The release number is changed as the Compiler is updated with improvements.

```
ENTER SI FILENAME
? MPLTST1
ENTER OT FILENAME
? MPLOT
◆RDY-EXE◆
```

Enter SI (Source Input) filename of program to be compiled.

Enter OT (Output) filename. Compiler output will automatically be saved.

3 ASSEMBLE THE SAMPLE PROGRAM

```
EXE,OLD,MPCASM(M437◆◆◆)
◆RDY-EXE◆
RUN,M=24000
```

Call the Cross Assembler and initiate execution.

```
05/14/76. 15.53.49.
PROGRAM MPCASM
```

1---

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPCASM
COPYRIGHT 1974 BY MOTOROLA INC

MOTOROLA MPU CROSS ASSEMBLER, RELEASE 1.4A

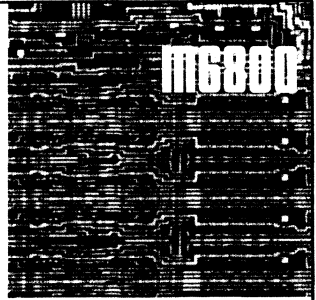
The release number is changed as the Cross Assembler is updated with improvements.

```
ENTER SI FILENAME
? MPLOT
FILE'S LABEL:
DEFAULT VIRTUAL MACHINE FILE (4K OF MEMORY)
---
```

Enter filename of the program to be assembled. This filename is the output from the compilation in Item 2 above.

1 PAGE 1 MPLOT 05/14/76 15:54.02

```
00001          NAM      MPLOT
00002          ◆◆◆      COMPILED WITH MPL VERSION 1.2
00003          OPT      MF=MFTST,NOP
00004          ◆00110  ◆/
00005          ◆00120  ◆ SAMPLE MPL PROGRAM
00006          ◆00130  ◆/
```



4 COMPILATION IN RJE MODE

```
NEW, CONTROL _____ Create JCL file with filename "CONTROL."  
READY - FOR!  
AUTO _____ Automatic line number assignment.  
00100 JOB.  
00110 ACCOUNT,M437???.  
00120 GET,M68MPL(M437◆◆◆)  
00130 RFL,70000.  
00140 M68MPL.  
00150 GOEXIT.  
00160 EXIT.  
00170 SAVE,OUTPUT=OUTFILE.  
00180 DFD,DAYFIL.  
00190 SAVE,DAYFIL.  
00200 EOR.  
00210 SOURCE  
00220 EOF.  
00230 ◆DEL◆ _____ Escape key will exit from auto mode.  
SAVE _____ Save file just created.  
◆RDY◆  
RJE _____ Submit file to the batch input queue.  
RJE COMPLETE, ID = RJE3AVN  
◆RDY◆
```

◆◆◆◆ NOTES ◆◆◆◆

LINE NO.	COMMENTS
00110	"M437???" SHOULD BE REPLACED WITH YOUR USER NUMBER
00170	"OUTFILE" IS THE FILENAME YOU WANT TO CONTAIN YOUR OUTPUT
00180 00190	"DAYFIL" IS THE FILENAME YOU WANT TO CONTAIN YOUR RUN STATS
00210	"SOURCE" IS THE SOURCE INPUT FILENAME
00230	ESCAPE KEY, DELETE KEY, OR CONTROL X MUST BE HIT TO ESCAPE AUTO MODE

LANGUAGE OF THE M6800 MPL COMPILER



ARITHMETIC ASSIGNMENT STATEMENT

General Form

$$a = b$$

Type of b	Type of a		
	BIN	DEC	CHAR
BIN	Assign.	Convert to numeric ASCII and assign.	Convert to numeric ASCII with zero suppression and assign, right justified, blank filled on left.
DEC	Convert to binary and assign.	Assign.	Zero suppress and assign.
CHAR	Not allowed.	Not allowed.	Assign. a > b Left justify, blank fill a. b > a Truncate b on the right and assign.

Forms in square brackets are optional. The following abbreviations are allowed:

DCL – DECLARE CHAR – CHARACTER
 BIN – BINARY DEF – DEFINED
 DEC – DECIMAL INIT – INITIAL

LOGICAL OPERATORS

IAND – IEOR – IOR – AND – OR – EQ – GT – GE –
 LT – LE – NE

ORIGIN

ORIGIN "HEX CONSTANT"

POINTER

VARIABLE:POINTER or POINTER → VARIABLE

PROCEDURE

PROCEDURE OPTIONS (MAIN)
 PROCEDURE OPTIONS (MAIN, STACK NAME)

SHIFT

W SHIFT k
 +k left shift
 -k right shift

SUBROUTINES

CALL LABEL or CALL LABEL (arg 1, . . . , arg n)
 LABEL: PROCEDURE or LABEL: PROCEDURE
 (arg 1, . . . , arg n)

or

CALL LABEL <a₁, a₂, a₃>
 LABEL: PROCEDURE <a₁, a₂, a₃>

CONTROL STATEMENTS

DO i = m₁ TO m₂ [BY m₃]
 DO WHILE Boolean expression
 DO i = m₁ TO m₂ [BY m₃] WHILE Boolean expression
 GO TO label (unconditional)
 GO TO label (label assigned in DECLARE)
 GO TO (x, x₁, . . . , x_n), i or
 GO TO labelname (i)
 IF a THEN s₁ [ELSE s₂]

DATA REPRESENTATION

BIT (i) – BINARY (1) – BINARY (2) – DECIMAL
 (m, n) – SIGNED DECIMAL (m, n) – CHARACTER
 (m)

DECLARE

The general form is:

DECLARE

[level #] name [(occurrence)]	BIT BINARY DECIMAL SIGNED DECIMAL CHARACTER LABEL	[(m)] [(m,n)]	[DEFINED name]
----------------------------------	---	------------------	-------------------

[BASED] [INITIAL (value 1, value 2 . . .)]



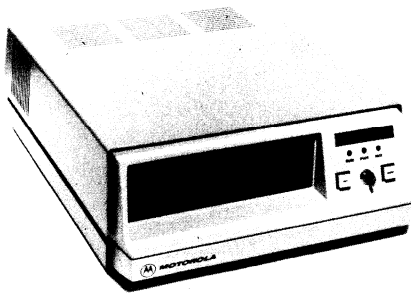
MOTOROLA
Semiconductor Products Inc.

5005 EAST McDOWELL ROAD, PHOENIX, ARIZONA 85008

M6800

SUPPORT SOFTWARE

EXORciser



PROGRAMMABLE LOGIC — *the easy way*

Motorola software for the M6800 microcomputer family is currently available for the EXORciser, Motorola's microcomputer system development tool:

RESIDENT EDITOR

The M6800 Resident Editor gives the user an easy means to create and modify source files for input to the Assembler. The interactive Resident Editor offers character, line, and character string commands.

RESIDENT ASSEMBLER

The M6800 Resident Assembler converts symbolic source code to M6800 machine-language with formatted listing. The Resident Assembler is compatible with Cross Assemblers provided by Motorola.

EXbug

The M6800 EXbug firmware provides the utility programs to load and debug programs for the Motorola MC6800 Microprocessor. EXbug includes many of the features found in the M6800 Interactive Simulator available for larger computers.

For detailed programming and support software information order your copy of the "M6800 Microprocessor Programming Manual" and the "M6800 EXORciser Resident Software Supplement" to the "M6800 EXORciser User's Guide."

Motorola Semiconductor Literature Distribution Center, P. O. Box 20924, Phoenix, Arizona 85036.

SYSTEM REQUIREMENTS

- EXORciser
- 8k bytes of RAM
- Terminal with RS-232 or TTY (20mA neutral loop current) interface and automatic reader/punch control

PROGRAMS AVAILABLE ON:

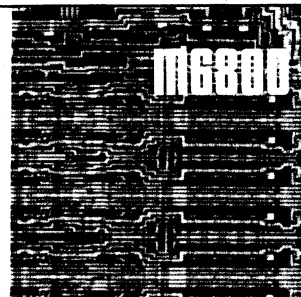
- Paper tape
- Cassette
- GE Timesharing File (no charge for access)



MOTOROLA Semiconductor Products Inc.

EXORciser and EXbug are trademarks of Motorola Inc.

M6800 SUPPORT SOFTWARE



The sample program shown here was developed on the EXORciser to give the new user a capsule view of the procedure for using the resident support software.

Item **1** describes the preparation of the sample program using the resident text editor.

Item **2** shows the procedure to load the resident assembler, the conversation to assemble the sample program, and the listing of the program generated by the assembler.

Item **3** explains the format of the Object Tape generated by the assembler and the loading of the Object Tape into an EXORciser.

Item **4** demonstrates how the user can test the sample program with the trace feature of the EXbug firmware.

O CREATE A SAMPLE PROGRAM

```
EXBUG 1.1 LOAD _____ Use the EXbug Loader to load the Editor program
$GL/CONT S _____ S--Load only a single file
HDR _____ Contents of header record are printed
EXBUG 1.1 MAID } _____ Call the editor and cause it to run (NNNN is the
♦NNNN:6 } _____ beginning address of the editor program)

M6800 EDITOR VERSION X.X _____ @Indicates editor ready for input
                                           B ESCAPE ESCAPE (Escape key echos as $) Positions
                                           workspace character pointer to beginning of workspace
                                           buffer
$B$$$ _____ I sets editor to input mode
$I NAM PGM _____ The first record should be a NAM assembler directive;
♦ REVISION 1 _____ the first six characters of operand will appear in the
  OPT 0 OUTPUT OBJECT TAPE _____ assembler listing header.
  OPT 3 SELECT PRINTING OF SYMBOLS _____
  ORG 256 _____ The ORG assembler directive sets the program counter.
COUNT EQU 03 $ INDICATES OCTAL
START LDS #STACK INZ STACK POINTER
LDX ADDR
LDA B #COUNT IMMEDIATE ADDRESSING
BACK LDA A 10 DIRECT ADDRESSING
CMP A 2,X INDEXED ADDRESSING
BEQ FOUND RELATIVE ADDRESSING
DEY IMPLIED ADDRESSING
DEC B ACCUMULATOR ONLY ADDRESSING
BNE BACK
WAI WAIT FOR INTERRUPT
SPC 1
FOUND JSR SUBRTN JUMP TO SUBROUTINE
JMP START EXTENDED ADDRESSING
♦ COMMENT STATEMENT NOTE TRUNCATION 01234567890123456789
SUBRTN TAB COMMENT FIELD TRUNCATION0123456789
ORA A BYTE SET MOST SIGNIFICANT BIT
RTS RETURN FROM SUBROUTINE
SPC 2
RMB 20 SCRATCH AREA FOR STACK
STACK RMB 1 START OF STACK
BYTE FCB #80 FORM CONSTANT BYTE
FCB $10,$4 $ INDICATES HEXADECIMAL
ADDR FDB DATA FORM CONSTANT DOUBLE BYTE
DATA FDC 'SET' FORM CONSTANT DATA STRING (ASCII)
END _____ The END assembler directive informs the assembler this
MON _____ is the last record of this assembly.
$$$ _____ The MON assembler directive informs the assembler this
$B$$$ _____ is the last file to be assembled.
                                           ESCAPE ESCAPE (Escape key echos as $) terminate
                                           input mode
                                           B--Set pointer to beginning of workspace
                                           E--Punch contents of workspace and exit editor program
```



2 ASSEMBLE THE SAMPLE PROGRAM

```

EXBUG 1.1 LOAD }
SGL/CDNT $    } LOAD the Resident Assembler
HDR          }
EXBUG 1.1 MAID }
*NNNN:G      } Call the resident assembler and cause it to
               } run.

```

```

M6800 ASSEMBLER VERSION X.XX
ENTER PASS: 1P,1S,2P,2L,2T

```

The release number is changed as the resident assembler is updated with improvements.

```

1P
M6800 ASSEMBLER VERSION X.XX
ENTER PASS: 1P,1S,2P,2L,2T

```

Start pass one

```

2P

```

Start pass two

```

---
PAGE 001 PGM
00001
00002
00003
00004
00005 0100
00006 0003
00007 0100 8E 0132
00008 0103 FE 0136
00009 0106 C6 03
00010 0108 96 0A
00011 010A A1 02
00012 010C 27 05
00013 010E 09
00014 010F 5A
00015 0110 26 F6
00016 0112 3E
00018 0113 BD 0119
00019 0116 7E 0100
00020
00021 0119 16
00022 011A BA 0133
00023 011D 39
00025 011E 0014
00026 0132 0001
00027 0133 80
00028 0134 10
00029 0135 04
00029 0136 0138
00030 0138 53
00030 0139 45
00030 013A 54
00031
COUNT 0003
START 0100
BACK 0108
FOUND 0113
SUBRTN 0119
STACK 0132
BYTE 0133
ADDR 0136
DATA 0138
END
TOTAL ERRORS 00000

```

Line number (assigned by Assembler)

Program counter (hexadecimal).

Hexadecimal instruction, data, or value.

An asterisk (*) may be used as the first character of a comment statement.

The # indicates immediate addressing.

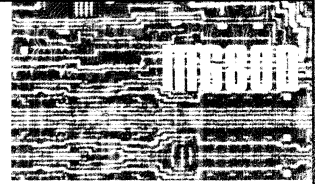
The missing line 17 was a SPC 1 assembler directive.

The \$ indicates a hexadecimal value follows.

NOTE: For more detailed information as to specific meaning of mnemonics and the details of each program refer to the M6800 MICROPROCESSOR PROGRAMMING MANUAL.

3

LOAD a PAPER TAPE



The Object Tape generated by the resident assembler is in the following format:

```

S006000048445218 _____ S0 ... indicates a header record
S11E01008E0132FE0135C5039509A1022705095A25F63EED01137E010015BACF }
S105011B01333970 _____ S1 ... indicates a data record
S10E0133801004013853455407 _____
S9030000FC _____ S9 ... indicates an end-of-file record

```

```

EXBUG 1.1 LOAD }
SGLCONT 3 _____ LOAD the sample program
HDR
EXBUG 1.1 PRINT }
REG ADDR 0000 100 _____ The printed dump feature of EXbug is used to list the
END ADDR 0000 13A _____ hexadecimal data loaded
EXEC Y

```

```

0100 8E 01 32 FE 01 35 C5 03 95 09 A1 02 27 05 09 5A ...2..6F...!..2
0110 26 F6 3E ED 01 13 7E 01 00 16 BA 01 33 35 01 87 ...0...!..39..
0120 80 00 40 00 50 01 20 08 00 C8 00 00 00 00 09 ...3..!.....
0130 00 08 00 80 10 04 01 38 53 45 54 08 64 01 00 DF .....8CET....
REG ADDR 0100 X _____ X causes control to return to EXbug

```

4

TEST THE SAMPLE PROGRAM

```

EXBUG 1.1 MAID _____ Call the MAID (Motorola Active Interface Debug)
*OR=00 54 _____ feature of EXbug
*IR _____ Change memory location A to contain 54
P-FASB X-0000 A-00 B-00 C-00 S-FF8A _____ Display Registers
P-FASB 100 _____ Set program counter to 0100
*#12= 000C _____ Convert 12 Decimal to hexadecimal
*CI= _____ Trace C instructions
P-0103 X-0000 A-00 B-00 C-00 S-0132 _____
P-0106 X-0138 A-00 B-00 C-00 S-0132 _____
P-0108 X-0138 A-00 B-03 C-00 S-0132 _____
P-010A X-0138 A-54 B-03 C-00 S-0132 _____
P-010C X-0138 A-54 B-03 C-04 S-0132 _____
P-0113 X-0138 A-54 B-03 C-04 S-0132 _____
P-0119 X-0138 A-54 B-03 C-04 S-0130 _____
P-011A X-0138 A-54 B-54 C-00 S-0130 _____
P-011D X-0138 A-D4 B-54 C-08 S-0130 _____
P-0116 X-0138 A-D4 B-54 C-08 S-0132 _____
P-0100 X-0138 A-D4 B-54 C-08 S-0132 _____
P-0103 X-0138 A-D4 B-54 C-00 S-0132 _____
*

```

Register heading:
P Program counter
X Index register
A Accumulator A
B Accumulator B
C Condition code register
S Stack pointer

When tracing, the contents of the MC6800 Microprocessor registers are printed after each instruction is executed

HELPFUL HINTS



The assembly directive "OPT" follows the short form as described in the M6800 Programming Manual. If the long form is used only the short form is syntactically checked; however, the long form is scanned until a space or comma is found. The OPT operands DB8 and DB10 are ignored.

When used with the Texas Instruments Silent 700 terminal equipped with dual cassettes, 1200 Baud, and Remote Device Control, the resident assembler has throughput 12 times faster than a teletypewriter.

The resident assembler is a two-pass assembler; however, the assembler provides options as to how these passes may be run.

1P — Normal pass one (clear symbol table and start pass one)

1S — Start pass one, do not clear symbol table before starting

2P — Normal pass two

2L — Start pass two, list only (allows use of TTY's without punch control)

2T — Start pass two, punch tape only (allows use of TTY's without punch control)

Pass two of the assembler may be used without pass one; but forward references will be undefined, and the user must patch these forward references before using the Object Tape generated. This is a useful feature if a TTY is used for assemblies and the program contains few forward references, i.e., a savings of one-half of assembly time.

RESIDENT EDITOR COMMAND SUMMARY

- A — Appends an input string from the reader device to the workspace. Ignores nulls and rubouts. Terminates on EOF character (1A Hex), workspace full, or after 50 lines.
- B — Positions the workspace character pointer to the beginning of the workspace buffer.
- Cstring1String2 — Searches for string1 in the workspace buffer, and, if found, replaces (Changes) string1 with string2. String2 need not be the same length as string1.
- nD — Deletes from 1 to 254 characters from the workspace buffer (n may be pos. or neg.). The characters are deleted from the present position of the workspace character pointer.
- E — Ends the edit operation by transferring the entire contents of the workspace buffer to the punch device, and by then copying that which is left in the reader device to the punch device until an EOF is encountered. Terminates by punching an EOF, blank trailer tape, and then re-starting the Editor.
- F — Outputs 6 inches of blank leader/trailer code to the punch device.
- Istring — Inserts string into the workspace buffer at the present position of the workspace character pointer. The contents of the workspace buffer are repositioned, if necessary, to accommodate the string.
- nK — Kills from 1 to 254 lines (up to and including CR) from the workspace buffer (n may be pos. or neg.). The lines are deleted from the present position of the workspace character pointer.
- nL — Position the workspace character pointer n lines from its present position in the workspace buffer (n may be pos. or neg.). If n equals 0, the workspace character pointer is positioned to the beginning of the line in which the workspace character pointer presently resides.
- nM — Position the workspace character pointer n characters from its present position in the workspace buffer (n may be pos. or neg.). If n equals 0, no repositioning occurs.
- nP — Punches from 1 to 254 lines from the beginning of the workspace buffer. Output is to the punch device. Lines output are deleted from the workspace.
- Sstring — Searches for string in the workspace buffer. Positions workspace character pointer after found string.

nT — Prints (Types) from 1 to 254 lines from the workspace buffer (n may be pos. or neg.). Printing begins from the present position of the workspace character pointer.

Z — Positions the workspace character pointer to the end of the contents of the workspace buffer.

EXORciser COMMAND SUMMARY

EXbug ROUTINES

- LOAD — Loader
- VERF — Verify
- PNCH — Punch
- PRNT — Print
- SRCH — Search
- MAID — Motorola active interface debug routine
- S10. — Set speed 10 cps
- S30. — Set speed 30 cps
- S120 — Set speed 120 cps

MAID COMMANDS

- n/ — Open byte addressed by n
- (LF) — Open next sequential location
- (CR) — Close open location
- (UA) — Open previous sequential location
- X — Return to EXbug scan loop
- n;V — Set breakpoint at location n
- ;U — Remove all breakpoints
- n;U — Remove breakpoint at location n
- n;W — Search for n bit pattern
- ;G — Execute target program from restart vector
- n;G — Execute target program from location n
- ;P — Continue executing from encountered breakpoint
- n;P — Continue executing until breakpoint found n times
- n;O — Calculate offset from current location to n
- N — Trace one instruction
- ;N — Trace one instruction
- n;N — Trace n instructions
- \$V — Display breakpoints
- \$M — Display search mask and limits
- \$R — Display/change target program registers
- \$T — Set trace mode and set trace to address
- \$S — Set stop on address compare
- ;T — Reset trace Mode
- ;S — Reset stop on address compare
- #n= — Convert decimal to hexadecimal
- #\$n= — Convert hexadecimal to decimal
- #@n= — Convert octal to hexadecimal

LANGUAGE OF THE M6800 MICROPROCESSOR



MICROPROCESSOR INSTRUCTION SET ALPHABETIC SEQUENCE

ABA	Add Accumulators
ADC	Add with Carry
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to Zero
BGE	Branch if Greater or Equal Zero
BGT	Branch if Greater than Zero
BHI	Branch if Higher
BIT	Bit Test
BLE	Branch if Less or Equal
BLS	Branch if Lower or Same
BLT	Branch if Less than Zero
BMI	Branch if Minus
BNE	Branch if Not Equal to Zero
BPL	Branch if Plus
BRA	Branch Always
BSR	Branch to Subroutine
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CBA	Compare Accumulators
CLC	Clear Carry
CLI	Clear Interrupt Mask
CLR	Clear
CLV	Clear Overflow
CMP	Compare
COM	Complement
CPX	Compare Index Register
DAA	Decimal Adjust
DEC	Decrement
DES	Decrement Stack Pointer
DEX	Decrement Index Register
EOR	Exclusive OR
INC	Increment
INS	Increment Stack Pointer
INX	Increment Index Register
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator
LDS	Load Stack Pointer
LDX	Load Index Register
LSR	Logical Shift Right
NEG	Negate
NOP	No Operation
ORA	Inclusive OR Accumulator
PSH	Push Data
PUL	Pull Data
ROL	Rotate Left
ROR	Rotate Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBA	Subtract Accumulators
SBC	Subtract with Carry
SEC	Set Carry
SEI	Set Interrupt Mask
SEV	Set Overflow
STA	Store Accumulator
STS	Store Stack Register
STX	Store Index Register
SUB	Subtract
SWI	Software Interrupt
TAB	Transfer Accumulators
TAP	Transfer Accumulators to Condition Code Reg.
TBA	Transfer Accumulators
TPA	Transfer Condition Code Reg. to Accumulator
TST	Test
TSX	Transfer Stack Pointer to Index Register
TXS	Transfer Index Register to Stack Pointer
WAI	Wait for Interrupt

INSTRUCTION ADDRESSING MODES AND ASSOCIATED EXECUTION TIMES

(in microseconds assuming a 1 MHz clock)

	(Dual Operand)							
	ACCX	Immediate	Direct	Extended	Indexed	Implied	Relative	
ABA	•	•	•	•	•	•	•	2
ADC x	•	•	2	3	4	5	•	•
ADD x	•	•	2	3	4	5	•	•
AND x	•	•	2	3	4	5	•	•
ASL	2	•	•	•	6	7	•	•
ASR	2	•	•	•	6	7	•	•
BCC	•	•	•	•	•	•	•	4
BCS	•	•	•	•	•	•	•	4
BEQ	•	•	•	•	•	•	•	4
BGE	•	•	•	•	•	•	•	4
BGT	•	•	•	•	•	•	•	4
BHI	•	•	•	•	•	•	•	4
BIT x	•	2	3	4	5	•	•	•
BLE	•	•	•	•	•	•	•	4
BLS	•	•	•	•	•	•	•	4
BLT	•	•	•	•	•	•	•	4
BMI	•	•	•	•	•	•	•	4
BNE	•	•	•	•	•	•	•	4
BPL	•	•	•	•	•	•	•	4
BRA	•	•	•	•	•	•	•	4
BSR	•	•	•	•	•	•	•	4
BVC	•	•	•	•	•	•	•	4
BVS	•	•	•	•	•	•	•	4
CBA	•	•	•	•	•	•	2	•
CLC	•	•	•	•	•	•	2	•
CLI	•	•	•	•	•	•	2	•
CLR	2	•	•	•	6	7	•	•
CLV	•	•	•	•	•	•	2	•
CMP x	•	•	2	3	4	5	•	•
COM	2	•	•	•	6	7	•	•
CPX	•	3	4	5	6	•	•	•
DAA	•	•	•	•	•	•	2	•
DEC	2	•	•	•	6	7	•	•
DES	•	•	•	•	•	•	•	4
DEX	•	•	•	•	•	•	•	4
EOR x	•	•	2	3	4	5	•	•
INC	2	•	•	•	6	7	•	•
INS	•	•	•	•	•	•	•	4
INX	•	•	•	•	•	•	•	4
JMP	•	•	•	•	3	4	•	•
JSR	•	•	•	•	9	8	•	•
LDA x	•	•	2	3	4	5	•	•
LDS	•	3	4	5	6	•	•	•
LDX	•	3	4	5	6	•	•	•
LSR	2	•	•	•	6	7	•	•
NEG	2	•	•	•	6	7	•	•
NOP	•	•	•	•	•	•	2	•
ORA x	•	•	2	3	4	5	•	•
PSH	4	•	•	•	•	•	•	•
PUL	4	•	•	•	•	•	•	•
ROL	2	•	•	•	6	7	•	•
ROR	2	•	•	•	6	7	•	•
RTI	•	•	•	•	•	•	10	•
RTS	•	•	•	•	•	•	5	•
SBA	•	•	•	•	•	•	•	2
SBC x	•	•	2	3	4	5	•	•
SEC	•	•	•	•	•	•	•	2
SEI	•	•	•	•	•	•	•	2
SEV	•	•	•	•	•	•	•	2
STA x	•	•	•	4	5	6	•	•
STS	•	•	•	5	6	7	•	•
STX	•	•	•	5	6	7	•	•
SUB x	•	•	2	3	4	5	•	•
SWI	•	•	•	•	•	•	12	•
TAB	•	•	•	•	•	•	•	2
TAP	•	•	•	•	•	•	•	2
TBA	•	•	•	•	•	•	•	2
TPA	•	•	•	•	•	•	•	2
TST	2	•	•	•	6	7	•	•
TSX	•	•	•	•	•	•	•	4
TXS	•	•	•	•	•	•	•	4
WAI	•	•	•	•	•	•	9	•

LIST OF ASSEMBLER DIRECTIVES

END	End of Program
EQU	Equate Symbol
FCB	Form Constant Byte
FCC	Form Constant Characters
FDB	Form Double Constant Byte
MON	Return to Console
NAM	Name
OPT	Option
ORG	Origin
PAGE	Top of Form
RMB	Reserve Memory Byte
SPC	Space Lines

ACCX (accumulator only) Addressing

In accumulator only addressing, either accumulator A or accumulator B is specified. These are one-byte instructions.

Immediate Addressing

In immediate addressing, the operand is contained in the second byte of the instruction. No further addressing of memory is required. The MPU addresses this location when it fetches the immediate instruction for execution. These are two/three-byte instructions.

Direct Addressing

In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes in the machine; i.e., locations zero through 255. That part of the memory should be used for temporary data storage and intermediate results. In most configurations, it should be a random access memory. These are two-byte instructions.

Extended Addressing

In extended addressing, the value contained in the second byte of the instruction is used as the higher eight-bits of the address of the operand. The third byte of the instruction is used as the lower eight-bits of the address of the operand. This gives one a 16-bit address for the operand. This is an absolute address in memory. These are three-byte instructions.

Indexed Addressing

In indexed addressing, the value contained in the second byte of the instruction is added to the index register's lower eight-bits in the MPU. The carry is then added to the higher order eight-bits of the index register. This result is then used to address memory. The modified address is held in a temporary address register so there is no change to the index register. These are two-byte instructions.

Implied Addressing

In the implied addressing mode the instruction gives the address (i.e., stack pointer, index register, etc.). These are one-byte instructions.

Relative Addressing

In relative addressing, the value contained in the second byte of the instruction is added to the program counters lowest eight-bits plus two. The carry or borrow is then added to the high eight-bits. This allows the user to address data within a range of -126 to +129 bytes of the present instruction. These are two-byte instructions.



MOTOROLA
Semiconductor Products Inc.

8005 EAST MCDOWELL ROAD, PHOENIX, ARIZONA 85008

Instruction Set

EXECUTABLE INSTRUCTIONS – ALPHABETIC LIST

ABA	ADD ACCUMULATORS	INS	INCREMENT STACK POINTER
ADC	ADD WITH CARRY	INX	INCREMENT INDEX REGISTER
ADD	ADD		
AND	LOGICAL AND	JMP	JUMP
ASL	ARITHMETIC SHIFT LEFT	JSR	JUMP TO SUBROUTINE
ASR	ARITHMETIC SHIFT RIGHT		
		LDA	LOAD ACCUMULATOR
BCC	BRANCH IF CARRY CLEAR	LDS	LOAD STACK POINTER
BCS	BRANCH IS CARRY SET	LDX	LOAD INDEX REGISTER
BEQ	BRANCH IF EQUAL TO ZERO	LSR	LOGICAL SHIFT RIGHT
BGE	BRANCH IF GREATER OR EQUAL TO ZERO		
BGT	BRANCH IF GREATER THAN ZERO	NEG	NEGATE
BHI	BRANCH IF HIGHER	NOP	NO OPERATION
BIT	BIT TEST		
BLE	BRANCH IF LESS OR EQUAL	ORA	INCLUSIVE OR ACCUMULATOR
BLS	BRANCH IF LOWER OR SAME		
BLT	BRANCH IF LESS THAN ZERO	PSH	PUSH DATA
BMI	BRANCH IF MINUS	PUL	PULL DATA
BNE	BRANCH IF NOT EQUAL TO ZERO		
BPL	BRANCH IF PLUS	ROL	ROTATE LEFT
BRA	BRANCH ALWAYS	ROR	ROTATE RIGHT
BSR	BRANCH TO SUBROUTINE	RTI	RETURN FROM INTERRUPT
BVC	BRANCH IF OVERFLOW CLEAR	RTS	RETURN FROM SUBROUTINE
BVS	BRANCH IF OVERFLOW SET		
		SBA	SUBTRACT ACCUMULATORS
CBA	COMPARE ACCUMULATORS	SBC	SUBTRACT WITH CARRY
CLC	CLEAR CARRY	SEC	SET CARRY
CLI	CLEAR INTERRUPT MASK	SEI	SET INTERRUPT MASK
CLR	CLEAR	SEV	SET OVERFLOW
CLV	CLEAR OVERFLOW	STA	STORE ACCUMULATOR
CMP	COMPARE	STS	STORE STACK REGISTER
COM	COMPLEMENT	STX	STORE INDEX REGISTER
CPX	COMPARE INDEX REGISTER	SUB	SUBTRACT
		SWI	SOFTWARE INTERRUPT
DAA	DECIMAL ADJUST		
DEC	DECREMENT	TAB	TRANSFER ACCUMULATORS
DES	DECREMENT STACK POINTER	TAP	TRANSFER ACCUMULATORS TO CONDITION CODE REG
DEX	DECREMENT INDEX REGISTER	TBA	TRANSFER ACCUMULATORS
		TPA	TRANSFER CONDITION CODE REG TO ACCUMULATOR
EOR	EXCLUSIVE OR	TST	TEST
		TSX	TRANSFER STACK POINTER TO INDEX REGISTER
INC	INCREMENT	TXS	TRANSFER INDEX REGISTER TO STACK POINTER
		WAI	WAIT FOR INTERRUPT

TABLE 3 - ACCUMULATOR AND MEMORY INSTRUCTIONS

ACCUMULATOR AND MEMORY OPERATIONS		MEMEMONIC		ADDRESSING MODES												BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	COND. CODE REG.						
				IMMED			DIRECT			INDEX			EXTND				INHER			5	4	3	2
OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	H	I	N	Z	V	C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3					A + M → A	†	†	†	†	†
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3					B + M → B	†	†	†	†	†
Add Acmltrs	ABA													1B	2	1		A + B → A	†	†	†	†	†
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3					A + M + C → A	†	†	†	†	†
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3					B + M + C → B	†	†	†	†	†
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3					A · M → A	•	•	•	•	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3					B · M → B	•	•	•	•	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3					A · M	•	•	•	•	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3					B · M	•	•	•	•	•
Clear	CLR																	00 → M	•	•	•	•	•
	CLRA													4F	2	1		00 → A	•	•	•	•	•
	CLRB													5F	2	1		00 → B	•	•	•	•	•
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3					A - M	•	•	•	•	•
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3					B - M	•	•	•	•	•
Compare Acmltrs	CBA													11	2	1		A - B	•	•	•	•	•
Complement, 1's	COM							63	7	2	73	6	3					M → M	•	•	•	•	•
	COMA													43	2	1		A → A	•	•	•	•	•
	COMB													53	2	1		B → B	•	•	•	•	•
Complement, 2's (Negate)	NEG							60	7	2	70	6	3					00 - M → M	•	•	•	•	•
	NEGA													40	2	1		00 - A → A	•	•	•	•	•
	NEGB													50	2	1		00 - B → B	•	•	•	•	•
Decimal Adjust, A	DAA													19	2	1		Converts Binary Add. of BCD Characters into BCD Format	•	•	•	•	•
Decrement	DEC							6A	7	2	7A	6	3					M - 1 → M	•	•	•	•	•
	DECA													4A	2	1		A - 1 → A	•	•	•	•	•
	DECB													5A	2	1		B - 1 → B	•	•	•	•	•
Exclusive OR	EDRA	88	2	2	98	3	2	A8	5	2	B8	4	3					A ⊕ M → A	•	•	•	•	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3					B ⊕ M → B	•	•	•	•	•
Increment	INC							6C	7	2	7C	6	3					M + 1 → M	•	•	•	•	•
	INCA													4C	2	1		A + 1 → A	•	•	•	•	•
	INCB													5C	2	1		B + 1 → B	•	•	•	•	•
Load Acmltr	LDA	86	2	2	96	3	2	A6	5	2	B6	4	3					M → A	•	•	•	•	•
	LDB	C6	2	2	D6	3	2	E6	5	2	F6	4	3					M → B	•	•	•	•	•
Or, Inclusive	ORA	8A	2	2	9A	3	2	AA	5	2	BA	4	3					A ∨ M → A	•	•	•	•	•
	ORB	CA	2	2	DA	3	2	EA	5	2	FA	4	3					B ∨ M → B	•	•	•	•	•
Push Data	PSHA													36	4	1		A → M _{SP} , SP - 1 → SP	•	•	•	•	•
	PSHB													37	4	1		B → M _{SP} , SP - 1 → SP	•	•	•	•	•
Pull Data	PULA													32	4	1		SP + 1 → SP, M _{SP} → A	•	•	•	•	•
	PULB													33	4	1		SP + 1 → SP, M _{SP} → B	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3					M	•	•	•	•	•
	ROLA													49	2	1		A	•	•	•	•	•
	ROLB													59	2	1		B	•	•	•	•	•
Rotate Right	ROR							66	7	2	76	6	3					M	•	•	•	•	•
	RORA													46	2	1		A	•	•	•	•	•
	RORB													56	2	1		B	•	•	•	•	•
Shift Left, Arithmetic	ASL							68	7	2	78	6	3					M	•	•	•	•	•
	ASLA													48	2	1		A	•	•	•	•	•
	ASLB													58	2	1		B	•	•	•	•	•
Shift Right, Arithmetic	ASR							67	7	2	77	6	3					M	•	•	•	•	•
	ASRA													47	2	1		A	•	•	•	•	•
	ASRB													57	2	1		B	•	•	•	•	•
Shift Right, Logic	LSR							64	7	2	74	6	3					M	•	•	•	•	•
	LSRA													44	2	1		A	•	•	•	•	•
	LSRB													54	2	1		B	•	•	•	•	•
Store Acmltr.	STAA							97	4	2	A7	6	2	B7	5	3		A → M	•	•	•	•	•
	STAB							D7	4	2	E7	6	2	F7	5	3		B → M	•	•	•	•	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3					A - M → A	•	•	•	•	•
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3					B - M → B	•	•	•	•	•
Subtract Acmltrs.	SBA													10	2	1		A - B → A	•	•	•	•	•
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3					A - M - C → A	•	•	•	•	•
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3					B - M - C → B	•	•	•	•	•
Transfer Acmltrs	TAB													16	2	1		A → B	•	•	•	•	•
	TBA													17	2	1		B → A	•	•	•	•	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3					M - 00	•	•	•	•	•
	TSTA													4D	2	1		A - 00	•	•	•	•	•
	TSTB													5D	2	1		B - 00	•	•	•	•	•

LEGEND:

- OP Operation Code (Hexadecimal);
- ~ Number of MPU Cycles;
- ≠ Number of Program Bytes;
- + Arithmetic Plus;
- Arithmetic Minus;
- Boolean AND;
- M_{SP} Contents of memory location pointed to be Stack Pointer;
- † Boolean Inclusive OR;
- ⊕ Boolean Exclusive OR;
- M̄ Complement of M;
- Transfer Into;
- 0 Bit = Zero;
- 00 Byte = Zero;
- H Half-carry from bit 3;
- I Interrupt mask;
- N Negative (sign bit);
- Z Zero (byte)
- V Overflow, 2's complement
- C Carry from bit 7
- R Reset Always
- S Set Always
- † Test and set if true, cleared otherwise
- Not Affected
- CCR Condition Code Register
- LS Least Significant
- MS Most Significant

TABLE 4 - INDEX REGISTER AND STACK MANIPULATION INSTRUCTIONS

INDEX REGISTER AND STACK		IMMED		DIRECT		INDEX		EXTND			INHER			BOOLEAN/ARITHMETIC OPERATION	5 4 3 2 1 0					
POINTER OPERATIONS	MNEMONIC	OP	#	OP	#	OP	#	OP	#	OP	#	OP	#		H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3							
Decrement Index Reg	DEX											09	4	1						
Decrement Stack Ptr	DES											34	4	1						
Increment Index Reg	INX											08	4	1						
Increment Stack Ptr	INS											31	4	1						
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3							
Load Stack Ptr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3							
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3							
Store Stack Ptr	STS				9F	5	2	AF	7	2	BF	6	3							
Index Reg → Stack Ptr	TXS											35	4	1						
Stack Ptr → Index Reg	TSX											30	4	1						

TABLE 5 - JUMP AND BRANCH INSTRUCTIONS

JUMP AND BRANCH OPERATIONS		RELATIVE		INDEX		EXTND			INHER			BRANCH TEST	5 4 3 2 1 0							
MNEMONIC	OP	#	OP	#	OP	#	OP	#	OP	#	H		I	N	Z	V	C			
Branch Always	BRA	20	4	2																
Branch If Carry Clear	BCC	24	4	2																
Branch If Carry Set	BCS	25	4	2																
Branch If = Zero	BEQ	27	4	2																
Branch If ≥ Zero	BGE	2C	4	2																
Branch If > Zero	BGT	2E	4	2																
Branch If Higher	BHI	22	4	2																
Branch If ≤ Zero	BLE	2F	4	2																
Branch If Lower Or Same	BLS	23	4	2																
Branch If < Zero	BLT	2D	4	2																
Branch If Minus	BMI	28	4	2																
Branch If Not Equal Zero	BNE	26	4	2																
Branch If Overflow Clear	BVC	28	4	2																
Branch If Overflow Set	BVS	29	4	2																
Branch If Plus	BPL	2A	4	2																
Branch To Subroutine	BSR	8D	8	2																
Jump	JMP				6E	4	2	7E	3	3										
Jump To Subroutine	JSR				AD	8	2	BD	9	3										
No Operation	NOP										01	2	1							
Return From Interrupt	RTI										38	10	1							
Return From Subroutine	RTS										39	5	1							
Software Interrupt	SWI										3F	12	1							
Wait for Interrupt	WAI										3E	9	1							

TABLE 6 - CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

CONDITIONS CODE REGISTER		INHER			BOOLEAN OPERATION	5 4 3 2 1 0					
OPERATIONS	MNEMONIC	OP	#	OP		H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	•
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Accmtr A → CCR	TAP	06	2	1	A → CCR	•	•	•	•	•	•
CCR → Accmtr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES:

- (Bit set if test is true and cleared otherwise)
- ① (Bit V) Test: Result = 10000000?
- ② (Bit C) Test: Result ≠ 00000000?
- ③ (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
- ④ (Bit V) Test: Operand = 10000000 prior to execution?
- ⑤ (Bit V) Test: Operand = 01111111 prior to execution?
- ⑥ (Bit V) Test: Set equal to result of N ⊕ C after shift has occurred.
- ⑦ (Bit N) Test: Sign bit of most significant (MS) byte of result = 1?
- ⑧ (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
- ⑨ (Bit N) Test: Result less than zero? (Bit 15 = 1)
- ⑩ (All) Load Condition Code Register from Stack. (See Special Operations)
- ⑪ (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
- ⑫ (All) Set according to the contents of Accumulator A.

M6800 PROGRAM

72 INSTRUCTIONS 6 ADDRESSING MODES

TR1074

DATA HANDLING INSTRUCTIONS (Data Movement)

FUNCTION	MNEMONIC	OPERATION
LOAD ACMLTR	LDAA	M → A
	LDAB	M → B
PUSH DATA	PSHA	A → M _{SP} , SP - 1 → SP
	PSHB	B → M _{SP} , SP - 1 → SP
PULL DATA	PULA	SP + 1 → SP, M _{SP} → A
	PULB	SP + 1 → SP, M _{SP} → B
STORE ACMLTR	STAA	A → M
	STAB	B → M
TRANSFER ACMLTRS	TAB	A → B
	TBA	B → A

TR1075

DATA HANDLING INSTRUCTIONS (ALTER DATA)

FUNCTION	MNEMONIC	OPERATION
CLEAR	CLR	$00 \rightarrow M$
	CLRA	$00 \rightarrow A$
	CLRB	$00 \rightarrow B$
DECREMENT	DEC	$M - 1 \rightarrow M$
	DECA	$A - 1 \rightarrow A$
	DECB	$B - 1 \rightarrow B$
INCREMENT	INC	$M + 1 \rightarrow M$
	INCA	$A + 1 \rightarrow A$
	INCB	$B + 1 \rightarrow B$
COMPLEMENT, 2'S (NEGATE)	NEG	$00 - M \rightarrow M$
	NEGA	$00 - A \rightarrow A$
	NEGB	$00 - B \rightarrow B$
COMPLEMENT, 1'S	COM	$\overline{M} \rightarrow M$
	COMA	$\overline{A} \rightarrow A$
	COMB	$\overline{B} \rightarrow B$

TR1076

DATA HANDLING INSTRUCTIONS (SHIFT AND ROTATE)

FUNCTION	MNEMONIC	OPERATION	
ROTATE LEFT	ROL	M	
	ROLA	A	
	ROLB	B	
ROTATE RIGHT	ROR	M	
	RORA	A	
	RORB	B	
SHIFT LEFT, ARITHMETIC	ASL	M	
	ASLA	A	
	ASLB	B	
SHIFT RIGHT, ARITHMETIC	ASR	M	
	ASRA	A	
	ASRB	B	
SHIFT RIGHT, LOGIC	LSR	M	
	LSRA	A	
	LSRB	B	

TR1077

ARITHMETIC INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
ADD	ADDA	$A + M \rightarrow A$
	ADDB	$B + M \rightarrow B$
ADD ACCUMULATORS	ABA	$A + B \rightarrow A$
ADD WITH CARRY	ADCA	$A + M + C \rightarrow A$
	ADCB	$B + M + C \rightarrow B$
COMPLEMENT, 2'S (NEGATE)	NEG	$00 - M \rightarrow M$
	NEGA	$00 - A \rightarrow A$
	NEGB	$00 - B \rightarrow B$
DECIMAL ADJUST, A	DAA	CONVERTS BINARY ADD. OF BCD CHARACTERS INTO BCD FORMAT
SUBTRACT	SUBA	$A - M \rightarrow A$
	SUBB	$B - M \rightarrow B$
SUBTRACT ACCUMULATORS	SBA	$A - B \rightarrow A$
SUBTRACT WITH CARRY	SBCA	$A - M - C \rightarrow A$
	SBCB	$B - M - C \rightarrow B$

TR1078

LOGIC INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
AND	ANDA	$A \bullet M \rightarrow A$
	ANDB	$B \bullet M \rightarrow B$
COMPLEMENT, 1'S	COM	$\overline{M} \rightarrow M$
	COMA	$\overline{A} \rightarrow A$
	COMB	$\overline{B} \rightarrow B$
EXCLUSIVE OR	EORA	$A \oplus M \rightarrow A$
	EORB	$B \oplus M \rightarrow B$
OR, INCLUSIVE	ORA A	$A + M \rightarrow A$
	ORA B	$B + M \rightarrow B$

TR1079

JUMP AND BRANCH INSTRUCTIONS

FUNCTION	MNEMONIC	BRANCH TEST
BRANCH ALWAYS	BRA	NONE
BRANCH IF CARRY CLEAR	BCC	$C = 0$
BRANCH IF CARRY SET	BCS	$C = 1$
BRANCH IF = ZERO	BEQ	$Z = 1$
BRANCH IF \geq ZERO	BGE	$N \oplus V = 0$
BRANCH IF $>$ ZERO	BGT	$Z + (N \oplus V) = 0$
BRANCH IF HIGHER	BHI	$C + Z = 0$
BRANCH IF \leq ZERO	BLE	$Z + (N \oplus V) = 1$
BRANCH IF LOWER OR SAME	BLS	$C + Z = 1$
BRANCH IF $<$ ZERO	BLT	$N \oplus V = 1$
BRANCH IF MINUS	BMI	$N = 1$
BRANCH IF NOT EQUAL ZERO	BNE	$Z = 0$
BRANCH IF PLUS	BPL	$N = 0$

TR1080

JUMP AND BRANCH INSTRUCTIONS

FUNCTION	MNEMONIC	BRANCH TEST
BRANCH IF OVERFLOW CLEAR	BVC	$V = 0$
BRANCH IF OVERFLOW SET	BVS	$V = 1$
BRANCH TO SUBROUTINE	BSR	
JUMP	JMP	
JUMP TO SUBROUTINE	JSR	
NO OPERATION	NOP	ADVANCES PROG. CNTR. ONLY
RETURN FROM SUBROUTINE	RTS	

TR1081

DATA TEST INSTRUCTIONS

FUNCTION	MNEMONIC	TEST
BIT TEST	BITA	A • M
	BIT B	B • M
COMPARE	CMPA	A - M
	CMPB	B - M
	CBA	A - B
TEST, ZERO OR MINUS	TST	M - 00
	TSTA	A - 00
	TSTB	B - 00

TR1082

CONDITION CODE REGISTER INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
CLEAR CARRY	CLC	0 → C
CLEAR INTERRUPT MASK	CLI	0 → I
CLEAR OVERFLOW	CLV	0 → V
SET CARRY	SEC	1 → C
SET INTERRUPT MASK	SEI	1 → I
SET OVERFLOW	SEV	1 → V
ACMLTR A → CCR	TAP	A → CCR
CCR → ACMLTR A	TPA	CCR → A

TR1083

INDEX REGISTER AND STACK POINTER INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
COMPARE INDEX REG	CPX	$X_H - M, X_L - (M + 1)$
DECREMENT INDEX REG	DEX	$X - 1 \rightarrow X$
DECREMENT STACK PNTR	DES	$SP - 1 \rightarrow SP$
INCREMENT INDEX REG	INX	$X + 1 \rightarrow X$
INCREMENT STACK PNTR	INS	$SP + 1 \rightarrow SP$
LOAD INDEX REG	LDX	$M \rightarrow X_H, (M + 1) \rightarrow X_L$
LOAD STACK PNTR	LDS	$M \rightarrow SP_H, (M + 1) \rightarrow SP_L$
STORE INDEX REG	STX	$X_H \rightarrow M, X_L \rightarrow (M + 1)$
STORE STACK PNTR	STS	$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$
INDX REG \rightarrow STACK PNTR	TXS	$X - 1 \rightarrow SP$
STACK PNTR \rightarrow INDX REG	TSX	$SP + 1 \rightarrow X$

TR1084

INTERRUPT HANDLING INSTRUCTIONS

FUNCTION	MNEMONIC	OPERATION
SOFTWARE INTERRUPT	SWI	$REGS \leftarrow M_{SP}$ $SP - 7 \leftarrow SP$ $M_{FFFA} \leftarrow PCH$ $M_{FFFB} \leftarrow PCL$ $1 \leftarrow I$
RETURN FROM INTERRUPT	RTI	$M_{SP} \leftarrow REGS$ $SP + 7 \leftarrow SP$
WAIT FOR INTERRUPT	WAI	$REGS \leftarrow M_{SP}$ $SP - 7 \leftarrow SP$

TH1087

INPUT/OUTPUT INSTRUCTIONS

NONE!



TR1085

Program Problems

SAMPLE PROGRAM

PROBLEM: WRITE A PROGRAM, IN MACHINE LANGUAGE AND IN M6800 SOURCE LANGUAGE, TO ADD THE DECIMAL NUMBERS 25, 35, 50, AND 17. STORE THE ANSWER AT RAM LOCATION 0A. ASSEMBLE THE SOURCE PROGRAM AND COMPARE THE ASSEMBLED PROGRAM WITH THE MACHINE LANGUAGE PROGRAM.

SOLUTION: $35_{10} = 100011_2 = 23_{16}$
 $50_{10} = 110010_2 = 32_{16}$
 $17_{10} = 010001_2 = 11_{16}$
 $25_{10} = 011001_2 = 19_{16}$

<u>MEMORY LOCATION</u>	<u>MACHINE LANGUAGE</u>		<u>COMMENT</u>
	(HEX)	(BINARY) (HEX)	
000B	10000110	(86)	LDA A IMM
000C	00011001	(19)	DATA TO BE PUT IN A
000D	10001011	(88)	ADD A IMM
000E	00100011	(23)	DATA TO BE ADDED TO A
000F	10001011	(88)	ADD A IMM
0010	00110010	(32)	DATA TO BE ADDED TO A
0011	10001011	(88)	ADD A IMM
0012	00010001	(11)	DATA TO BE ADDED TO A
0013	10010111	(97)	STORES A IN LOCATION
0014	00001010	(0A)	0A

TR1189

PROG-2 Program Problems

READY
LIST

ADD4NR 20:38EST 11/11/75

100 NAM ADD4NR ADD 4 NUMPERS PROGRAM
110 ORG \$A
120 TEMP RMP 1
130 LDA A #25
140 ADD A #35
150 ADD A #32
160 AED A #10001
170 STA A TEMP
180 MON

} SAME PROGRAM
WRITTEN IN
MNEMONIC CODING
INDICATES IMMEDIATE
\$ INDICATES HEX NUMBER
% INDICATES BINARY NUMBER

TR1073

READY
FUN MPCASM

MPCASM 20:39EST 11/11/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIFLE FOR MPCASM
COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU CROSS ASSEMBLER, RELEASE 1.4

ENTER SI FILENAME
?ADD4NR

PAGE 1 ADD4NR 11/11/75 20:39:00

00100 NAM ADD4NR ADD 4 NUMPERS PROGEAM
00110 000A ORG \$A
00120 000A 0001 TEMP RMP 1
00130 000F 86 19 LDA A #25
00140 000D 8E 23 ADD A #35
00150 000F 8E 32 ADD A #32
00160 0011 8E 11 ADD A #10001
00170 0013 97 0A STA A TEMP
00180 MON

} SAME PROGRAM
ASSEMBLED BY
TIME-SHARING
CROSS-ASSEMBLER

TR1190

READY
LIST

ADD4NR 20:38EST 11/11/75

100 NAM ADD4NR ADD 4 NUMBERS PROGRAM
110 ORG \$A
120 TEMP RMB 1
130 LDA A #25
140 ADD A #35
150 ADD A #32
160 ADD A #10001
170 STA A TEMP
180 MON

TR1073

READY
RUN MPCASM

MPCASM 20:39EST 11/11/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPCASM
COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU CROSS ASSEMBLER, RELEASE 1.4

ENTER SI FILENAME
?ADD4NR

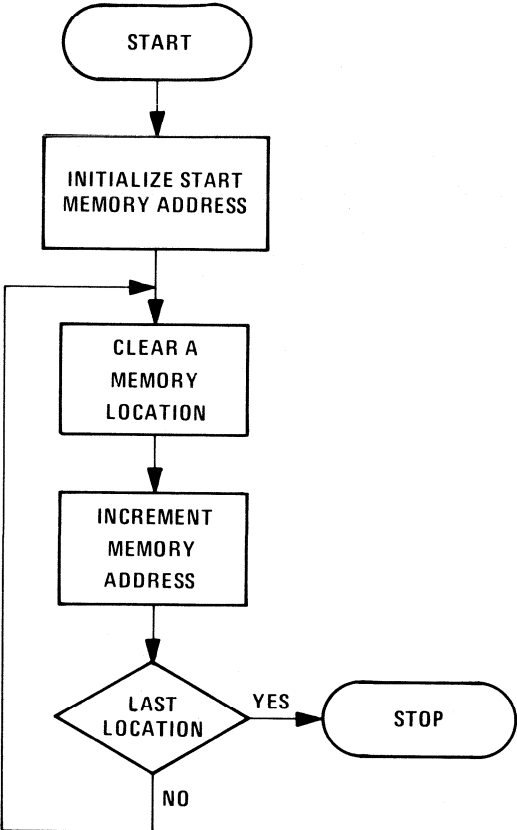
PAGE 1 ADD4NR 11/11/75 20:39.00

00100			NAM	ADD4NR	ADD 4 NUMBERS PROGRAM
00110	000A		ORG	\$A	
00120	000A	0001	TEMP	RMB	1
00130	000F	86 19	LDA	A	#25
00140	000D	8E 23	ADD	A	#35
00150	000F	8E 32	ADD	A	#32
00160	0011	8F 11	ADD	A	#10001
00170	0013	97 0A	STA	A	TEMP
00180			MON		

TR1190

PROG-4 Program Problems

CLEAR MEMORY LOCATIONS \$70 thru \$78



TR1211-1

PROG-6 Program Problems

A SOLUTION

```
LDX #$70
L1 CLR 0,X
   INX
   CPX #$79
   BNE L1
   BRA *
```

OTHER SOLUTIONS

```
(B) LDX #$9
    L1 CLR $6F,X
       DEX
       BNE L1
       BRA *
```

```
(C) LDS #$78
    CLR A
MORE PSH A
    TSX
    CPX #$70
    BNE MORE
    BRA *
```

PROBLEM

Clear memory in locations 0000 through \$00FF.

PROG- 8 Program Problems

CLEAR MEMORY
(0 → FF)

SOME SOLUTIONS

- | | | |
|-----|-----------------|-------------|
| (1) | LDX #\$0 | 13 BYTES |
| | CLR A | 4360 CYCLES |
| | AGAIN STA A 0,X | |
| | INX | |
| | CPX #\$100 | |
| | BNE AGAIN | |
| | BRA * | |
| (2) | LDX #\$0 | 11 BYTES |
| | AGAIN CLR 0,X | 4611 CYCLES |
| | INX | |
| | CPX #\$100 | |
| | BNE AGAIN | |
| | BRA * | |
| (3) | LDX #\$FF | 10 BYTES |
| | AGAIN CLR 0,X | 3850 CYCLES |
| | DEX | |
| | BNE AGAIN | |
| | CLR 0,X | |
| | BRA * | |

PROBLEM

Load memory with a data table:

ADDR	DATA
0000	00
0001	01
0002	02
0003	03
.	.
.	.
.	.
00FD	FD
00FE	FE
00FF	FF

PROG-10 Program Problems

A SOLUTION

```
LDX #$0  
CLR A  
NEXT STA A $0,X  
INC A  
INX  
CPX #$100  
BNE NEXT  
BRA *
```

PROBLEM

Write a program to build a table from 0-FF MEM location. The data in this table is to be FF → 0.

ADDR	DATA
0000	FF
0001	FE
0002	FD
.	.
.	.
.	.
00FD	02
00FE	01
00FF	00

PROG-12 Program Problems

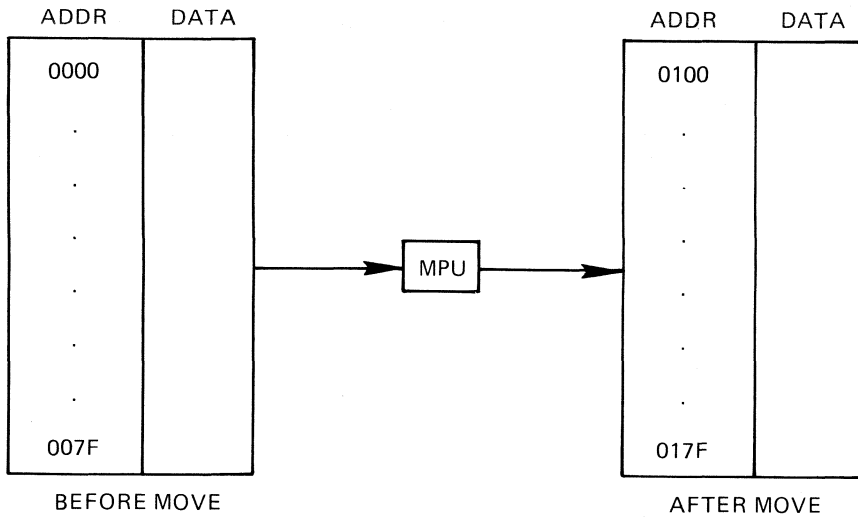
SOME SOLUTIONS

```
(A)          LDA A #$FF
              LDX #$00
AGAIN        STA A $0,X
              INX
              DEC A
              BNE AGAIN
              STA A $FF
              BRA *
```

```
(B)          LDS #$FF
              CLR A
AG           PSH A
              INC A
              BNE AG
              BRA *
```

PROBLEM – Move or Transfer \$80 Bytes of Data

The first byte is located at memory location 0000, and is to be transferred to memory location \$100. Start your program at \$500. This problem can be solved by using only the index register and the A accumulator.



PROG-14 Program Problems

PROBLEM

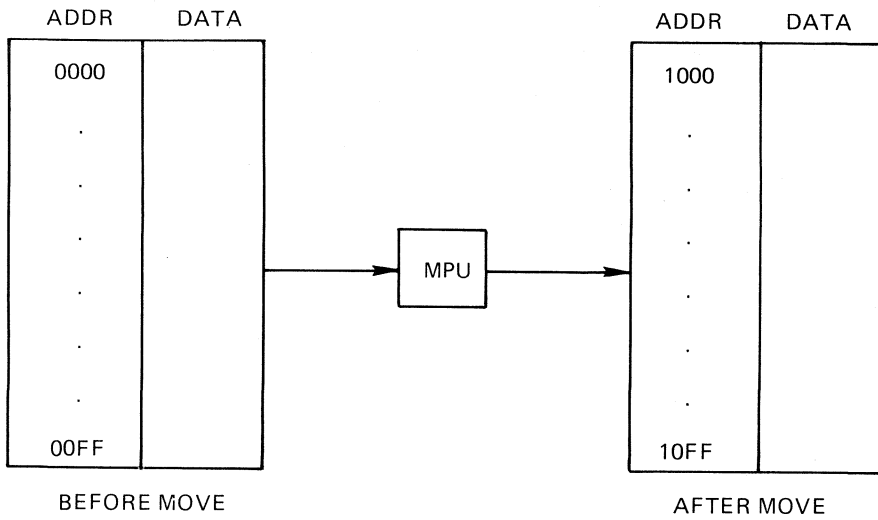
Move \$80 bytes of data from MEM LOC's 0 → \$7F to MEM LOC's \$100 → \$17F.

A SOLUTION

```
LDX #$0  
MORE LDA A,$0,X  
INX  
STA A,$FF,X  
CPX #$80  
BNE MORE  
BRA *
```

PROBLEM – Move or Transfer \$100 Bytes of Data.

The first byte to be moved is located at memory location 0000 and this byte is to be moved to memory location \$1000.



PROG-16 Program Problems

PROBLEM

Move \$100 bytes of data from \$0 to \$1000.

A SOLUTION

```
LDX #$1000
LDS #$FFFF
MORE PUL A
STA A $0,X
INX
CPX #$1100
BNE MORE
BRA*
```


PROGRAMMING PROBLEM FOR THE THIRD DAY

To gain some "hands-on" experience in both software and hardware, we ask that you attempt this homework problem before the beginning of the third-day session. You will have the opportunity to assemble and run your program solution on the EXORciser. You will receive personal help and coaching in areas where you require them.

Assume, as part of your system, you have two MCM6810 RAMs which start at the very bottom of memory. Your problem is to check the first (lowest) five bytes of the *second* RAM. If the *contents* of the RAM location is an odd number, invert each bit and store this result back in that RAM address. If the *contents* of that RAM location is an even number, clear that RAM location. Write the source program to accomplish the above problem. Start your program at location \$2000. Include a flow-chart of your solution.

Example Programs

EXAMPLE PROGRAMS AND SYSTEMS

Table of Contents

Loading and Storing Data	EX-2
Subtracting Absolute Value	EX-3
PIA Polling Routine	EX-5
Event Counter Priority Service Routine	EX-9
Multiply Subroutine	EX-11
System – BCD to LED Display	EX-19
System – Machine Control	EX-25
ACIA Memory Load/Dump Program	EX-33

All sample programs are for illustration only. It may not be the most efficient solution and is shown only as an example of programming techniques.

EX-2 Example Programs and Systems

SAMPLE PROGRAM — Loading and Storing Data

Write a program for the following sequence.

1. Begin with data 7F and load it into the A accumulator, then store the data in memory location 50.
2. From location 50, load the data into the B accumulator, then store it extended in memory location 0113.
3. Reload data into the A accumulator from the extended memory location and store the data in location 6A, then Jump back to the beginning.

Assume this program will be used in a microcomputer system with Hex RAM addresses 000 through 200 (512 bytes) and ROM addresses 800 through FFF (2048 bytes). All numbers are in Hex relation.

Source Program

```
EDU1          12:09EST    02/06/75

100  NAM LTR1
101  OPT MEM
102  ORG $6A
103  TEMP RMB 1
105  ORG $0800
110  START LDA A #$7F      START OF PROGRAM
120  STA A $50
130  LDA B $50            ADDRESS OF DATA
140  STA B $0113
150  LDA A $0113
180  STA A TEMP
190  JMP START
200  MON
```

Assembled Program

```
00100          NAM      LTR1
00101          OPT      MEM
00102 006A     ORG      $6A
00103 006A 0001  TEMP    RMB      1
00105 0800     ORG      $0800
00110 0800 86 7F  START  LDA A   #$7F      START OF PROGRAM
00120 0802 97 50          STA A   $50
00130 0804 D6 50          LDA B   $50      ADDRESS OF DATA
00140 0806 F7 0113        STA B   $0113
00150 0809 B6 0113        LDA A   $0113
00180 080C 97 6A          STA A   TEMP
00190 080E 7E 0800        JMP     START
00200          MON
```

SAMPLE PROGRAM – Subtracting Absolute Value of Two Numbers

Problem: Calculate a quantity Z which will be absolute value of Y subtracted from the absolute value of W. If the result is less than or equal to zero, set Z equal to zero.

$$Z = |W| - |Y| \quad \text{if } |W| > |Y|$$

$$Z = 0 \quad \text{if } |W| \leq |Y|$$

Source Program for Absolute Value Problem

```

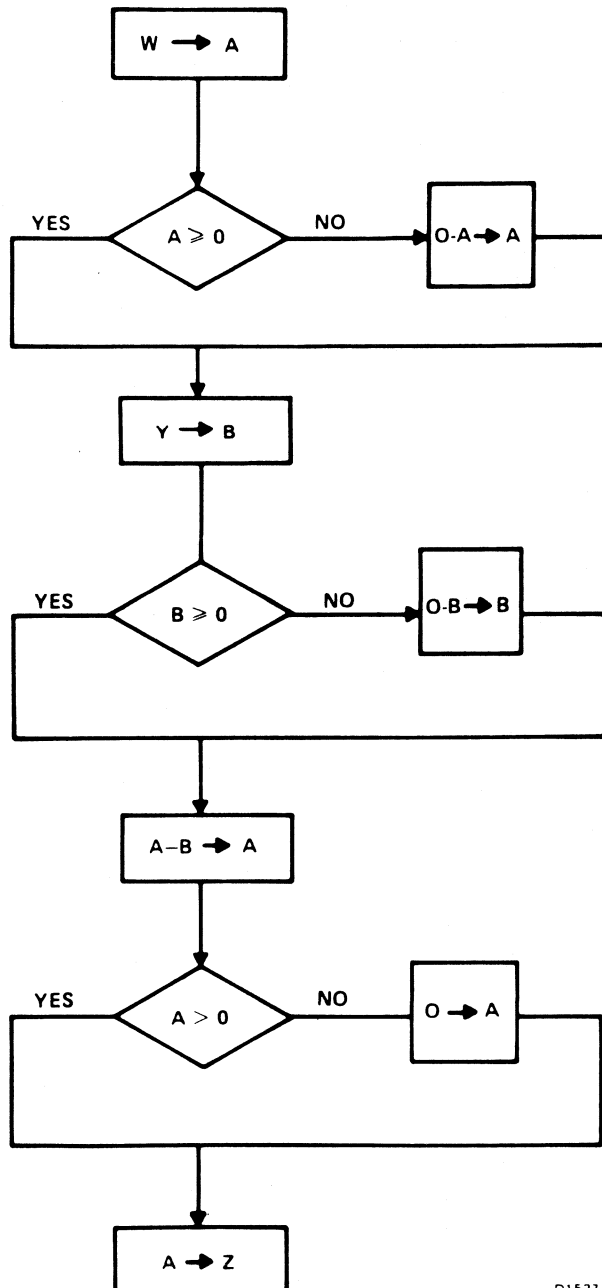
100  NAM ABS
110  OPT M
120  ORG 0
130  W RMB 1
140  Y RMB 1
150  Z RMB 1
160  ORG $0500
170  LDA A W
180  BPL Z1 IS W POSITIVE?
190  NEG A W WAS NEG, MAKE POS.
200  Z1 LDA B Y
210  BPL Z2 IS Y POSITIVE?
220  NEG B Y WAS NEG, MAKE POS.
230  Z2 SBA SUBTRACT Y FROM W
240  BGT Z3 IS Z POSITIVE?
250  CLR A RESULT WAS ZERO OR NEG.
260  Z3 STA A Z STORE ANSWER IN Z.
270  PRA *
280  MON
    
```

Assembled Program for Absolute Value Problem

```

00100          NAM      ABS
00110          OPT      M
00120 0000          ORG      0
00130 0000 0001      W      RMB      1
00140 0001 0001      Y      RMB      1
00150 0002 0001      Z      RMB      1
00160 0500          ORG     $0500
00170 0500 96 00          LDA   A W
00180 0502 2A 01          BPL   Z1      IS W POSITIVE?
00190 0504 40          NEG   A      W WAS NEG, MAKE POS.
00200 0505 D6 01      Z1     LDA   B Y
00210 0507 2A 01          BPL   Z2      IS Y POSITIVE?
00220 0509 50          NEG   B      Y WAS NEG, MAKE POS.
00230 050A 1E          Z2     SBA          SUBTRACT Y FROM W
00240 050B 2E 01          BGT   Z3      IS Z POSITIVE?
00250 050D 4F          CLR   A      RESULT WAS ZERO OR NEG.
00260 050E 97 02      Z3     STA   A Z  STORE ANSWER IN Z.
00270 0510 20 FE          PRA   *
00280          MON
    
```

EX-4 Example Programs and Systems



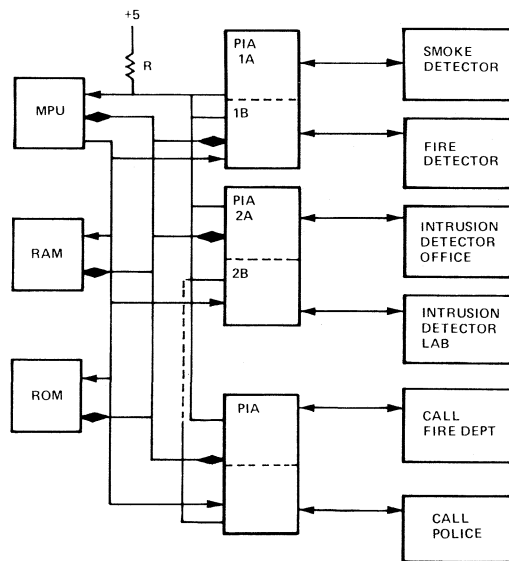
D1537

PIA POLLING ROUTINE

The following routine illustrates one of the various techniques of determining which PIA has generated an interrupt. Recall that each PIA has an A side and a B side which may cause the \overline{IRQ} line to go low thus generating an interrupt. All the PIA interrupt lines are tied together and connected to the one interrupt input pin (\overline{IRQ}) of the MPU. Consequently, when an interrupt is generated, some bit 6 or bit 7 of a PIA is set. The only way to determine where the interrupt came from is to poll bit 6 and bit 7 of each PIA control register to see if it is a "1" (thus an interrupt).

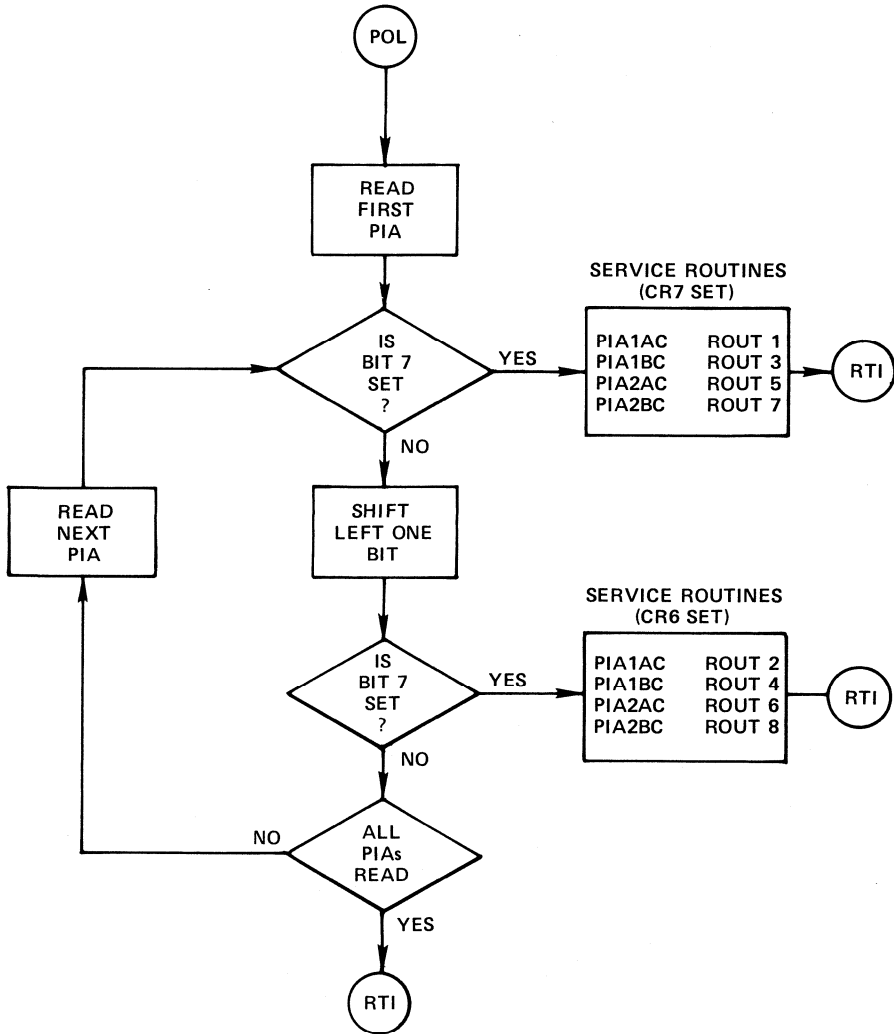
This routine polls the control registers of two PIAs. It reads the contents of each control register and executes the BMI instruction which effectively checks to see if bit 7 is set. If bit 7 is not set, a ROL A instruction is executed which shifts bit 6 into bit 7 thus permitting use of the BMI instruction again. Once a set control bit is detected, it branches to a subroutine to service that particular interrupt. After servicing the interrupt, an RTI instruction is executed which causes the processor to return to whatever it was doing before the interrupt.

PIA POLLING EXAMPLE SYSTEM BLOCK DIAGRAM



TM1127

Flow Chart for PIA Polling Routine



Source Program for PIA Polling Routine

```

100  NAM POLL
110  OPT MEM
120  PIA1AC EQU $4005
130  PIA1BC EQU $4007
140  PIA2AC EQU $4009
150  PIA2BC EQU $400B
200  OP6 $100
210  POLL LDA A PIA1AC
220  BMI POUT1
230  PDL A
240  BMI POUT2
250  LDA A PIA1BC
260  BMI POUT3
270  PDL A
280  BMI POUT4
290  LDA A PIA2AC
300  BMI POUT5
310  PDL A
320  BMI POUT6
330  LDA A PIA2BC
340  BMI POUT7
350  PDL A
360  BMI POUT8
370  RTI
380  POUT1 NOP      ♦THIS IS PIA1AC CB1 SERVICE ROUTINE
390  RTI
400  POUT2 NOP      ♦THIS IS PIA1AC CB2 SERVICE ROUTINE
410  RTI
420  POUT3 NOP      ♦THIS IS PIA1BC CB1 SERVICE ROUTINE
430  RTI
440  POUT4 NOP      ♦THIS IS PIA1BC CB2 SERVICE ROUTINE
450  RTI
460  POUT5 NOP      ♦THIS IS PIA2AC CB1 SERVICE ROUTINE
470  RTI
480  POUT6 NOP      ♦THIS IS PIA2AC CB2 SERVICE ROUTINE
490  RTI
500  POUT7 NOP      ♦THIS IS PIA2BC CB1 SERVICE ROUTINE
510  RTI
520  POUT8 NOP      ♦THIS IS PIA2BC CB2 SERVICE ROUTINE
530  RTI
540  MON

```

TR1131

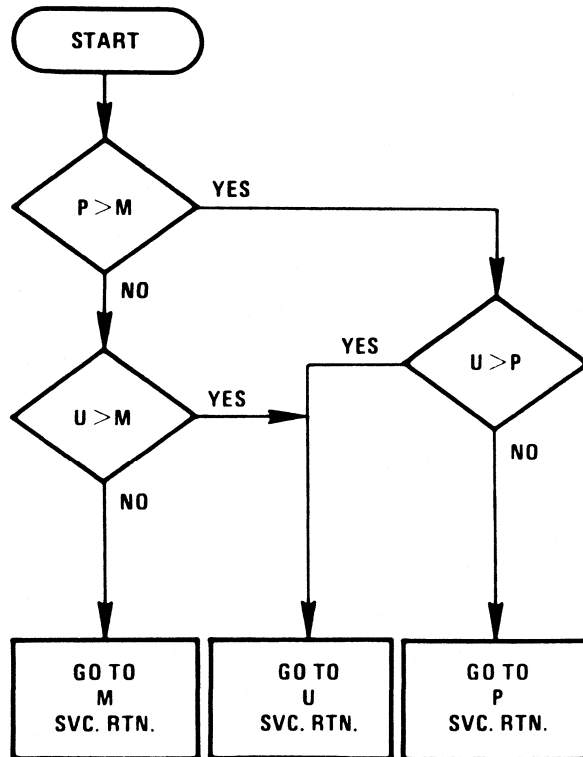
Assembled Program for PIA Polling Routine

00100				NAM	POLL	
00110				DPT	MEM	
00120		4005	PIA1AC	EQU	\$4005	
00130		4007	PIA1BC	EQU	\$4007	
00140		4009	PIA2AC	EQU	\$4009	
00150		400B	PIA2BC	EQU	\$400B	
00200	0100			DRG	\$100	
00210	0100	B6 4005	POLL	LDA A	PIA1AC	
00220	0103	2B 1C		BMI	ROUT1	
00230	0105	49		RDL A		
00240	0106	2B 1B		BMI	ROUT2	
00250	0108	B6 4007		LDA A	PIA1BC	
00260	010B	2B 18		BMI	ROUT3	
00270	010D	49		RDL A		
00280	010E	2B 17		BMI	ROUT4	
00290	0110	B6 4009		LDA A	PIA2AC	
00300	0113	2B 14		BMI	ROUT5	
00310	0115	49		RDL A		
00320	0116	2B 13		BMI	ROUT6	
00330	0118	B6 400B		LDA A	PIA2BC	
00340	011B	2B 10		BMI	ROUT7	
00350	011D	49		RDL A		
00360	011E	2B 0F		BMI	ROUT8	
00370	0120	3B		RTI		
00380	0121	01	ROUT1	NOP		◆THIS IS PIA1AC CA1 SERVICE
00390	0122	3B		RTI		
00400	0123	01	ROUT2	NOP		◆THIS IS PIA1AC CA2 SERVICE
00410	0124	3B		RTI		
00420	0125	01	ROUT3	NOP		◆THIS IS PIA1BC CB1 SERVICE
00430	0126	3B		RTI		
00440	0127	01	ROUT4	NOP		◆THIS IS PIA1BC CB2 SERVICE
00450	0129	3B		RTI		
00460	0129	01	ROUT5	NOP		◆THIS IS PIA2AC CA1 SERVICE
00470	012A	3B		RTI		
00480	012B	01	ROUT6	NOP		◆THIS IS PIA2AC CA2 SERVICE
00490	012C	3B		RTI		
00500	012D	01	ROUT7	NOP		◆THIS IS PIA2BC CB1 SERVICE
00510	012E	3B		RTI		
00520	012F	01	ROUT8	NOP		◆THIS IS PIA2BC CB2 SERVICE
00530	0130	3B		RTI		
00540				MON		

EVENT COUNTER PRIORITY SERVICE ROUTINE

Three event counters are used to monitor a process. The value of each of these counters is read (using PIAs) and stored in memory. A unique service routine is required depending on which counter has the greatest number. If two counters have the same number which is greater than the third, then the priority of service is M (located at \$0), P (located at \$1), then U (located at \$2). The service routine for M largest is at \$E000, P largest is at \$D000, and U largest is at \$C000. Program to start at location \$1000.

Basic Flow Chart



Assembled Program

PAGE 1 GTR 12/03/75 14:12.00

	NAM	GTR
00100	OPT	M
00110	ORG	\$1000
00120 1000		
00130	*LOAD M INTO ACC A FROM \$0	
00140 1000 96 00	LDA A	\$0
00150	*COMPARE ACC A WITH P IN \$1	
00160 1002 91 01	CMP A	\$1
00170	*IF P>M GO TO PGREAT	
00180 1004 25 07	BCS	PGREAT
00190	*COMPARE ACC A WITH U IN \$2	
00200 1006 91 02	CMP A	\$2
00210	*IF U>M GO TO UGREAT	
00220 1008 25 0C	BCS	UGREAT
00230	*OTHERWISE JUMP TO M SERVICE ROUTINE	
00240 100A 7E E000	JMP	\$E000 M IS THE LARGEST
00250	*LOAD P INTO ACC A FROM \$1	
00260 100D 96 01	PGREAT LDA A	\$1
00270	*COMPARE ACC A WITH U IN \$2	
00280 100F 91 02	CMP A	\$2
00290	*IF U>P GO TO UGREAT	
00300 1011 25 03	BCS	UGREAT
00310	*OTHERWISE JUMP TO P SERVICE ROUTINE	
00320 1013 7E D000	JMP	\$D000 P IS THE LARGEST
00330	*U GREATER, JUMP TO U SERVICE ROUTINE	
00340 1016 7E C000	UGREAT JMP	\$C000
00350	MON	

MULTIPLY SUBROUTINE

This subroutine multiplies two 8-bit unsigned binary numbers. The product of the two 8-bit numbers is formed by shifting the multiplier one bit to the right and checking for a one or zero. If a one is present, the multiplicand is added to the product (answer).

The multiplicand is then shifted one bit to the left. This has the effect of multiplying the multiplicand by two. The multiplier is again shifted one bit to the right and the shifted bit checked for a one or zero. If it is a one, the shifted multiplicand is added to the product. The process is repeated until the multiplier has no more ones remaining. When no more ones remain in the multiplier, the problem is finished and the product is the final product.

Example

Multiply $170_{10} \times 5_{10} = 850_{10}$

$170_{10} = AA_{16}$

$5 = 05_{16}$

1010 1010 Multiplicand (M)

0000 0101 Multiplier (N)

This 1 requires the multiplicand M to be added to product.

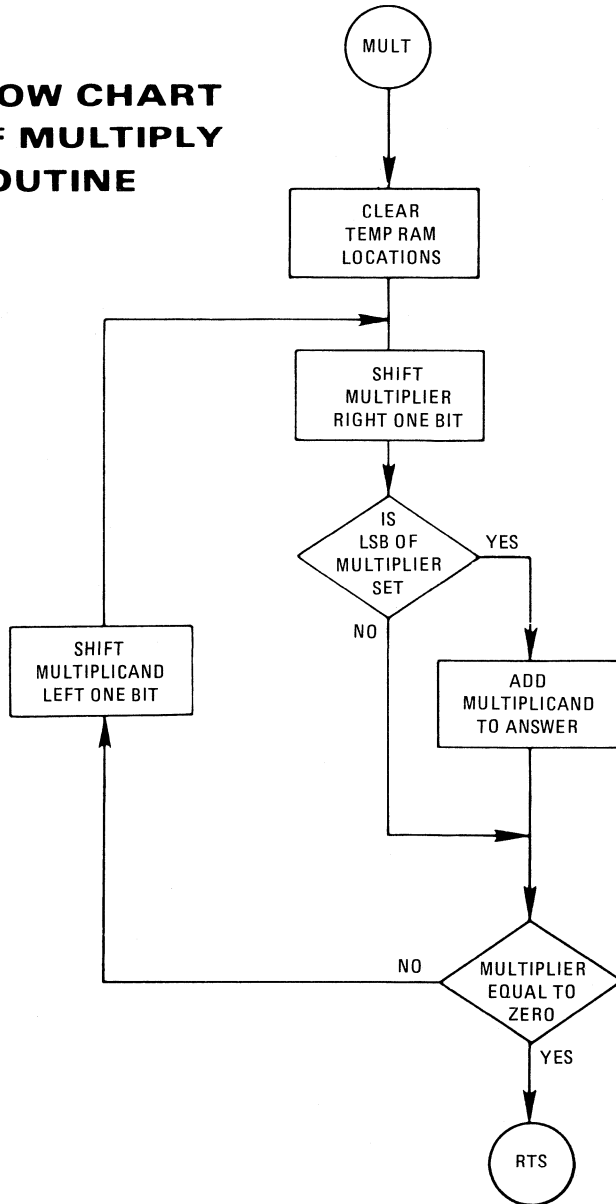
This 1 requires the multiplicand shifted left twice (4 x M) to be added to the product.

Since all remaining higher bits of the multiplier are zero, the problem is finished.

	1010	1010	M	
10	1010	10		
				4 x M
11	0101	0010		
3	5	2		

$AA_{16} \times 5_{16} = 352_{16} = 850_{10}$

FLOW CHART OF MULTIPLY ROUTINE



TR1141

READY
LIST

CMULT 20:52EST 11/11/75

```

100  NAM CMULT
110  OPT M,S
120  *****
130  * REV 003  11-10-75  EAINTEK
140  *
150  * THIS SUBROUTINE MULTIPLIES TWO 8 BIT BYTES.
160  * THE MULTIPLICAND IS STORED IN BYTE NB1.
170  * THE MULTIPLIER IS STORED IN BYTE NB2.
180  * THE RESULT IS STORED IN BYTES ANS2 AND ANS1.
190  * ANS2 IS THE UPPER BYTE OF THE RESULT.
200  * ANS1 IS THE LOWER BYTE OF THE RESULT.
210  *****
220  SPC 1
230  ORG 0
240  NE1A RMB 1 SHIFT MULTIPLICAND STORE
250  NB1 RMB 1 MULTIPLICAND
260  NB2 RMB 1 MULTIPLIER
270  ANS2 RMB 1 UPPER BYTE OF RESULT
280  ANS1 RMB 1 LOWER BYTE OF RESULT
290  SPC 1
300  ORG $10
310  SPC 1
330  MULT CLF A CLEAR ANSWER & SHIFT AREAS
340  STA A NE1A
350  STA A ANS1
360  STA A ANS2
370  LDA A NB2 NE2=MULTIPLIER
380  BRA LOOP1
385  SPC 1
390  LOOP2 ASL NE1 SHIFT MULTIPLICAND LEFT
400  ROL NE1A UPPER BYTE OF MULTIPLICAND
410  LOOP1 LSR A SHIFT MULTIPLIER RIGHT
420  FCC NOADD SHIFT AND DON'T ADD
430  LDA B ANS1 ADD SHIFTED MULTIPLICAND-
440  ADD B NB1 TO ANS1 AND ANS2.
450  STA B ANS1 LOWER BYTE OF RESULT
460  LDA B ANS2
470  ADC B NE1A ADD WITH CARRY
480  STA B ANS2 UPPER BYTE OF RESULT
490  TST A
500  NOADD BNE LOOP2 START SHIFTING AGAIN,
510  RTS FINISHED!!!
520  MON

```

READY

EX-14 Example Programs and Systems

PAGE 1 CMULT 11/11/75 20:55.00

```

00100          NAM    CMULT
00110          OPT    M,S
00120          *****
00130          * REV 003 11-10-75 BAINTEH
00140          *
00150          * THIS SUBROUTINE MULTIPLIES TWO 8 BIT BYTES.
00160          * THE MULTIPLICAND IS STORED IN BYTE NR1.
00170          * THE MULTIPLIER IS STORED IN BYTE NR2.
00180          * THE RESULT IS STORED IN BYTES ANS2 AND ANS1.
00190          * ANS2 IS THE UPPER BYTE OF THE RESULT.
00200          * ANS1 IS THE LOWER BYTE OF THE RESULT.
00210          *****

00230 0000          ORG    0
00240 0000 0001    NR1A  RMB  1      SHIFT MULTIPLICAND STORE
00250 0001 0001    NR1   RMB  1      MULTIPLICAND
00260 0002 0001    NR2   RMB  1      MULTIPLIER
00270 0003 0001    ANS2  RMB  1      UPPER BYTE OF RESULT
00280 0004 0001    ANS1  RMB  1      LOWER BYTE OF RESULT

00300 0010          ORG    $10

00330 0010 4F      MULT  CLR A      CLEAR ANSWER & SHIFT AREAS
00340 0011 97 00    STA A  NR1A
00350 0013 97 04    STA A  ANS1
00360 0015 97 03    STA A  ANS2
00370 0017 96 02    LDA A  NR2      NR2=MULTIPLIER
00380 0019 20 06    PRA    LOOP1

00390 001E 78 0001 LOOP2 ASL  NR1    SHIFT MULTIPLICAND LEFT
00400 001E 79 0000      ROL  NR1A   UPPER BYTE OF MULTIPLICAND
00410 0021 44          LOOP1 LSR A    SHIFT MULTIPLIER RIGHT
00420 0022 24 0D      FCC    NOADD   SHIFT AND DON'T ADD
00430 0024 D6 04      LDA B  ANS1   ADD SHIFTED MULTIPLICAND-
00440 0026 DB 01      ADD B  NB1    TO ANS1 AND ANS2.
00450 0028 D7 04      STA B  ANS1   LOWER BYTE OF RESULT
00460 002A D6 03      LDA B  ANS2
00470 002C D9 00      ADC B  NR1A   ADD WITH CARRY
00480 002E D7 03      STA B  ANS2   UPPER BYTE OF RESULT
00490 0030 4D          TST A
00500 0031 26 E8      NOADD BNE  LOOP2  START SHIFTING AGAIN
00510 0033 39          RTS
00520          MON

```

SYMBOL TABLE

ANS1	0004	ANS2	0003	LOOP1	0021	LOOP2	001E	MULT	0010	TR1143
NB1	0001	NR1A	0000	NR2	0002	NOADD	0031			

RUN MPSSIM

MPSSIM 21:01EST 11/11/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPSSIM
COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU SIMULATOR, RELEASE 1.3

ENTER MF FILENAME
?CMULT1

*INPUT AND DISPLAY
BASE IS HEX*

FILE'S LABEL:

MULT1 IS MF FOR CMULT PROGRAM SIMULATION.

HH IA OC EA P X A F C S T
*0000 *** *0000*0001 0000 00 00 000Z00 00F2 0000104
?ML

MACRO LIBRARY LISTING

TH (CSR P10,S00F0,T0.SM1,#1,#2.T50.DM0,5)
GO (CSR P10,S00F0,T0.SM1,#1,#2.R100.DM0,5)
RES (DM0,5)

*LIST OF MACROS THAT
WERE DEFINED BEFORE
IN A RUN OF CMULT1
ON MPSSIM*

1132 REMAINING CHARACTERS

TR OAA,0

*TRACE WITH NB1 = AA₁₆ = 170₁₀
NB2 = 0.*

*0010 CLR A*0010*0011 0000 00 00 000Z00*00F0 000000?
*0011 STA A*0000*0013 0000 00 00 000Z00 00F0 0000006
*0013 STA A*0004*0015 0000 00 00 000Z00 00F0 0000010
*0015 STA A*0003*0017 0000 00 00 000Z00 00F0 0000014
*0017 LDA A*0002*0019 0000 00 00 000Z00 00F0 0000017
*0019 BRA *001A*0021 0000 00 00 000Z00 00F0 0000021
*0021 LSR A*0021*0022 0000 00 00 000Z00 00F0 0000023
*0022 BCC *0023*0031 0000 00 00 000Z00 00F0 0000027
*0031 ENE *0032*0033 0000 00 00 000Z00 00F0 0000031
HH IA OC EA P X A F C S T
*0033 RTS *00F2*0000 0000 00 00 000Z00*00F2 0000036
INST FAULT
*0000 *** *0000*0001 0000 00 00 000Z00 00F2 0000038

*NOTE NUMBER OF
MACHINE CYCLES
REQUIRED TO
COMPLETE
MULTIPLICATION*

?RES

*RES MACRO WILL
DISPLAY MEMORY
LOCATIONS STARTING
AT LOCATION 00 (ZERO).
THE NUMBER OF
LOCATIONS DISPLAYED
IS FIVE.*

0000 00 AA 00 00 00

ANS 2 ↑ ↑ ANS 1

∴ RESULT OR ANSWER = 00 00.

TR1144

EX-16 Example Programs and Systems

$NB1 = 5_{16} = 5_{10}$
 $NB2 = 5_{16} = 5_{10}$
 $5_{10} \times 5_{10} = 25_{10} = 19_{16}$

```

7TR 5,5 ← TRACE
*0010 CLR A*0010*0011 0000 00 00 000700*00F0 0000002
*0011 STA A*0000*0013 0000 00 00 000Z00 00F0 0000006
*0013 STA A*0004*0015 0000 00 00 000Z00 00F0 0000010
*0015 STA A*0003*0017 0000 00 00 000Z00 00F0 0000014
*0017 LDA A*0002*0019 0000*05 00 000000 00F0 0000017
*0019 FRA *001A*0021 0000 05 00 000000 00F0 0000021
*0021 LSR A*0021*0022 0000*02 00 0000VC 00F0 0000023
*0022 FCC *0023*0024 0000 02 00 0000VC 00F0 0000027
HH IA OC FA P X A B C S T
*0024 LDA B*0004*0026 0000 02 00 000Z0C 00F0 0000030
*0026 ADD B*0001*0028 0000 02*05 000000 00F0 0000033
*0028 STA B*0004*002A 0000 02 05 000000 00F0 0000037
*002A LDA B*0003*002C 0000 02*00 000Z00 00F0 0000040
*002C ADC B*0000*002E 0000 02 00 000Z00 00F0 0000043
*002E STA B*0003*0030 0000 02 00 000Z00 00F0 0000047
*0030 TST A*0030*0031 0000 02 00 000000 00F0 0000049
*0031 BNE *0032*001B 0000 02 00 000000 00F0 0000053
*001B ASL *0001*001E 0000 02 00 000000 00F0 0000059
*001E ROL *0000*0021 0000 02 00 000700 00F0 0000065
HH IA OC EA P X A B C S T
*0021 LSR A*0021*0022 0000*01 00 000000 00F0 0000067
*0022 FCC *0023*0031 0000 01 00 000000 00F0 0000071
*0031 BNE *0032*001E 0000 01 00 000000 00F0 0000075
*001E ASL *0001*001E 0000 01 00 000000 00F0 0000081
*001E ROL *0000*0021 0000 01 00 000Z00 00F0 0000087
*0021 LSR A*0021*0022 0000*00 00 000ZVC 00F0 0000089
*0022 BCC *0023*0024 0000 00 00 000ZVC 00F0 0000093
*0024 LDA B*0004*0026 0000 00*05 00000C 00F0 0000096
*0026 ADD B*0001*0028 0000 00*19 000000 00F0 0000099
*0028 STA B*0004*002A 0000 00 19 000000 00F0 0000103
HH IA OC EA P X A B C S T
*002A LDA B*0003*002C 0000 00*00 000Z00 00F0 0000106
*002C ADC B*0000*002E 0000 00 00 000700 00F0 0000109
*002E STA B*0003*0030 0000 00 00 000Z00 00F0 0000113
*0030 TST A*0030*0031 0000 00 00 000Z00 00F0 0000115
*0031 BNE *0032*0033 0000 00 00 000Z00 00F0 0000119
*0033 RTS *00F2*0000 0000 00 00 000700*00F2 0000124
INST FAULT
*0000 *** *0000*0001 0000 00 00 000Z00 00F2 0000126
?RES

```

0000 00 14 05 00 19 ANSWER IS CORRECT $19_{16} = 25_{10}$ TR1191

RUN $NB1 = AA_{16} = 170_{10}$
 $NB2 = AA_{16} = 170_{10}$
 $170_{10} \times 170_{10} = 28900_{10} = 70E4_{16}$

```

?GO OAA, OAA
INST FAULT
0000 *** B 0000 0001 0000 00*70 H00Z00 00F2 0000280
?DL
    
```

```

*0033 RTS *00F2*0000 0000 00 70 H00Z00 00F2 0000278
?RES
    
```

RESULT
 0000 55 00 AA **70 E4** U....
 ?GO **(4,4)**

$4_{16} \times 4_{16} = 4_{10} \times 4_{10} = 16_{10} = 10_{16}$

```

*0000 *** *0000*0001 0000 00*00 000Z00 00F2 0000104
?DL
    
```

DISPLAY LAST INSTRUCTION

HH	IA	OC	EA	P	X	A	B	C	S	T
*0033	RTS	*00F2*0000	0000	00	00	00	00	00Z00	00F2	0000102

```

?RES
    
```

RESULT
 0000 00 10 04 00 **(10)**.....
 ?GO **(0A,0A)**

$A_{16} \times A_{16} = 10_{10} \times 10_{10} = 100_{10} = 64_{16}$

```

INST FAULT
*0000 *** *0000*0001 0000 00 00 000Z00 00F2 0000148
?DL
    
```

```

*0033 RTS *00F2*0000 0000 00 00 000Z00 00F2 0000146
?RES
    
```

RESULT
 0000 00 50 0A 00 **(64)**.P...
 ?GO **(OFF,OFF)**

$FF_{16} \times FF_{16} = 255_{10} \times 255_{10} = 65025_{10} = FE01_{16}$

```

MEMORY FAULT AT 80FF
*0000 CLR *80FF*0003 0000 00*FE H00Z00 00F2 0000372
?DL
    
```

```

*0033 RTS *00F2*0000 0000 00 FE H00Z00 00F2 0000366
?RES
    
```

RESULT
 0000 7F 80 FF **(FE 01)**.....
 ?EX

PROGRAM STOP AT 0

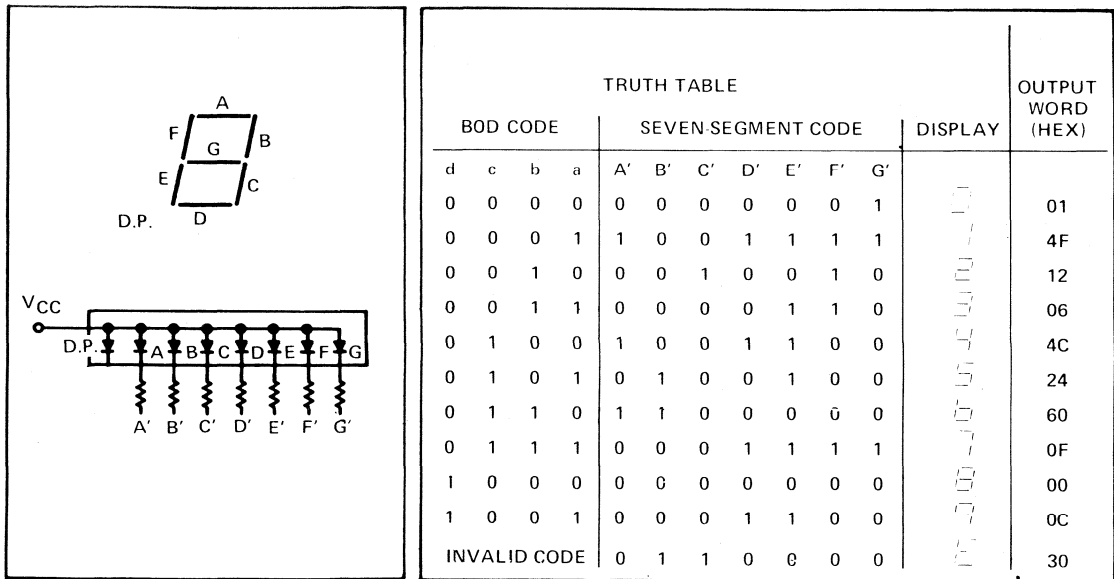
THIS IS THE WORST CASE
 TIME REQ'D. TO MULTIPLY
 WITH THE LARGEST 8 BIT
 NUMBER. TIME INCLUDES
 THE RTS INSTRUCTION.

TR1146

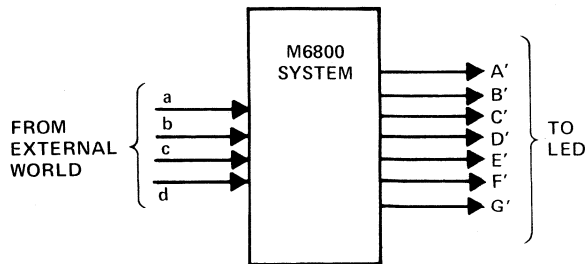
SAMPLE SYSTEM PROBLEM – BCD to LED (Table Look-Up)

As an example of a system problem, assume a system composed of a 7-segment LED and a 4-wire BCD signal. The object will be to use an M6800 to convert the BCD signal to the 7-segment code necessary to the LED.

Even though the same task could be done with one TTL IC (7447), this problem illustrates not only a very simple complete system, but more important it demonstrates one method of using a look-up table without getting entangled with more complex concepts.

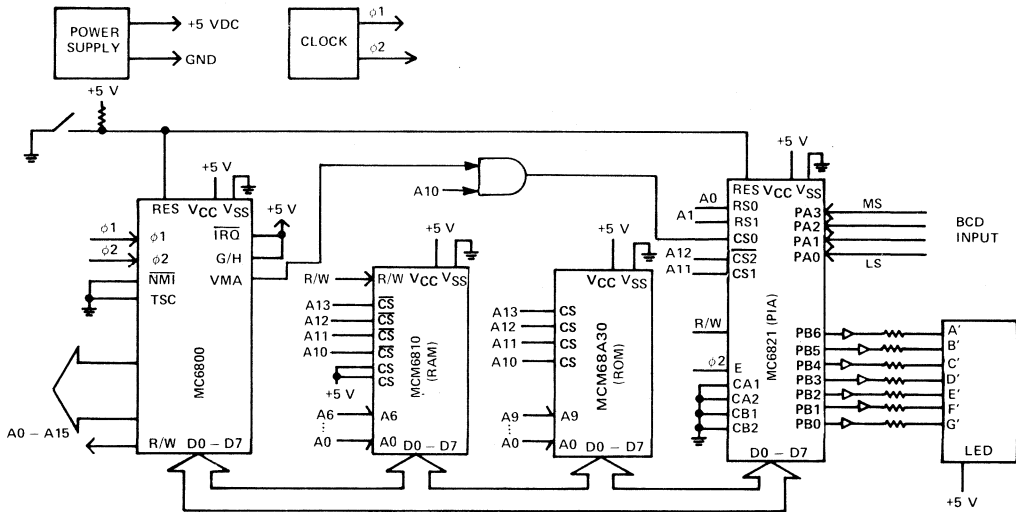


TR1133



TR1134

SYSTEM CONFIGURATION



- | | |
|---------------|---------------|
| PIA ADDRESSES | 0C00 - PIA1AD |
| | 0C01 - PIA1AC |
| | 0C02 - PIA1BD |
| | 0C03 - PIA1BC |
| ROM ADDRESSES | 3C00 - 3FFF |
| RAM ADDRESSES | 0000 - 007F |

TR1135-3

SOURCE LISTING

LIST

```

BCDLED      20:29EST      11/10/75

100  NAM BCDLED
110  OPT M
115  SPC 1
120  ORG 0
130  INDEX RMB 2
140  ORG $0C00 PIA ADDRESSES
150  PIA1AD RMB 1
160  PIA1AC RMB 1
170  PIA1BD RMB 1
180  PIA1BC RMB 1
190  ORG $3F00 BUILD TABLE
200  TABLE FCB $1,$4F,$12,$6,$4C,$24,$60,$F,,,$C
210  FCB $30,$30,$30,$30,$30,$30 ERROR INPUTS
220  ORG $3FFE
230  FDP START RESTART VECTOR
235  SPC 1
240  ORG $3C00 BEGIN PROGRAM
250  START LDA A #$FF
260  STA A PIA1BD B-SIDE ALL OUTPUTS
270  LDA A #100000100
280  STA A PIA1AC
290  STA A PIA1BC
300  LDX #TABLE GET STARTING ADR OF TABLE
310  STX INDEX
320  LOOP LDA A PIA1AD READ PCD INPUT
330  AND A #100001111 MASK 4 MSB
340  STA A INDEX+1
350  LDX INDEX
360  LDA A 0,X
370  STA A PIA1BD OUTPUT TO LED
380  BRA LOOP DO IT AGAIN
390  MON

```

READY

TR1194

ASSEMBLY LISTING

ENTER SI FILENAME
?BCDLED

ENTER MF FILENAME
?LEDSIM:C

FILE'S LABEL:
MF FOR SIMULATION OF BCDLED PROGRAM

PAGE 1 BCDLED 11/10/75 20:30.00

```

00100          NAM      BCDLED
00110          OPT      M

00120 0000          ORG      0
00130 0000 0002    INDEX  RMB      2
00140 0C00          ORG      $0C00      PIA ADDRESSES
00150 0C00 0001    PIA1AD RMB      1
00160 0C01 0001    PIA1AC RMB      1
00170 0C02 0001    PIA1BD RMB      1
00180 0C03 0001    PIA1PC RMB      1
00190 3F00          ORG      $3F00      BUILD TABLE
00200 3F00 01      TABLE FCB      $1, $4F, $12, $6, $4C, $24, $60, $F, , $C
          3F01 4F
          3F02 12
          3F03 06
          3F04 4C
          3F05 24
          3F06 60
          3F07 0F
          3F08 00
          3F09 0C
00210 3F0A 30          FCB      $30, $30, $30, $30, $30, $30      ERROR INPU
          3F0B 30
          3F0C 30
          3F0D 30
          3F0E 30
          3F0F 30
00220 3FFE          ORG      $3FFE
00230 3FFE 3C00      FDP      START      RESTART VECTOR

00240 3C00          ORG      $3C00      BEGIN PROGRAM
00250 3C00 86 FF    START  LDA  A      #$FF
00260 3C02 B7 0C02  STA  A      PIA1BD      B-SIDE ALL OUTPUTS
00270 3C05 86 04    LDA  A      #$00000100
00280 3C07 B7 0C01  STA  A      PIA1AC
00290 3C0A B7 0C03  STA  A      PIA1BC
00300 3C0D CE 3F00  LDX  #TABLE      GET STARTING ADR OF TABLE
00310 3C10 DF 00    STX  INDEX
00320 3C12 B6 0C00 LOOP  LDA  A      PIA1AD      READ BCD INPUT
00330 3C15 84 0F    AND  A      #$00001111      MASK 4 MSB
00340 3C17 97 01    STA  A      INDEX+1
00350 3C19 DE 00    LDX  INDEX
00360 3C1B A6 00    LDA  A      0,X
00370 3C1D B7 0C02  STA  A      PIA1BD      OUTPUT TO LED
00380 3C20 20 F0    BRA  LOOP      DO IT AGAIN
00390          MON

```

TR1193

SIMULATION

RUN MPSSIM

MPSSIM 20:35EST 11/10/75

MOTOROLA SPD, INC. OWNS AND IS RESPONSIBLE FOR MPSSIM
 COPYRIGHT 1973 & 1974 BY MOTOROLA INC

MOTOROLA MPU SIMULATOR, RELEASE 1.3

ENTER MF FILENAME
 ?LEDSIM:C

FILE'S LABEL:
 MF FOR SIMULATION OF BCDLED PROGRAM
 HH IA OC EA P X A B C S T
 0000 *** 0000 0000 0000 00 00 000000 0000 00000000
 ?ML

MACRO LIBRARY LISTING
 TRA [SR P3C00,S007F,T0,SM0C00,#1,RP3C20,T20]
 RUN [SR P3C00,S007F,T0,SM0C00,#1,RP3C20,R20]
 RES [DM0C02]

1124 REMAINING CHARACTERS
 ?TRA 0

```
*3C00 LDA A*3C01*3C02 0000*FF 00 00N000*007F 0000002
*3C02 STA A*0C02*3C05 0000 FF 00 00N000 007F 0000007
*3C05 LDA A*3C06*3C07 0000*04 00 000000 007F 0000009
*3C07 STA A*0C01*3C0A 0000 04 00 000000 007F 0000014
*3C0A STA A*0C03*3C0D 0000 04 00 000000 007F 0000019
*3C0D LDX *3C0F*3C10*3F00 04 00 000000 007F 0000022
*3C10 STX *0001*3C12 3F00 04 00 000000 007F 0000027
*3C12 LDA A*0C00*3C15 3F00*00 00 000Z00 007F 0000031
*3C15 AND A*3C16*3C17 3F00 00 00 000Z00 007F 0000033
HH IA OC EA P X A B C S T
*3C17 STA A*0001*3C19 3F00 00 00 000Z00 007F 0000037
*3C19 LDX 0001*3C1B 3F00 00 00 000000 007F 0000041
*3C1B LDA A*3F00*3C1D 3F00*01 00 000000 007F 0000046
*3C1D STA A*0C02*3C20 3F00 01 00 000000 007F 0000051
BKPT AT 3C20
*3C20 ERA *3C21*3C12 3F00 01 00 000000 007F 0000055
?RES
```

0C02 01 .

TR1192

EX-24 Example Programs and Systems

?SMOC00,33

?T20

```
*3C12 LDA A*0C00*3C15 3F07*33 00 000000 007F 0000059
HH IA OC EA P X A B C S T
*3C15 AND A*3C16*3C17 3F07*03 00 000000 007F 0000061
*3C17 STA A*0001*3C19 3F07 03 00 000000 007F 0000065
*3C19 LDX 0001*3C1B*3F03 03 00 000000 007F 0000069
*3C1E LDA A*3F03*3C1D 3F03*06 00 000000 007F 0000074
*3C1D STA A*0C02*3C20 3F03 06 00 000000 007F 0000079
BKPT AT 3C20
*3C20 BRA *3C21*3C12 3F03 06 00 000000 007F 0000083
?SMOC00,7B.T20
```

```
*3C12 LDA A*0C00*3C15 3F03*7B 00 000000 007F 0000087
*3C15 AND A*3C16*3C17 3F03*0B 00 000000 007F 0000089
*3C17 STA A*0001*3C19 3F03 0B 00 000000 007F 0000093
*3C19 LDX 0001*3C1B*3F0B 0B 00 000000 007F 0000097
HH IA OC EA P X A B C S T
*3C1B LDA A*3F0B*3C1D 3F0B*30 00 000000 007F 0000102
*3C1D STA A*0C02*3C20 3F0B 30 00 000000 007F 0000107
BKPT AT 3C20
*3C20 BRA *3C21*3C12 3F0B 30 00 000000 007F 0000111
?RES
```

0C02 30 0
?RUN 9

```
BKPT AT 3C20
3C20 BRA 3C21 3C12*3F09*0C 00 000000 007F 0000055
?RES
```

0C02 0C .
?RUN 5

```
BKPT AT 3C20
3C20 BRA 3C21 3C12*3F05*24 00 000000 007F 0000055
?RES
```

0C02 24 \$
?RUN 2E

```
BKPT AT 3C20
3C20 BRA 3C21 3C12*3F0E*30 00 000000 007F 0000055
?RES
```

0C02 30 0
?RS.EX

PROGRAM STOP AT 0

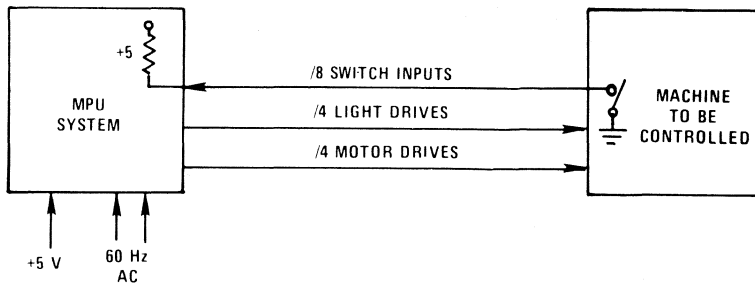
TR1195

SAMPLE SYSTEM – Machine Control

The following system description is that of an MC6800 controlling a hypothetical machine. This machine may be part of an industrial or commercial process and could be involved in manufacturing such as photographic processing or other assembly-line operations. While the application is imaginary, it does serve to illustrate how hardware and software marry and that each has its proper place in any system architecture. Although the system will function, its purpose is the illustration of techniques and the intention to clarify the configuring of an entire system.

Function of System

The input from the machine to be controlled consists of eight switches, manual and/or cam-operated that provide the input information to the electronics about the machine's condition. As these switches are mechanical, their output contains severe bouncing which must be eliminated by some method. The driven items (controlled outputs) are four incandescent lamps for operator indication or optical control functions within the machine, and four AC motors driving various parts.



TR 1199

The processor system examines the state of the eight inputs after performing a software debounce, and calculates an appropriate output based on this information. It then gives this output to the lamps and motors in a judicious fashion.

System Operation

A system schematic is shown in the following pages. The clock consists of a cross-coupled monostable, MC8602 (see pages 4-10, application manual), with an MC3459 driver and the restart circuit is an MC1455 (pages 4-43 of same). Interrupts will be given to the system via CA1 and CB1 of the PIA from a one-shot, MC74121. The Q output goes high, close to the zero crossing of the ac line and come low about 4 ms later, at the peak of the line cycle.

EX-26 Example Programs and Systems

The restart routine sets the stack pointer, initializes the PIA, clears out some RAM locations, clears the interrupt mask, then falls into the executive code. The executive code is a loop which runs continuously, looking at the eight inputs and comparing them to a look-up table. This table is composed of two-byte pairs. The first byte of a pair is indicative of a certain input combination, and the second byte is the corresponding output pattern that would be given to the lights and motors, if the first byte should be a match with the actual inputs. The reader should notice that the executive code takes its input word not from the PIA, but from RAM—at EXINP—and gives its output not to the PIA, but to RAM—at EXOUT. Therefore, the executive code does not handle the system tasks of I/O; these are taken care of in the interrupt routines.

Notice that CA1 and CB1 are tied together and are driven by the A output of the one-shot. Line CA1 has been programmed to be sensitive to a rising edge and CB1 a falling edge. Since the one-shot is driven by the ac line, we have indicators in terms of interrupts corresponding to the zero-crossing (CA1) and peak (CB1) times of the line cycle.

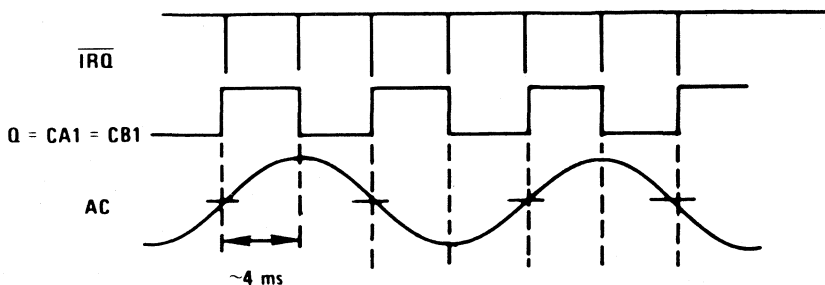
The system makes use of three interrupts. Valley, Peak, and NMI.

The Valley interrupt comes from CA1 and occurs near the zero-cross of the ac line. The lights are turned on at this zero-current time, greatly increasing their life and reducing inrush stress on the SCRs. This output data is taken from RAM location EXOUT, the executive output. Valley has the second function of bringing in the input information from PIAPA, conditioning (debouncing) it, and storing it at EXINP, the input for the executive. Debouncing is performed by checking the inputs at 8 ms intervals; and, if they are equal for three consecutive times, the data is stored in EXINP, thus eliminating external hardware which would otherwise be required.

The Peak interrupt comes from CB1 and occurs near the peak of the line cycle. This is the optimum time to turn off inductive loads, and the only function of this interrupt is to output the four bits of motor data from EXOUT to PIAPB.

There is a third interrupt, NMI. Here it is used to indicate the impending loss of power. In order not to leave the motors and lights running uncontrolled because of a power-supply or other failure, NMI turns off all outputs and goes into an infinite loop with no escape.

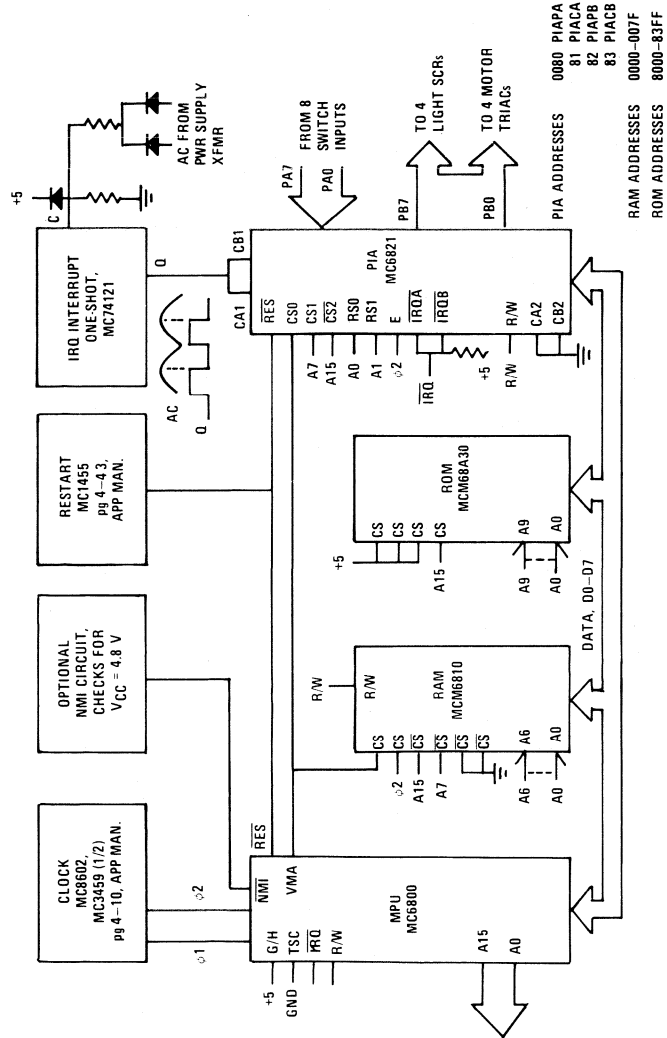
If an oscilloscope were hung on the IRQ line, it would look like an inverted picket fence timed to the line frequency.



TR1200

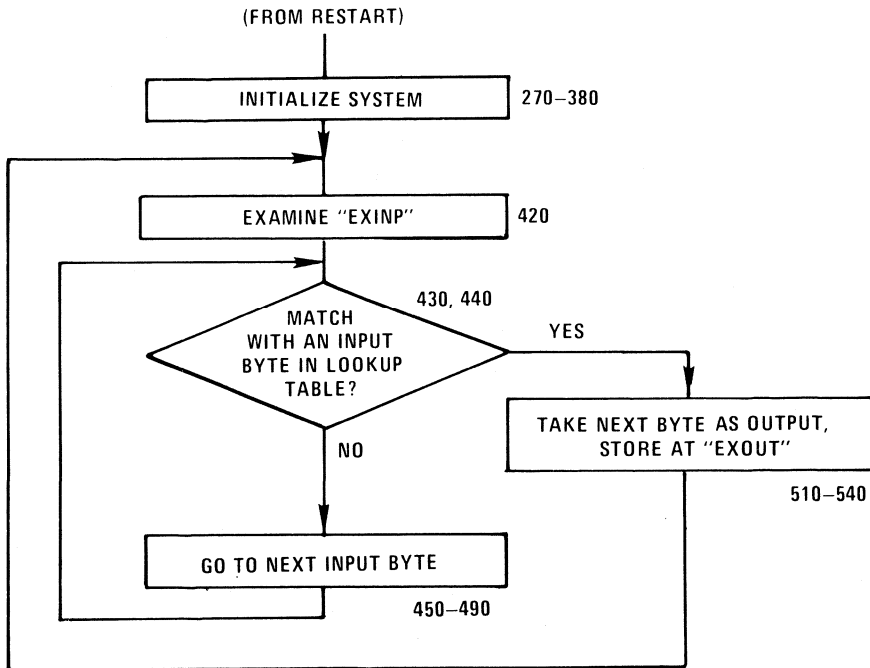
It is obvious that a processor such as the MC6800 is not fully utilized in this application. It could perform several other functions in addition to controlling this machine, or control many of them simultaneously.

SYSTEM CONFIGURATION



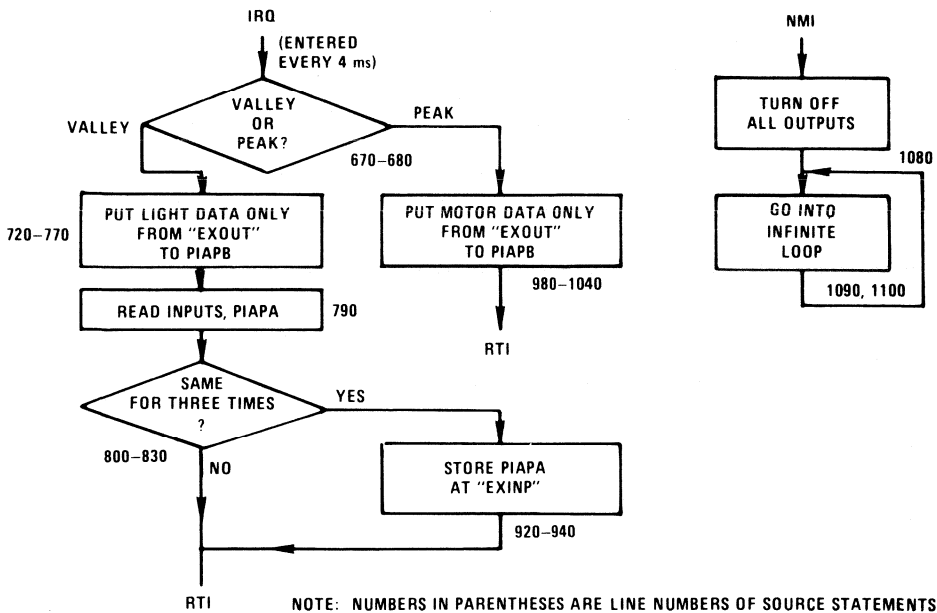
TR1203.3

SOFTWARE FLOW



TR1201

INTERRUPT ROUTINES



TR1202

PAGE 2 HYDER 11/03/75 11:21.00

```

00640
00650
00660
00670 3049 96 33 POLL LDA A PIACB GET CB
00680 304B 2B 27 BMI PEAK VALLEY OR PEAK INTERRUPT?
00690
00700 *****INTERRUPT FOR VALLEY (LIGHTS & INPUTS)*****
00710
00720 304D 86 F0 VALLEY LDA A #$F0
00730 304F 94 02 AND A EXOUT OUTPUT 4 BITS OF LIGHT
00740 3051 D6 32 LDA B PIAPB DATA ONLY W/O CHANGING
00750 3053 C4 0F AND B #$0F MOTOR OUTPUTS
00760 3055 1B ABA
00770 3056 97 32 STA A PIAPB
00780
00790 3058 96 30 LDA A PIAPA INPUTS SAME AS
00800 305A 91 03 CMP A TEMP LAST TIME? CLEARS INTERRUPT
00810 305C 27 06 BEQ SAME
00820 305E 97 03 STA A TEMP STORE NEW DATA
00830 3060 7F 0001 CLR COUNT ZERO COUNTER
00840 3063 3B RTI GO BACK TO EXEC
00850
00860 3064 C6 01 SAME LDA B #01 THIRD TIME MATCH?
00870 3066 D1 01 CMP B COUNT
00880 3068 27 04 BEQ GOODIN IF SO, GO TO GOODIN
00890 306A 7C 0001 INC COUNT IF NOT, JUST INC COUNTER
00900 306D 3B RTI AND RETURN
00910
00920 306E 97 00 GOODIN STA A EXINP PUT GOOD DATA IN RAM
00930 3070 7F 0001 CLR COUNT
00940 3073 3B RTI
00950
00960 *****INTERRUPT FOR PEAK (MOTOR) OUTPUTS*****
00970
00980 3074 86 0F PEAK LDA A #$0F OUTPUT 4 BITS OF
00990 3076 94 02 AND A EXOUT MOTOR DATA W/O CHANGING
01000 3078 D6 32 LDA B PIAPB LIGHT DATA, ALSO CLEARS INT
01010 307A C4 0F AND B #$F0
01020 307C 1B ABA
01030 307D 97 32 STA A PIAPB
01040 307F 3B RTI
01050
01060 *****OPTIONAL NMI INTERRUPT*****
01070
01080 3080 7F 0082 NMI CLR PIAPB TURN OFF ALL OUTPUTS
01090 3083 01 HANGUP NOP
01100 3084 20 FD BRA HANGUP GO TO SLEEP
01110
01120 *****SET VECTORS IN UPPER ROM*****
01130
01140 83F8 ORG $83F8
01150 83F8 3049 FDB POLL,0000,NMI,RESTRT
01160
01170 MON

```

TR1205

ACIA MEMORY LOAD/DUMP PROGRAM

Example

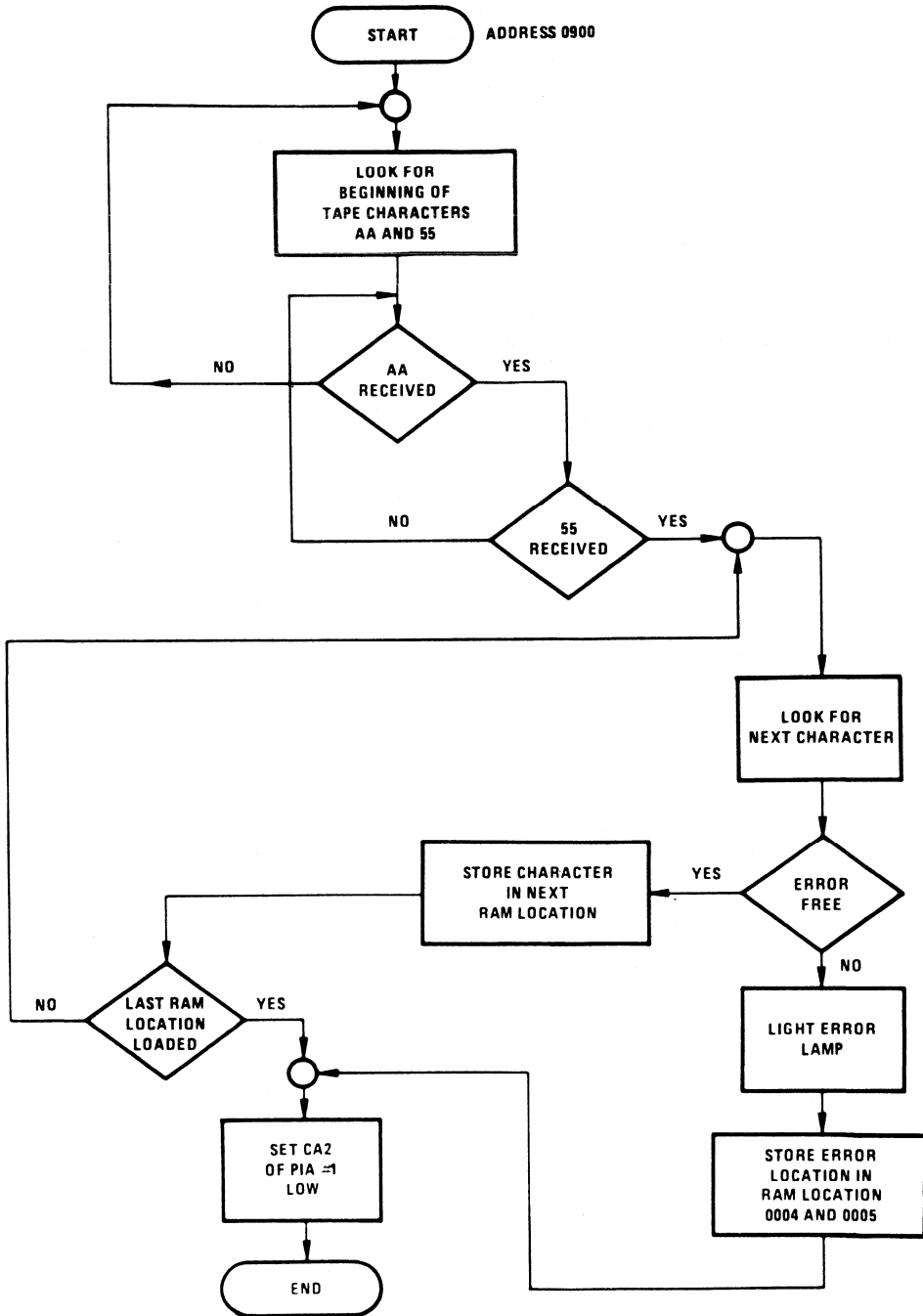
Assume three 128 x 8 MCM6810 RAMs in a system. It is desired to load a 256-byte program into the two upper RAMs starting with address 0080.

Hand load:	RAM Loc.	Value
	0000	00
	0001	80
	0002	01
	0003	80
Starting Vector		0000
Stopping Vector (Last Address + 1)		007F
		0080
		00FF
		0100
		017F

After the hand-loading of the starting and stopping vectors, the load program is executed by starting the MPU at program address 0900. When the program has finished loading, the CA2 line of PIA1 will go low. This signal can be used to stop the tape recorder or turn off a lamp to indicate the end of the loading process.

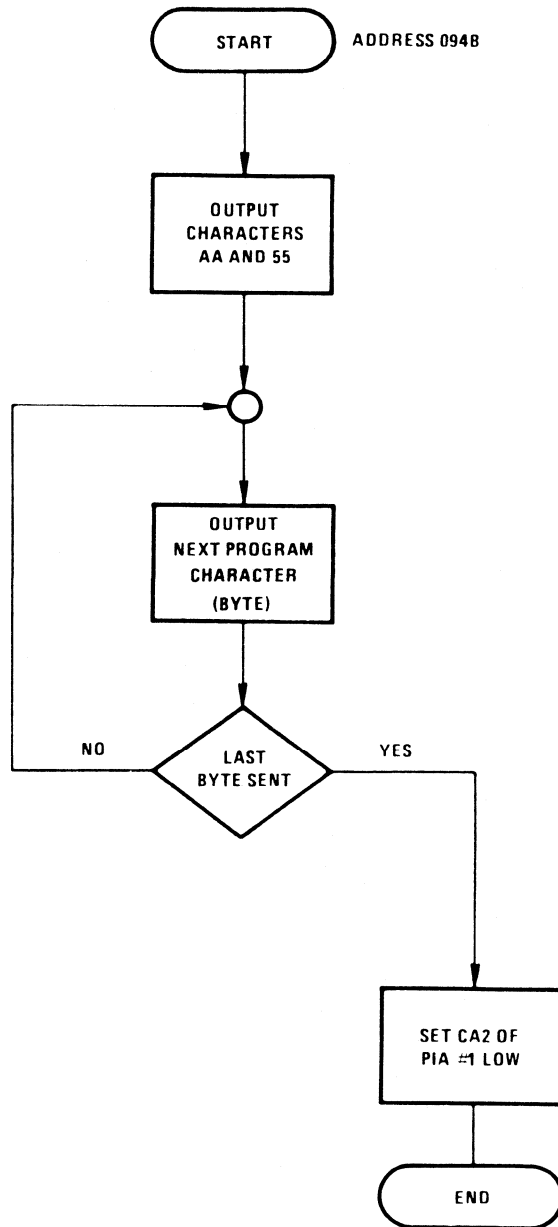
The *memory dump program* works as follows: The start-and-stop memory dump addresses or vectors are hand-loaded into RAM locations 0000, 0001, 0002, and 0003 in the same manner as in the previous load program. Program execution begins at memory address 094B. The characters AA and 55 are first dumped or placed on the tape in order to indicate the beginning of memory dump or listing. Each program character or byte is dumped via the ACIA and Modem until the last memory location has been addressed and dumped. When the dump operation is complete, the CA2 lead of PIA1 will go low, indicating dump complete.

Load via ACIA (MC6850)



EX-34 Example Programs and Systems

Dump via ACIA (MC6850)



Source Program for Load/Dump via ACIA Program

```

1.000  NAM LIBOOT
2.000  OPT M
3.000  ♦ THIS PROGRAM LOADS OR DUMPS MEMORY
4.000  ♦ PLACE START ADDRESS IN LOC 00 & 01
5.000  ♦ PLACE END ADDRESS + 1 IN LOC 02 & 03
6.000  ♦ IF ERROR OCCURS, CHECK LOC 04 & 05 FOR ADDRESS.
7.000  ♦ CA2 STOPS DRIVE AT EOT OR ERROR.
8.000  ♦ CB2 GIVES ERROR INDICATION.
9.000  ♦ DUMP PROGRAM STARTS AT LOC 094B.
10.000 PIA1AC EQU $0805
11.000 PIA1BC EQU $0807
12.000 ACIAC EQU $0806
13.000 ACIAD EQU $0809
14.000  ORG $0900
15.000  LDA A #$03
16.000  STA A ACIAC ACIA MASTER RESET
17.000  LDX $00 LOAD START ADDRESS
18.000  LDA A #$19 ACIA 8 BITS EVEN PARITY
19.000  STA A ACIAC
20.000  LOOP LDA A ACIAC
21.000  ROR A
22.000  BCC LOOP RECEIVER FULL?
23.000  LDA A ACIAD
24.000  CMPAA CMP A #$AA IS FIRST CHAR "AA"?
25.000  BNE LOOP BRANCH IF NOT
26.000  LOOP1 LDA A ACIAC
27.000  ROR A
28.000  BCC LOOP1
29.000  LDA A ACIAD
30.000  CMP A #$55 IS SECOND CHAR "55"?
31.000  BNE CMPAA IF NOT, TRY FOR AN "AA"
32.000  LOOP2 LDA A ACIAC
33.000  TAB TRANSFER A TO B
34.000  AND B #$70
35.000  BNE ERROR BRANCH IF ERROR
36.000  ROR A
37.000  BCC LOOP2
38.000  LDA A ACIAD LOAD A CHAR FROM TAPE
39.000  STA A 0,X STORE IN MEMORY
40.000  INX INCREMENT ADDRESS
41.000  CPX $02 LOAD COMPLETED?
42.000  BNE LOOP2 GO GET MORE
43.000  END LDA A #$30
44.000  STA A PIA1AC TURN OFF CA2
45.000  BRA ♦
46.000  ERROR LDA A #$36
47.000  STA A PIA1BC TURN ON ERROR LIGHT
48.000  STX $04 STORE ADR OF ERROR
49.000  BRA END
50.000  PAGE
51.000  LDX $00      ♦START OF DUMP PROGRAM

```

Source Program for Load/Dump via ACIA Program (continued)

```
52.000 LDA A #$19
53.000 STA A ACIAC
54.000 LDA A #$AA ♦OUTPUT CONTROL CHAR
55.000 STA A ACIAD
56.000 LOOP5 LDA A ACIAC
57.000 ROR A
58.000 ROR A
59.000 BCC LOOP5 ♦XMIT BUFFER EMPTY?
60.000 LDA A #$55 ♦OUTPUT SECOND CONTROL CHAR
61.000 STA A ACIAD
62.000 LOOP6 LDA A ACIAC
63.000 ROR A
64.000 ROR A
65.000 BCC LOOP6 ♦XMIT BUFFER EMPTY?
66.000 LOOP4 LDA A 0,X
67.000 STA A ACIAD ♦OUTPUT CHAR TO TAPE
68.000 LOOP3 LDA A ACIAC
69.000 ROR A
70.000 ROR A
71.000 BCC LOOP3 ♦XMIT BUFFER EMPTY?
72.000 INX
73.000 CPX $02
74.000 BNE LOOP4
75.000 BRA END
76.000 MON
```


Assembled Program for Load/Dump via ACIA Program

```

00010          NAM      LDBOOT
00020          OPT      M
00030          ♦ THIS PROGRAM LOADS OR DUMPS MEMORY
00040          ♦ PLACE START ADDRESS IN LOC 00 & 01
00050          ♦ PLACE END ADDRESS + 1 IN LOC 02 & 03
00060          ♦ IF ERROR OCCURS, CHECK LOC 04 & 05 FOR ADDRESS.
00070          ♦ CA2 STOPS DRIVE AT EOT OR ERROR.
00080          ♦ CB2 GIVES ERROR INDICATION.
00090          ♦ DUMP PROGRAM STARTS AT LOC 094B.
00100      0805  PIA1AC EQU    $0805
00110      0807  PIA1BC EQU    $0807
00120      0806  ACIAC  EQU    $0806
00130      0809  ACIAD  EQU    $0809
00140  0900          ORG    $0900
00150  0900  86 03          LDA  A    #$03
00160  0902  B7 0806       STA  A    ACIAC    ACIA MASTER RESET
00170  0905  DE 00          LDX  $00    LOAD START ADDRESS
00180  0907  86 19          LDA  A    #$19    ACIA 8 BITS EVEN PARITY
00190  0909  B7 0806       STA  A    ACIAC
00200  090C  B6 0806  LOOP  LDA  A    ACIAC
00210  090F  46           ROR  A
00220  0910  24 FA          BCC   LOOP    RECEIVER FULL?
00230  0912  B6 0809       LDA  A    ACIAD
00240  0915  81 AA  CMPAA  CMP  A    #$AA    IS FIRST CHAR "AA"?
00250  0917  26 F3          BNE   LOOP    BRANCH IF NOT
00260  0919  B6 0806  LOOP1 LDA  A    ACIAC
00270  091C  46           ROR  A
00280  091D  24 FA          BCC   LOOP1
00290  091F  B6 0809       LDA  A    ACIAD
00300  0922  81 55          CMP  A    #$55    IS SECOND CHAR "55"?
00310  0924  26 EF          BNE   CMPAA    IF NOT, TRY FOR AN "AA"
00320  0926  B6 0806  LOOP2 LDA  A    ACIAC
00330  0929  16           TAB
00340  092A  C4 70          AND  B    #$70
00350  092C  26 14          BNE   ERROR    BRANCH IF ERROR
00360  092E  46           ROR  A
00370  092F  24 F5          BCC   LOOP2
00380  0931  B6 0809       LDA  A    ACIAD    LOAD A CHAR FROM TAPE
00390  0934  A7 00          STA  0,X    STORE IN MEMORY
00400  0936  08           INX
00410  0937  9C 02          CPX  $02    INCREMENT ADDRESS
00420  0939  26 EB          BNE   LOOP2    LOAD COMPLETED?
00430  093B  86 30          LDA  A    #$30    GO GET MORE
00440  093D  B7 0805       STA  A    PIA1AC    TURN OFF CA2
00450  0940  20 FE          BRA  ♦
00460  0942  86 36  ERROR  LDA  A    #$36
00470  0944  B7 0807       STA  A    PIA1BC    TURN ON ERROR LIGHT
00480  0947  DF 04          STX  $04    STORE ADR OF ERROR
00490  0949  20 F0          BRA  END

```

Assembled Program for Load/Dump via ACIA Program (continued)

```

□
00510 094B DE 00          LDX    $00          ♦START OF DUMP PROGRAM
00520 094D 86 19          LDA    A    #$19
00530 094F B7 0806        STA    A    ACIAC
00540 0952 86 AA          LDA    A    #$AA    ♦OUTPUT CONTROL CHAR
00550 0954 B7 0809        STA    A    ACIAC
00560 0957 B6 0806 LOOP5  LDA    A    ACIAC
00570 095A 46             ROR    A
00580 095B 46             ROR    A
00590 095C 24 F9          BCC    LOOP5        ♦XMIT BUFFER EMPTY?
00600 095E 86 55          LDA    A    #$55    ♦OUTPUT SECOND CONTROL CHAR
00610 0960 B7 0809        STA    A    ACIAC
00620 0963 B6 0806 LOOP6  LDA    A    ACIAC
00630 0966 46             ROR    A
00640 0967 46             ROR    A
00650 0968 24 F9          BCC    LOOP6        ♦XMIT BUFFER EMPTY?
00660 096A A6 00 LOOP4    LDA    A    0,X
00670 096C B7 0809        STA    A    ACIAC    ♦OUTPUT CHAR TO TAPE
00680 096F B6 0806 LOOP3  LDA    A    ACIAC
00690 0972 46             ROR    A
00700 0973 46             ROR    A
00710 0974 24 F9          BCC    LOOP3        ♦XMIT BUFFER EMPTY?
00720 0976 08             INX
00730 0977 9C 02          CPX    $02
00740 0979 26 EF          BNE    LOOP4
00750 097B 20 BE          BRA    END
00760                      MON

```

System Configuration

SYSTEM CONFIGURATION

Perhaps the strongest points of the M6800 system is the extreme ease of use and connection given to the user and the almost complete lack of external parts needed for operation. This section is intended as an aid to connecting the various lines in order to have a complete operating M6800 system. In general, three sets of lines need to be connected: the data bus, the address bus, and the control bus. It is assumed that when this step in the design cycle is reached, the parts required for the particular system are known. An example follows this section.

DATA BUS

The data bus connection is trivial; the eight bus lines D0 through D7 simply connect to those pins on every package marked D0 through D7.

ADDRESS BUS

The sole purpose of the address bus wiring is to give each memory location in every part its own *unique* address. This is accomplished by connecting the various chip selects on each part to the address bus in order to select that part from the others, and then wiring the lower order address lines to select a particular location within each part. This can be thought of as selecting an individual page, and then selecting one word on that page, thus accessing only one word in the whole book!

Usually RAM is located in lowest memory to take advantage of the direct memory addressing mode. ROM is in high memory because we must access the eight highest memory locations in order to have the four fixed vectors (IRQ, SWI, NMI, RESTART) available to the MPU. PIAs and ACIAs are usually located in middle memory.

The rules for accomplishing this task are as follows.

1. Connect the lower address lines that will enable a particular location within each part. These are RS0 and RS1 for PIAs, Rs for ACIAs, A0 through A6 for RAMs, and A0 through A9 for ROMs.

2. Configure the address lines to the various chip selects to select a particular part. This is done by:
 - a. Connecting address lines to select a device *type*; that is when ROM is addressed, RAM, PIA, and ACIA are disabled; when RAM is addressed, ROM, PIA, and ACIA are disabled; when PIA/ACIA is address, ROM and RAM are disabled.
 - b. Next, use other address lines to select one part within each type; one ROM out of all the ROMs, one RAM out of all RAMs, etc.

If we now check that each part has its own unique addresses, and the MPU can access the four vectors in upper ROM, then the addressing task is complete.

To recap what has been done: We used upper address lines to select a device type; other lines selected one individual part in that type; and lastly the lower address lines picked out one memory location within that part. Unused chip selects simply tie to the appropriate level, +5 V or ground.

CONTROL BUS

The control lines are necessary for timing and control of the system. These lines are \overline{IRQ} , \overline{RES} , $\phi 2$, R/W, and VMA.

The \overline{IRQ} lines from ACIA or PIA may be tied together (wire-OR'ed) and run to either \overline{IRQ} or \overline{NMI} on the processor. They should be pulled up to +5 V through a single 3-10 k resistor at the MPU for optimum results.

The RES signal is applied by external circuitry to the MPU and should also be connected to the PIA RES pin.

$\phi 2$ is used as a sync signal to any part in the system that acts as RAM, that is, the MPU can write into it. It should be wired to the E pin of the PIA, ACIA and a chip select of the RAM. This $\phi 2$ signal need not be the same $\phi 2$ that goes to the MPU, for this is a non-TTL-compatible clock signal. Rather, a TTL-type $\phi 2$ should be bussed around the system.

The R/W line simply connects to the R/W pin on the RAM, PIA, and ACIA. In some cases, it may be pulled up to +5 V through a 3-10 k resistor, as discussed later.

VMA is a signal that, when high, indicates a valid address is being applied to the bus by the MPU. There are two general times when the address on the bus is invalid.

1. During some internal operations, the MPU allows invalid addresses to appear on the bus, although they are unused.
2. Anytime the bus is three-stated, it is floating and therefore invalid.

We need to be sure that VMA is used to prevent destruction of data in our system by writing into a location wrongly, or in the case of a PIA or ACIA, by reading a register and accidentally erasing some pending interrupt flags.

During condition 1, above, the MPU hold R/W in the read state so that nothing can be written. We have only to protect the PIAs and ACIAs against flag erasure by connecting VMA ANDed with an address line to a chip select as will be shown in the following example.

During the three-state modes (HALT, WAI, etc.) VMA goes low and we have to make sure that nothing gets written into a memory location by accident. This can be done with a pullup resistor on R/W to insure a read state, or in some systems VMA may be connected to a chip select on the RAMs, if available. Either way, protection is adequate.

NOTE: VMA may be connected to ROM if desired. This is not absolutely necessary as data in ROM cannot be destroyed, but no harm will come in doing so.

Other special connections for systems involving DMA or dynamic memory cycle-stealing refresh are discussed in the M6800 applications manual, section 4.

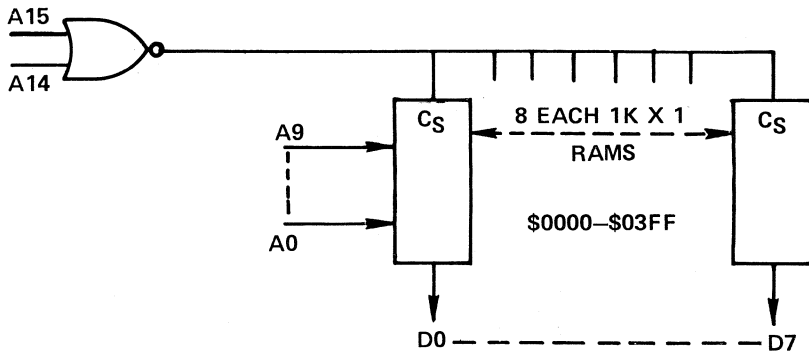
BUS BUFFERING

The M6800 system is guaranteed to run worst case at a maximum clock rate of two megahertz with ten family devices on the busses. If more loading than this is necessary, then bus buffers are required. Section 4-19 of the Applications Manual.

Other miscellaneous lines may need to be connected depending on the system complexity. These include TSC, HALT, and BA. Often these are simply tied to logic levels, but their further use is discussed in the applications manual, section 4. DBE is normally tied to $\phi 2$, which is the pin adjacent to it on the MPU package.

MEMORY EXPANSION

Occasionally the user needs larger amounts of RAM than the MCM6810. In these cases, expansion is extremely simple. If the user needs 1K x 8, for example, probably the cheapest way is to use eight 1K x 1 RAMs in parallel. In this case, A0 through A9 are common to all eight packages, and their enable is shown below. 4K x 8 of RAM is identical except that lines A0 through A11 are used.



The student should notice that a great many M6800 system parts may be attached to busses with no external address decoding whatsoever. The only external gate required other than clocks is that for VMA·A_{XX} to the PIAs. With only standard parts, large amounts of I/O and memory are available to the user. The great ease of configuring an M6800 system is shown in the following example.

System Configuration Example

Suppose our problem is to configure a system that contains:

3 RAMs
3 ROMs
3 PIAs
1 ACIA

Using the system layout sheet, we list the devices in the left-hand column. Next, we put an 'X' on each address line connection for each part, i.e., A0 through A6 for RAMs, A0 through A9 for ROMs; A0, A1 to RS0, RS1 for PIA; and, A0 to RS for the ACIA. See Figure 2.

We must now select a device *type* to the exclusion of the others. Remembering that ROM must be in high memory, we place a CS on A15. Then a \overline{CS} for RAMs, PIAs, and ACIAs isolates the ROM device type. Now whenever A15 is high, only ROM is accessed. See Figure 3.

NOTE: PIAs and ACIAs are considered to be the same device type as they have the same chip select configuration and are therefore treated alike in addressing.

To separate RAM from PIA/ACIA, we use A14 as shown in Figure 4. It can now be seen that with a combination of A15 and A14, each device type can be called out while excluding the others.

Now, with each device type selected, we may select one individual part within each type. Starting with the RAMs, we see that we will need two address lines to distinguish among the three. Using A7 and A8 as shown in Figure 5, we can now select one and only one RAM with A7, A8, A14, and A15.

The same thing can be done with ROMs in Figure 6. Starting with all CS (no \overline{CS}) on ROM#3, we work backward through the binary sequence on A10 and A11.

Now the PIAs and the ACIA have only one CS left for use. The standard most often employed is shown in Figure 7.

We may now check that each device has its own unique address assignments by determining the location of each part as done in the right-hand columns of Figure 7. These numbers will be used in the software to address each part. (Unused address lines are assumed low, 0.) We can see that some devices have several addresses that can be called to enable them. This is no problem, since we have control over those addresses called by our program and we choose to use the unique ones. For example, address \$401C will enable all three PIAs, but we choose to use \$4004, \$4008, and \$4010.

The last thing to check is that the MPU can get at the interrupt and restart vectors. We apply \$FFF8 through \$FFFF to our address map and notice that the upper eight locations in ROM#3 are enabled. This is true even though we are using addresses \$8C00 through \$8FFF to denote this ROM. Here is an example of non-fully-decoded addressing, i.e., address \$8FFF, \$9FFF, \$AFFF through \$FFFF all appear to be the same. Using more chip selects to more fully decode this area would eliminate this effect if we needed to, but our system does not.

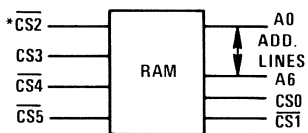
The control lines are connected as shown in Figure 8. Notice that R/W is pulled up to +5 V. The pullup resistor could be eliminated in this system by applying VMA to a CS on each RAM, although either approach is acceptable to avoid alteration of data.

Notice our system has two unused address lines, A12 and A13. These do not have to be bussed all over our system. Figure 9 shows the same system with three unused address lines. The system is identical except that some of the parts are at different unique addresses.

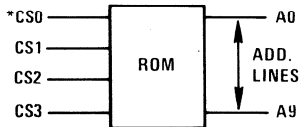
SYSTEM CONNECT

MUST CONNECT

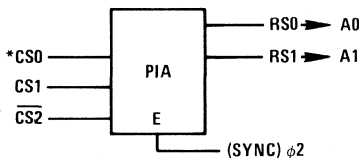
1. DATA BUS
2. INTERNAL ADDRESSES
3. ADDRESSES TO ENABLE EACH PART
 - A. GIVE EACH PART ITS OWN UNIQUE ADDRESSES
 - B. MAKE SURE ALL UPPER-ROM VECTORS ARE AVAILABLE
ACCOMPLISH THIS BY USING TWO GROUPS OF ADDRESS LINES TO ENABLES (OR CS):
 - i. USE ONE GROUP TO SELECT A DEVICE TYPE, I.E., RAMS, ROMS, PIA/ACIA
 - ii. USE SECOND GROUP TO SELECT OUT ONE INDIVIDUAL PART IN EACH TYPE
4. CONTROL BUS
($\phi 2$, VMA R/W)



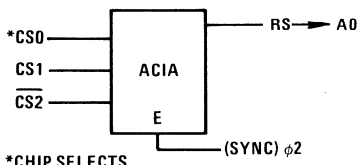
RAM: 80₁₆ BYTES OF MEMORY EACH. 2 CS, 4 $\overline{\text{CS}}$, ONE CS MUST BE TIED TO $\phi 2$. USUALLY LOCATED IN LOWER MEMORY.



ROM: 400₁₆ BYTES OF MEMORY EACH. 4 CS, CUSTOMER DEFINES WHICH CS AND $\overline{\text{CS}}$. USUALLY LOCATED IN UPPER MEMORY.



PIA: 4 BYTES OF MEMORY EACH. 2 CS, 1 $\overline{\text{CS}}$. ENABLE (SYNC) TIED TO $\phi 2$. USUALLY RS0 TIED ADD. LINE A0, RS1 to A1. USUALLY LOCATED IN MIDDLE MEMORY.



ACIA: 2 BYTES OF MEMORY EACH. 2 CS, 1 $\overline{\text{CS}}$. ENABLE (SYNC) TIE TO $\phi 2$. USUALLY RS TIED TO A0. USUALLY LOCATED IN MIDDLE MEMORY.

TR1150

Figure 2 – SYSTEM LAYOUT WORK SHEET

Device	MPU Address Lines (A0 – A15)															Address		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1										X	X	X	X	X	X	X		
RAM #2										X	X	X	X	X	X	X		
RAM #3										X	X	X	X	X	X	X		
ROM #1							X	X	X	X	X	X	X	X	X	X		
ROM #2							X	X	X	X	X	X	X	X	X	X		
ROM #3							X	X	X	X	X	X	X	X	X	X		
PIA #1															RS1	RS0		
PIA #2															RS1	RS0		
PIA #3															RS1	RS0		
ACIA #1																RS		

Figure 3 – SYSTEM LAYOUT WORK SHEET

Device	MPU Address Lines (A0 – A15)															Address		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1	\overline{CS}									X	X	X	X	X	X	X		
RAM #2	\overline{CS}									X	X	X	X	X	X	X		
RAM #3	\overline{CS}									X	X	X	X	X	X	X		
ROM #1	CS						X	X	X	X	X	X	X	X	X	X		
ROM #2	CS						X	X	X	X	X	X	X	X	X	X		
ROM #3	CS						X	X	X	X	X	X	X	X	X	X		
PIA #1	\overline{CS}														RS1	RS0		
PIA #2	\overline{CS}														RS1	RS0		
PIA #3	\overline{CS}														RS1	RS0		
ACIA #1	\overline{CS}															RS		

Figure 4 – SYSTEM LAYOUT WORK SHEET

Device	MPU Address Lines (A0 – A15)															Address		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1	\overline{CS}	\overline{CS}								X	X	X	X	X	X	X		
RAM #2	\overline{CS}	\overline{CS}								X	X	X	X	X	X	X		
RAM #3	\overline{CS}	\overline{CS}								X	X	X	X	X	X	X		
ROM #1	CS						X	X	X	X	X	X	X	X	X	X		
ROM #2	CS						X	X	X	X	X	X	X	X	X	X		
ROM #3	CS						X	X	X	X	X	X	X	X	X	X		
PIA #1	\overline{CS}	CS													RS1	RS0		
PIA #2	\overline{CS}	CS													RS1	RS0		
PIA #3	\overline{CS}	CS													RS1	RS0		
ACIA #1	\overline{CS}	CS														RS		

Figure 5 – SYSTEM LAYOUT WORK SHEET

MPU Address Lines (A0–A15)																Address		
Device	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X		
RAM #2	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X		
RAM #3	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X		
ROM #1	CS						X	X	X	X	X	X	X	X	X	X		
ROM #2	CS						X	X	X	X	X	X	X	X	X	X		
ROM #3	CS						X	X	X	X	X	X	X	X	X	X		
PIA #1	$\overline{\text{CS}}$	CS													RS1	RS0		
PIA #2	$\overline{\text{CS}}$	CS													RS1	RS0		
PIA #3	$\overline{\text{CS}}$	CS													RS1	RS0		
ACIA #1	$\overline{\text{CS}}$	CS														RS		

Figure 6 – SYSTEM LAYOUT WORK SHEET

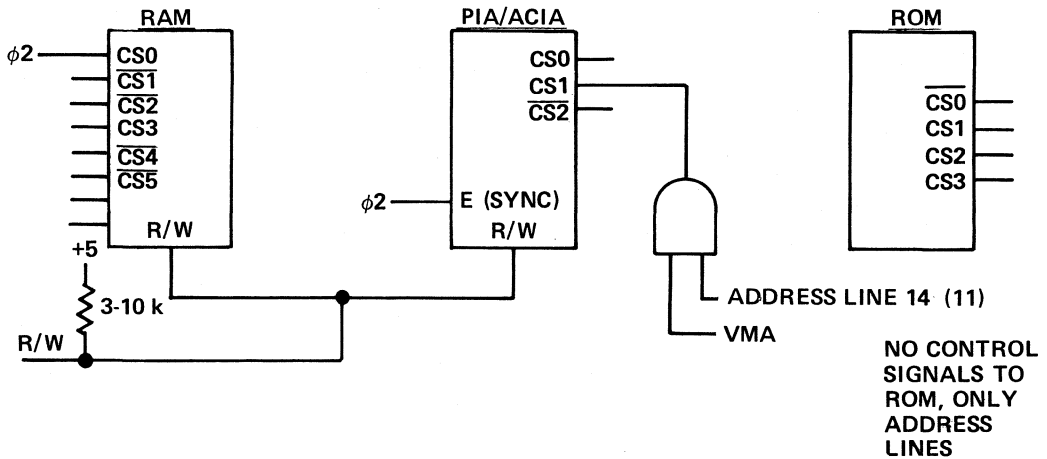
MPU Address Lines (A0–A15)																Address		
Device	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X		
RAM #2	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X		
RAM #3	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X		
ROM #1	CS				$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	X	X	X		
ROM #2	CS				$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	X	X	X		
ROM #3	CS				$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	X	X	X		
PIA #1	$\overline{\text{CS}}$	CS													RS1	RS0		
PIA #2	$\overline{\text{CS}}$	CS													RS1	RS0		
PIA #3	$\overline{\text{CS}}$	CS													RS1	RS0		
ACIA #1	$\overline{\text{CS}}$	CS														RS		

Figure 7 – SYSTEM LAYOUT WORK SHEET

MPU Address Lines (A0–A15)																Address		
Device	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	0000	007F
RAM #2	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	0080	00FF
RAM #3	$\overline{\text{CS}}$	$\overline{\text{CS}}$						$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	0100	017F
ROM #1	CS				$\overline{\text{CS}}$	CS	X	X	X	X	X	X	X	X	X	X	8400	87FF
ROM #2	CS				$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	X	X	X	8800	8BFF
ROM #3	CS				$\overline{\text{CS}}$	$\overline{\text{CS}}$	X	X	X	X	X	X	X	X	X	X	8C00	8FFF
PIA #1	$\overline{\text{CS}}$	CS												CS	RS1	RS0	4004	4007
PIA #2	$\overline{\text{CS}}$	CS											CS		RS1	RS0	4008	400B
PIA #3	$\overline{\text{CS}}$	CS										CS			RS1	RS0	4010	4013
ACIA #1	$\overline{\text{CS}}$	CS								CS						RS	4020	4021

Unused inputs tied to ground or to +5 V, as necessary.

SYS-8 System Configuration



System 2		MPU Address Lines (A0 - A15)														Address			
Device		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	From	To
RAM #1	CS0, CS3, +5 V				$\overline{\text{CS5}}$	$\overline{\text{CS4}}$			$\overline{\text{CS2}}$	$\overline{\text{CS1}}$	X	X	X	X	X	X	X	0000	007F
RAM #2	CS3, +5 V; $\overline{\text{CS4}}$, gnd				$\overline{\text{CS5}}$	$\overline{\text{CS2}}$			$\overline{\text{CS1}}$	CS0	X	X	X	X	X	X	X	0080	00FF
RAM #3	CS3, +5 V; $\overline{\text{CS4}}$, gnd				$\overline{\text{CS5}}$	$\overline{\text{CS2}}$			CS0	$\overline{\text{CS1}}$	X	X	X	X	X	X	X	0100	017F
ROM #1	CS1, CS2, +5 V				CS3	$\overline{\text{CS0}}$	CS	X	X	X	X	X	X	X	X	X	X	1400	17FF
ROM #2	CS0, gnd; CS2, +5 V				CS3	CS1	$\overline{\text{CS}}$	X	X	X	X	X	X	X	X	X	X	1800	18FF
ROM #3	CS0, gnd; CS2, +5 V				CS3	CS1	CS	X	X	X	X	X	X	X	X	X	X	1C00	1FFF
PIA #1					$\overline{\text{CS2}}$	CS1									CS0	RS1	RS0	0804	0807
PIA #2					$\overline{\text{CS2}}$	CS1								CS0		RS1	RS0	0808	080B
PIA #3					$\overline{\text{CS2}}$	CS1							CS0			RS1	RS0	0810	0813
ACIA #1					$\overline{\text{CS2}}$	CS1						CS0					RS	0820	0821

