查询HD68000CP-8供应商 捷多邦，专业PCB打样工厂，24小时加急出货

HD68000/HD68HC000

MPU (Micro Processing Unit)

— HD68000 —

The HD68000 is the first in a family of advanced microprocessors from Hitachi. Utilizing VLSI technology, the HD68000 is a fully-implemented 16-bit microprocessor with 32-bit registers, a rich basic instruction set, and versatile addressing modes.

The HD68000 possesses an asynchronous bus structure with a 24-bit address bus and a 16-bit data bus.

FEATURES

- 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations of Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes

— HD68HC000 —

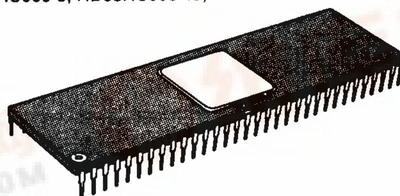
The HD68HC000 is a 16-bit microprocessor of HD68000 family, which is exactly compatible with the conventional HD68000.

The HD68HC000 is a complete CMOS device and the power dissipation is extremely low.

FEATURES

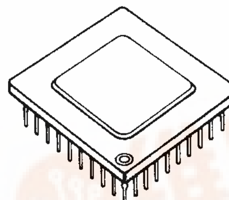
- Instruction Compatible with NMOS HD68000
- Pin Compatible with NMOS HD68000
- AC Timing Compatible with NMOS HD68000
- Low Power Dissipation (I_{CC} typ = 20 mA, I_{CC} max = 35 mA at f = 12.5 MHz)

HD68000-8, HD68000-10, HD68000-12
HD68HC000-8, HD68HC000-10, HD68HC000-12



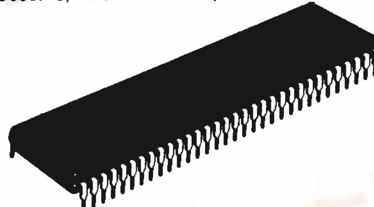
(DC-64)

HD68000Y-8, HD68000Y-10, HD68000Y-12
HD68HC000Y-8, HD68HC000Y-10, HD68HC000Y-12



(PGA-68)

HD68000P-8
HD68HC000P-8, HD68HC000P-10, HD68HC000P-12

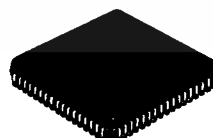


(DP-64)

HD68000PS-8
HD68HC000PS-8, HD68HC000PS-10, HD68HC000PS-12

(DP-64S)

HD68000CP-8
HD68HC000CP-8, HD68HC000CP-10, HD68HC000CP-12



(CP-68)

4

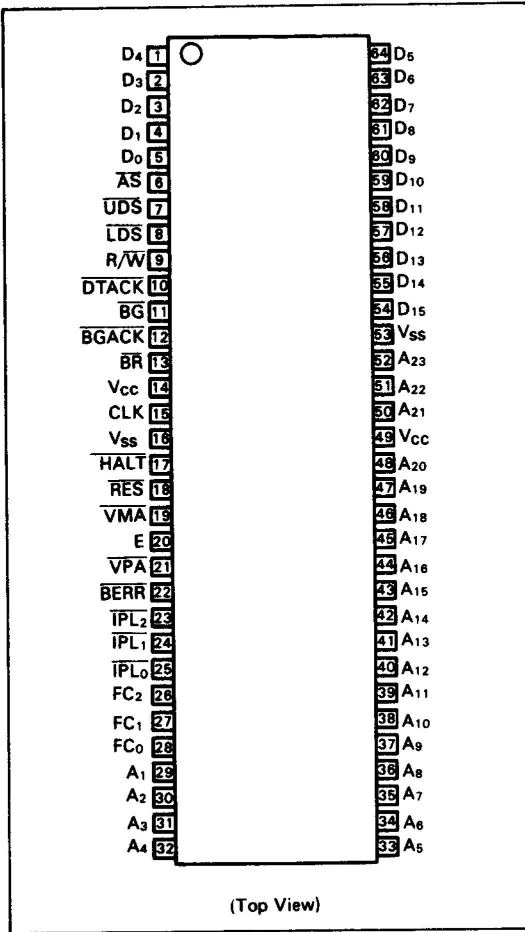
■ TYPE OF PRODUCTS

Type No.	Process	Clock Frequency (MHz)	Package
HD68000-8	NMOS	8.0	DC-64
HD68000-10		10.0	
HD68000-12		12.5	
HD68000Y-8		8.0	PGA-68
HD68000Y-10		10.0	
HD68000Y-12		12.5	
HD68000P-8		8.0	DP-64
HD68000PS8		8.0	DP-64S
HD68000CP8		8.0	CP-68
HD68HC000-8	CMOS	8.0	DC-64
HD68HC000-10		10.0	
HD68HC000-12		12.5	
HD68HC000Y-8		8.0	PGA-68
HD68HC000Y-10		10.0	
HD68HC000Y-12		12.5	
HD68HC000P-8		8.0	DP-64
HD68HC000P-10		10.0	
HD68HC000P-12		12.5	
HD68HC000PS8		8.0	DP-64S
HD68HC000PS10		10.0	
HD68HC000PS12		12.5	
HD68HC000CP8		8.0	CP-68
HD68HC000CP-10		10.0	
HD68HC000CP-12		12.5	

(Note) HD68000 refers to the NMOS version 68000, and HD68HC000 refers to the CMOS version 68000. 68000 stands for NMOS and CMOS version.

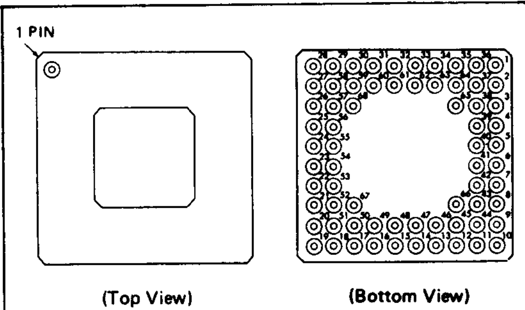
■ PIN ARRANGEMENT

● DC-64, DP-64, DP-64S



(Top View)

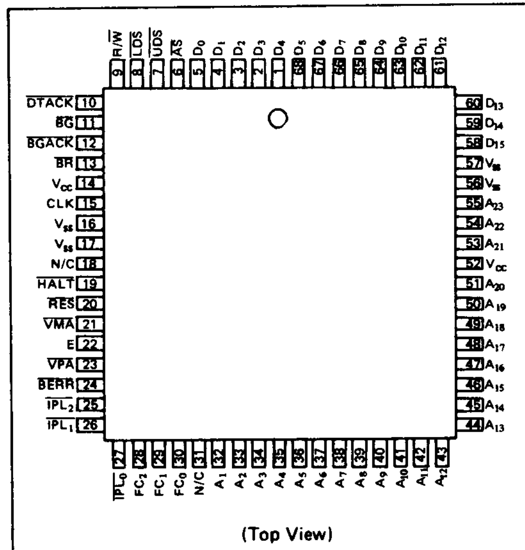
● PGA-68



(Top View) (Bottom View)

Pin No.	Function	Pin No.	Function	Pin No.	Function	Pin No.	Function
1	N/C	18	A_9	35	D_1	52	A_{12}
2	$\overline{DTACK}$	19	N/C	36	$\overline{AS}$	53	A_{15}
3	$\overline{BGACK}$	20	A_{14}	37	$\overline{LDS}$	54	A_{18}
4	$\overline{BR}$	21	A_{16}	38	$\overline{BG}$	55	V_{CC}
5	CLK	22	A_{17}	39	V_{CC}	56	V_{SS}
6	$\overline{HALT}$	23	A_{19}	40	V_{SS}	57	A_{23}
7	$\overline{VMA}$	24	A_{20}	41	$\overline{RES}$	58	D_{14}
8	E	25	A_{21}	42	$\overline{VPA}$	59	D_{11}
9	$\overline{BERR}$	26	A_{22}	43	$\overline{IPL_2}$	60	D_9
10	N/C	27	D_{15}	44	$\overline{IPL_0}$	61	D_6
11	FC_2	28	D_{12}	45	FC_1	62	D_3
12	FC_0	29	D_{10}	46	N/C	63	D_0
13	A_1	30	D_8	47	A_2	64	$\overline{UDS}$
14	A_3	31	D_7	48	A_5	65	$R/\overline{W}$
15	A_4	32	D_5	49	A_8	66	$\overline{IPL_1}$
16	A_6	33	D_4	50	A_{10}	67	A_{13}
17	A_7	34	D_2	51	A_{11}	68	D_{13}

● CP-68



(Top View)

4

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	HD68000	HD68HC000	Unit
		Value	Value	
Supply Voltage	V_{CC}*	−0.3 ~ +7.0	−0.3 ~ +6.5	V
Input Voltage	V_{in}*	−0.3 ~ +7.0	−0.3 ~ +6.5	V
Operating Temperature Range	T_{opr}	0 ~ +70	0 ~ +70	°C
Storage Temperature	T_{stg}	−55 ~ +150	−55 ~ +150	°C

*With respect to V_{SS} (SYSTEM GND)

(NOTE) Permanent LSI damage may occur if maximum ratings are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

Since the HD68HC000 is a C-MOS device, users are expected to be cautious on "latch-up" problem caused by voltage fracturations.

■ RECOMMENDED OPERATING CONDITIONS

Item		Symbol	HD68000			HD68HC000			Unit
			min	typ	max	min	typ	max	
Supply Voltage		V_{CC}*	4.75	5.0	5.25	4.75	5.0	5.25	V
Input Voltage	CLK	V_{IH}*	2.0	–	V_{CC}	2.8	–	V_{CC}	V
	Other Inputs					2.0	–	V_{CC}	
	All Inputs	V_{IL}*	−0.3	–	0.8	−0.3	–	0.8	V
Operating Temperature		T_{opr}	0	25	70	0	25	70	°C

* With respect to V_{SS} (SYSTEM GND)

■ ELECTRICAL CHARACTERISTICS

● DC CHARACTERISTICS (V_{CC} = 5V ± 5%, V_{SS} = 0V, Ta = 0 ~ +70°C, Fig. 1, unless otherwise noted.)

Item		Symbol	Test Condition	HD68000		HD68HC000		Unit
				min	max	min	max	
Input "High" Voltage	CLK	V_{IH}		2.0	V_{CC}	2.8	V_{CC}	V
	Other Inputs					2.0	V_{CC}	
Input "Low" Voltage		V_{IL}		V_{SS}-0.3	0.8	V_{SS}-0.3	0.8	V
Input Leakage Current	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL_0}$ ~ $\overline{IPL_2}$, $\overline{VPA}$, CLK	I_{in}	@ 5.25V	–	2.5	–	2.5	μA
	$\overline{HALT}$, $\overline{RES}$			–	20	–	20	
Three-State (Off State) Input Current	$\overline{AS}$, A_1 ~ A_{23}, D_0 ~ D_{15}, FC_0 ~ FC_2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	I_{TSI}	@ 2.4V/0.4V	–	20	–	20	μA
Output "High" Voltage	$\overline{AS}$, A_1 ~ A_{23}, $\overline{BG}$, D_0 ~ D_{15}, FC_0 ~ FC_2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$, E	V_{OH}	I_{OH} = -400μA	2.4	–	V_{CC}-0.75	–	V
	E*			V_{CC}-0.75	–			
Output "Low" Voltage	$\overline{HALT}$	V_{OL}	I_{OL}=1.6 mA	–	0.5	–	0.5	V
	A_1~A_{23}, $\overline{BG}$, FC_0 ~ FC_2		I_{OL}=3.2 mA	–	0.5	–	0.5	
	$\overline{RES}$		I_{OL}=5.0 mA	–	0.5	–	0.5	
	$\overline{AS}$, D_0 ~ D_{15}, $\overline{LDS}$, R/$\overline{W}$, E, $\overline{UDS}$, $\overline{VMA}$		I_{OL}=5.3 mA	–	0.5	–	0.5	
Power Dissipation	CERAMIC PACKAGE	P_D	f = 6 MHz f = 8 MHz f = 10 MHz	–	1.5	–	–	W
			f = 12.5 MHz	–	1.75			
	PLASTIC PACKAGE		f = 8 MHz, V_{CC} = 5V, Ta = 25°C	–	0.9			
Current Dissipation		I_D**	f = 8 MHz	–	–	–	25	mA
			f = 10 MHz	–	–	–	30	
			f = 12.5 MHz	–	–	–	35	
Capacitance (Package Type Dependent)		C_{in}	V_{in} = 0V, Ta = 25°C, f = 1 MHz	–	20.0	–	20.0	pF

*With external pull up resistor of 1.1 kΩ.
**Without load.

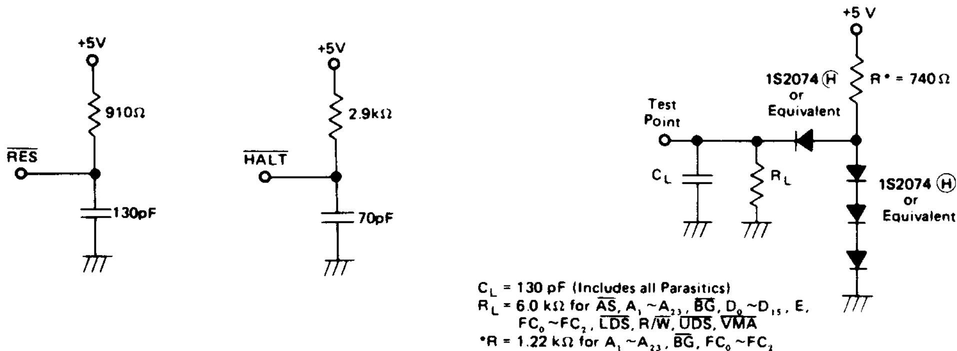
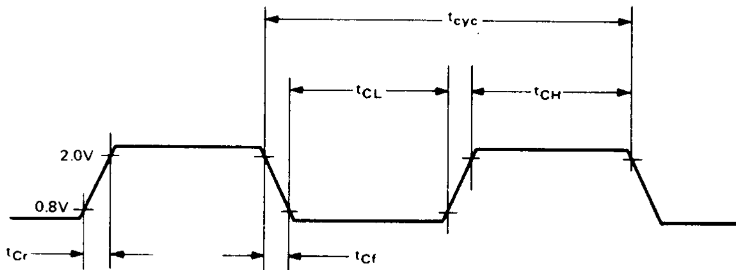


Figure 1 Test Loads

- **AC CHARACTERISTICS** (V_{CC} = 5V ± 5%, V_{SS} = 0V, Ta = 0 ~ +70°C, unless otherwise noted.)

CLOCK TIMING

Item		Symbol	Test Condition	8 MHz		10 MHz		12.5 MHz		Unit
				min	max	min	max	min	max	
Frequency of Operation		f	Fig. 2	4.0	8.0	4.0	10.0	4.0	12.5	MHz
Cycle Time		t_{cyc}		125	250	100	250	80	250	ns
Clock Pulse Width		t_{CL}		55	125	45	125	35	125	ns
		t_{CH}		55	125	45	125	35	125	ns
Rise and Fall Times		t_{Cr}		–	10	–	10	–	5	ns
		t_{Cf}		–	10	–	10	–	5	ns



(NOTE)

Timing measurements are referenced to and from a low voltage of 0.8 volt and high a voltage of 2.0 volts, unless otherwise noted.
The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 volt and 2.0 volts.

Figure 2 Clock Input Timing

READ AND WRITE CYCLES

Num.	Item	Symbol	Test Condition	8 MHz		10 MHz		12.5 MHz		Unit
				min	max	min	max	min	max	
1	Clock Period	t_{cyc}		125	250	100	250	80	250	ns
2	Clock Width Low	t_{CL}		55	125	45	125	35	125	ns
3	Clock Width High	t_{CH}		55	125	45	125	35	125	ns
4	Clock Fall Time	t_{Cf}		–	10	–	10	–	5	ns
5	Clock Rise Time	t_{Cr}		–	10	–	10	–	5	ns
6	Clock Low to Address Valid	t_{CLAV}		–	70	–	60	–	55*	ns
6A	Clock High to FC Valid	t_{CHFCV}		–	70	–	60	–	55	ns
7	Clock High to Address, Data Bus High Impedance (Maximum)	t_{CHADZ}		–	80	–	70	–	60	ns
8	Clock High to Address, FC Invalid (Minimum)	t_{CHAFI}		0	–	0	–	0	–	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low	t_{CHSL}		0	60	0	55	0	55	ns
11[2]	Address Valid to $\overline{AS}$, $\overline{DS}$ Low (Read)/ $\overline{AS}$ Low (Write)	t_{AVSL}		30	–	20	–	0	–	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$ Low (Read)/ $\overline{AS}$ Low (Write)	t_{FCVSL}		60	–	50	–	40	–	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	Fig. 3,	–	70	–	55	–	50	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	t_{SHAFI}	Fig. 4	30	–	20	–	10	–	ns
14[2]	$\overline{AS}$, $\overline{DS}$ Width Low (Read)/$\overline{AS}$ Low (Write)	t_{SL}		240	–	195	–	160	–	ns
14A[2]	$\overline{DS}$ Width Low (Write)	t_{DSL}		115	–	95	–	80	–	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}		150	–	105	–	65	–	ns
16	Clock High to Control Bus High Impedance	t_{CHCZ}		–	80	–	70	–	60	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High (Read)	t_{SHRH}		40	–	20	–	10	–	ns
18[1]	Clock High to R/$\overline{W}$ High	t_{CHRH}		0	70	0	60	0	60	ns
20[1]	Clock High to R/$\overline{W}$ Low (Write)	t_{CHRL}		–	70	–	60	–	60	ns
20A[6]	$\overline{AS}$ Low to R/$\overline{W}$ Valid (Write)	t_{ASRV}		–	20	–	20	–	20	ns
21[2]	Address Valid to R/$\overline{W}$ Low (Write)	t_{AVRL}		20	–	0	–	0	–	ns
21A[2]	FC Valid to R/$\overline{W}$ Low (Write)	t_{FCVRL}		60	–	50	–	30	–	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	t_{RLSL}		80	–	50	–	30	–	ns
23	Clock Low to Data Out Valid (Write)	t_{CLDO}		–	70	–	55	–	55	ns
25[2]	$\overline{AS}$, $\overline{DS}$ High to Data Out Invalid (Write)	t_{SHDOI}		30	–	20	–	15	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	t_{DOSL}		30	–	20	–	15	–	ns
27[5]	Data In to Clock Low (Setup Time on Read)	t_{DICL}		15	–	10	–	10	–	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}		0	245	0	190	0	150	ns
29	$\overline{AS}$, $\overline{DS}$ High to Data In Invalid (Hold Time on Read)	t_{SHDII}		0	–	0	–	0	–	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}		0	–	0	–	0	–	ns
31[2, 5]	$\overline{DTACK}$ Low to Data In (Setup Time)	t_{DALDI}		–	90	–	65	–	50	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	$t_{RHr,f}$		0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}		–	70	–	60	–	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}		–	70	–	60	–	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}		1.5	90 ns +3.5	1.5	80 ns +3.5	1.5	70 ns +3.5	Clk. Per.

* 57 for HD68HC000

4

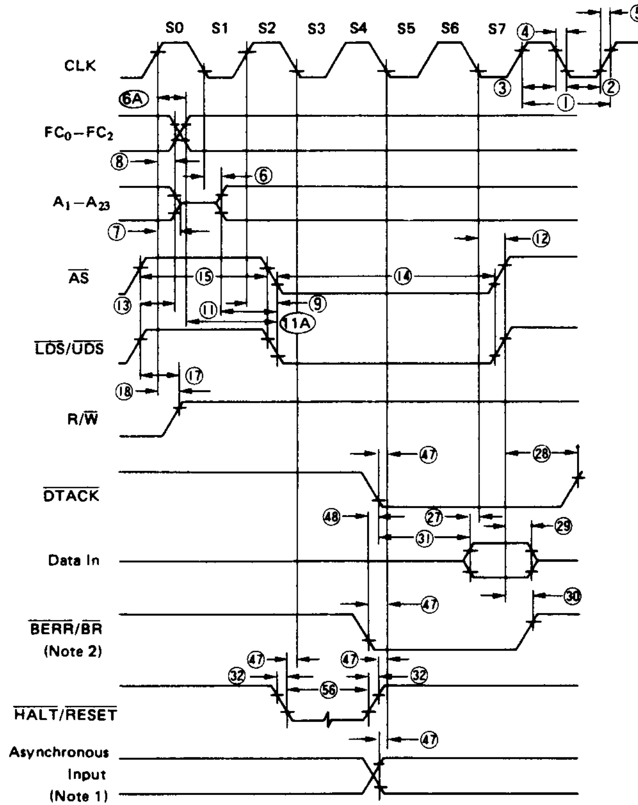
READ AND WRITE CYCLES (CONTINUED)

Num.	Item	Symbol	Test Condition	8 MHz		10 MHz		12.5 MHz		Unit
				min	max	min	max	min	max	
36[7]	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}		1.5	90ns +3.5	1.5	80ns +3.5	1.5	70ns +3.5	Clk.Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}		1.5	90ns +3.5	1.5	80ns +3.5	1.5	70ns +3.5	Clk.Per.
37A[8]	$\overline{BGACK}$ Low to $\overline{BR}$ High	t_{GALBRH}		20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	t_{GLZ}		–	80	–	70	–	60	ns
39	$\overline{BG}$ Width High	t_{GH}		1.5	–	1.5	–	1.5	–	Clk.Per.
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}		–	70	–	70	–	70	ns
41	Clock Low to E Transition	t_{CLET}		–	70	–	55	–	45	ns
42	E Output Rise and Fall Time	$t_{Er,f}$		–	25	–	25	–	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}		200	–	150	–	90	–	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}	Fig. 3, Fig. 4	0	120	0	90	0	70	ns
45	E Low to Control, Address Bus Invalid (Address Hold Time)	t_{ELCAI}		30	–	10	–	10	–	ns
46	$\overline{BGACK}$ Width Low	t_{GAL}		1.5	–	1.5	–	1.5	–	Clk.Per.
47[5]	Asynchronous Input Setup Time	t_{ASI}		20	–	20	–	20	–	ns
48[3]	$\overline{BERR}$ Low to $\overline{DTACK}$ Low	t_{BELDAL}		20	–	20	–	20	–	ns
49[9]	$\overline{AS}$, $\overline{DS}$ High to E Low	t_{SHEL}		–70	70	–55	55	–45	45	ns
50	E Width High	t_{EH}		450	–	350	–	280	–	ns
51	E Width Low	t_{EL}		700	–	550	–	440	–	ns
53	Clock High to Data Out Invalid	t_{CHDOI}		0	–	0	–	0	–	ns
54	E Low to Data Out Invalid	t_{ELDOI}		30	–	20	–	15	–	ns
55	R/$\overline{W}$ to Data Bus Driven	t_{RLDBD}		30	–	20	–	10	–	ns
56[4]	$\overline{HALT}$/$\overline{RESET}$ Pulse Width	t_{HRPW}		10	–	10	–	10	–	Clk.Per.
57	$\overline{BGACK}$ High to Control Bus Driven	t_{GABD}		1.5	–	1.5	–	1.5	–	Clk.Per.
58[7]	$\overline{BG}$ High to Control Bus Driven	t_{GHBD}		1.5	–	1.5	–	1.5	–	Clk.Per.

NOTES:

1. For a loading capacitance of less than or equal to 50 picofarads, substract 5 nanoseconds from the value given in the maximum columns.
2. Actual value depends on clock period.
3. If #47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, #48 may be 0 nanoseconds.
4. For power up, the MPU must be held in $\overline{RES}$ state for 100 ms to allow stabilization of on-chip circuitry. After the system is powered up, #56 refers to the minimum pulse width required to reset the system.
5. If the asynchronous setup time (#47) requirements are satisfied, the $\overline{DTACK}$ low-to-data setup time (#31) requirement can be ignored. The data must only satisfy the data-in clock-low setup time (# 27) for the following cycle.
6. When $\overline{AS}$ and R/$\overline{W}$ are equally loaded (±20%), subtract 10 nanoseconds from the values given in these columns.
7. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
8. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reasserted.
9. The falling edge of S6 triggers both the negation of the strobes ($\overline{AS}$ and x$\overline{DS}$) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.

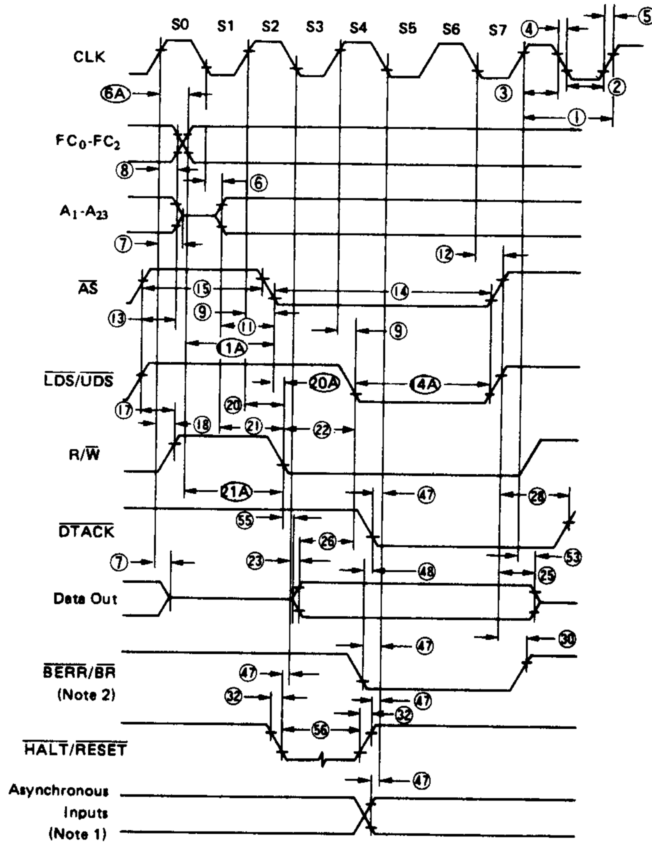
These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



NOTES:
1. Setup time for the synchronous inputs BGACK, $\overline{IPL}_{0-2}$ and VPA guarantees their recognition at the next falling edge of the clock.
2. BR need fall at this time only in order to insure being recognized at the end of this bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 volt and 2.0 volts.

Figure 3. Read Cycle Timing

These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.



NOTES:

1. Timing measurements are referenced to and from a low voltage of 0.8 volt and a high voltage of 2.0 volts, unless otherwise noted. The voltage swing through this range should start outside and pass through the range such that the rise or fall will be linear between 0.8 volt and 2.0 volts.
2. Because of loading variations, R/W may be valid after AS even though both are initiated by the rising edge of S2 (Specification 20A).

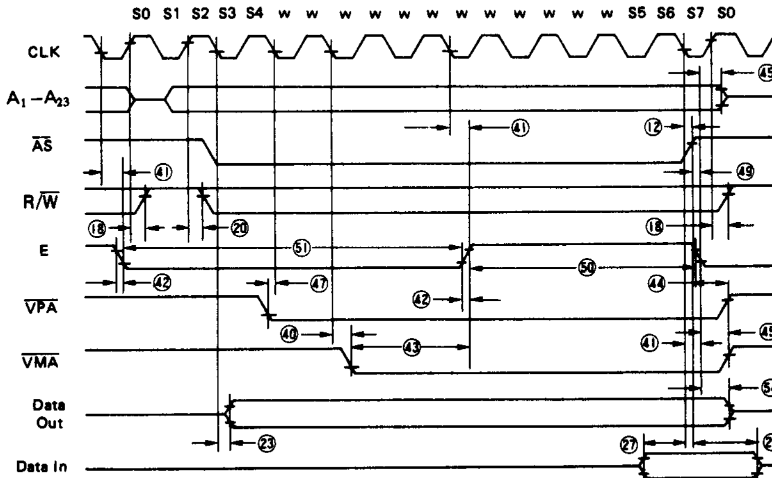
Figure 4. Write Cycle Timing

- **HMCS6800 TIMING**

Num.	Item	Symbol	Test Condition	6 MHz		8 MHz		10 MHz		12.5 MHz		Unit
				min	max	min	max	min	max	min	max	
12	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}		–	80	–	70	–	55	–	50	ns
18	Clock High to R/$\overline{W}$ High	t_{CHRH}		0	80	0	70	0	60	0	60	ns
20	Clock High to R/$\overline{W}$ Low (Write)	t_{CHRL}		–	80	–	70	–	60	–	60	ns
23	Clock Low to Data Out Valid (Write)	t_{CLDO}		–	80	–	70	–	55	–	55	ns
27	Data In to Clock Low (Setup Time on Read)	t_{DICL}		25	–	15	–	10	–	10	–	ns
29	$\overline{AS}$, $\overline{DS}$ High to Data In Invalid (Hold Time on Read)	t_{SHDII}	Fig. 5,	0	–	0	–	0	–	0	–	ns
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	Fig. 6	–	80	–	70	–	70	–	70	ns
41	Clock Low to E Transition	t_{CLET}		–	35	–	70	–	55	–	45	ns
42	E Output Rise and Fall Time	$t_{Er,f}$		–	25	–	25	–	25	–	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}		240	–	200	–	150	–	90	–	ns
44	$\overline{AS}$, $\overline{DS}$ High to $\overline{VPA}$ High	t_{SHVPH}		0	160	0	120	0	90	0	70	ns
45	E Low to Control, Address Bus Invalid (Address Hold Time)	t_{ELCAI}		35	–	30	–	10	–	10	–	ns
47	Asynchronous Input Setup Time	t_{ASI}		25	–	20	–	20	–	20	–	ns
49[1]	$\overline{AS}$, $\overline{DS}$ High to E Low	t_{SHEL}		–80	–	–70	70	–55	55	–45	45	ns
50	E Width High	t_{EH}		600	–	450	–	350	–	280	–	ns
51	E Width Low	t_{EL}		900	–	700	–	550	–	440	–	ns
54	E Low to Data Out Invalid	t_{ELDOI}		40	–	30	–	20	–	15	–	ns

NOTE:

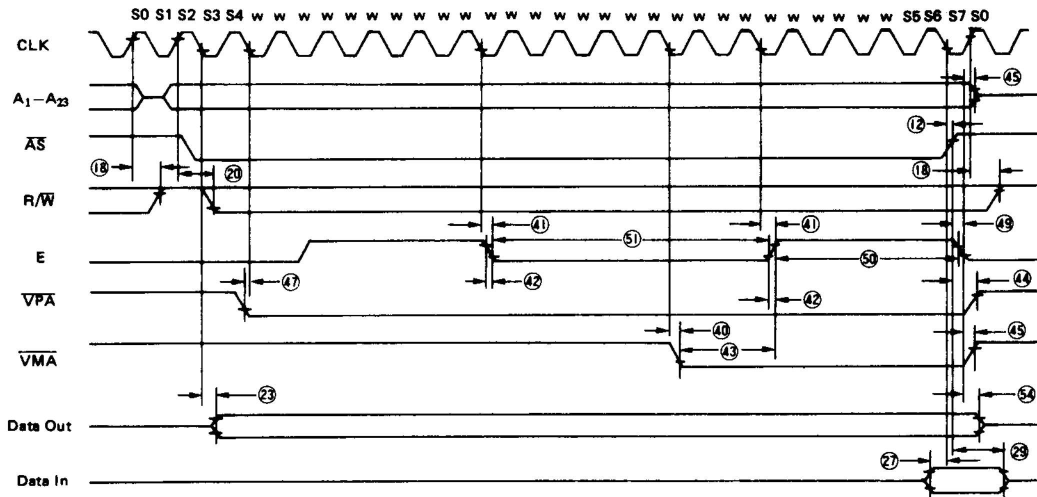
1. The falling edge of S6 triggers both the negation of the strobes ($\overline{AS}$ and x$\overline{DS}$) and the falling edge of E. Either of these events can occur first, depending upon the loading on each signal. Specification #49 indicates the absolute maximum skew that will occur between the rising edge of the strobes and the falling edge of the E clock.



NOTE: This timing diagram is included for those who wish to design their own circuit to generate $\overline{VMA}$. It shows the best case possibly attainable.

Figure 5. HD6800 Timing—Best Case

4



NOTE: This timing diagram is included for those who wish to design their own circuit to generate $\overline{VMA}$. It shows the worst case possibly attainable.

Figure 6. HD6800 Timing—Worst Case

BUS ARBITRATION

Num.	Item	Symbol	Test Condition	8 MHz min	8 MHz max	10 MHz min	10 MHz max	12.5 MHz min	12.5 MHz max	Unit
7	Clock High to Address, Data Bus High Impedance	t_{CHADZ}	Fig. 7 ~ Fig. 9	–	80	–	70	–	60	ns
16	Clock High to Control Bus High Impedance	t_{CHCZ}		–	80	–	70	–	60	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}		–	70	–	60	–	50	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}		–	70	–	60	–	50	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}		1.5	90ns +3.5	1.5	80ns +3.5	1.5	70ns +3.5	Clk.Per.
36[1]	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}		1.5	90ns +3.5	1.5	80ns +3.5	1.5	70ns +3.5	Clk.Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}		1.5	90ns +3.5	1.5	80ns +3.5	1.5	70ns +3.5	Clk.Per.
37A[2]	$\overline{BGACK}$ Low to $\overline{BR}$ High	t_{GALBRH}		20	1.5 Clocks	20	1.5 Clocks	20	1.5 Clocks	ns
38	$\overline{BG}$ Low to Control, Address, Data Bus High Impedance ($\overline{AS}$ High)	t_{GLZ}		–	80	–	70	–	60	ns
39	$\overline{BG}$ Width High	t_{GH}		1.5	–	1.5	–	1.5	–	Clk.Per.
46	$\overline{BGACK}$ Width Low	t_{GAL}		1.5	–	1.5	–	1.5	–	Clk.Per.
47	Asynchronous Input Setup Time	t_{ASI}		20	–	20	–	20	–	ns
57	$\overline{BGACK}$ High to Control Bus Driven	t_{GABD}		1.5	–	1.5	–	1.5	–	Clk.Per.
58[1]	$\overline{BG}$ High to Control Bus Driven	t_{GHBD}		1.5	–	1.5	–	1.5	–	Clk.Per.

NOTES:

1. The processor will negate $\overline{BG}$ and begin driving the bus again if external arbitration logic negates $\overline{BR}$ before asserting $\overline{BGACK}$.
2. The minimum value must be met to guarantee proper operation. If the maximum value is exceeded, $\overline{BG}$ may be reasserted.

Figures 7, 8, and 9 depict the three bus arbitration cases that can arise. Figure 7 shows the timing where $\overline{AS}$ is negated when the processor asserts $\overline{BG}$ (Idle Bus Case). Figure 8 shows the timing where $\overline{AS}$ is asserted when the processor asserts $\overline{BG}$ (Active Bus Case). Figure 9 shows the timing where more than one bus master are requesting the bus. Refer to Bus Arbitration for a complete discussion of bus arbitration.

The waveforms shown in Figures 7, 8, and 9 should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

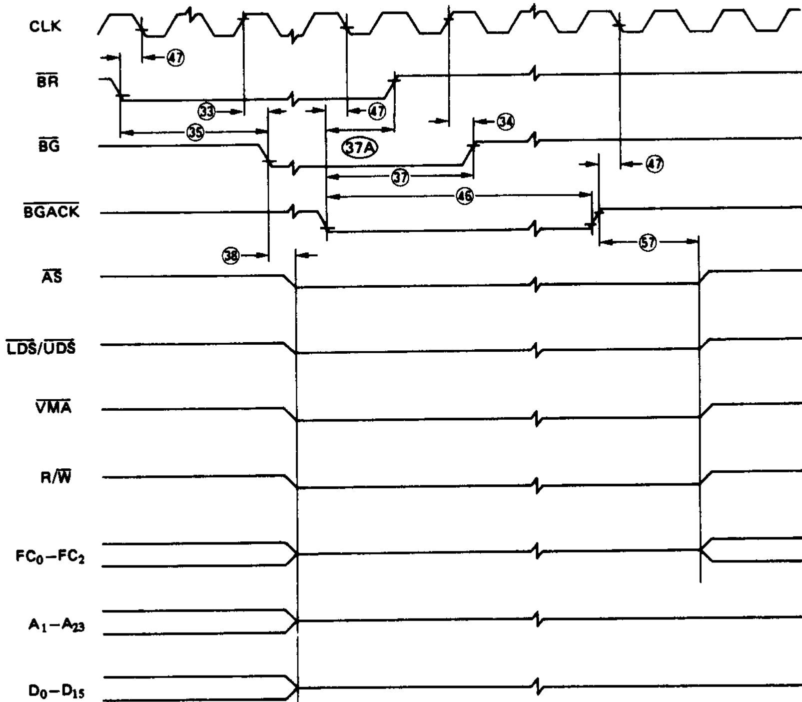


Figure 7. Bus Arbitration Timing Diagram — Idle Bus Case

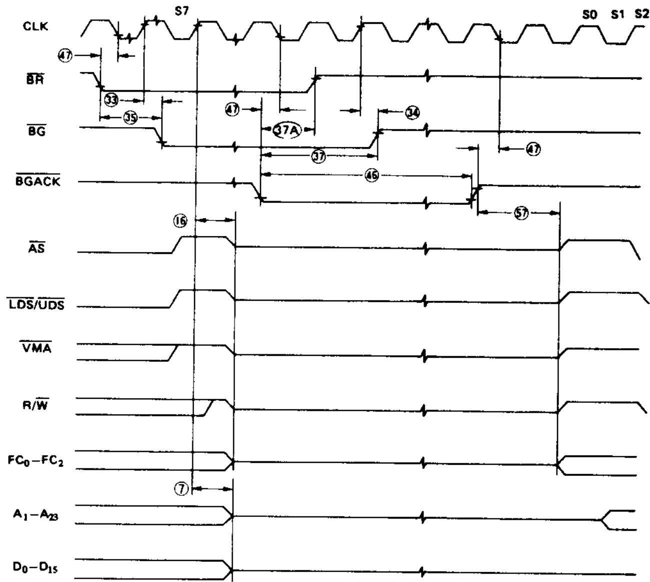


Figure 8. Bus Arbitration Timing Diagram — Active Bus Case

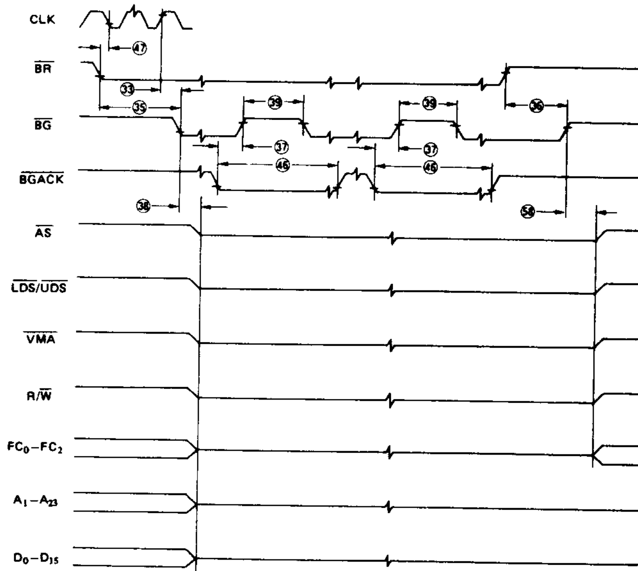


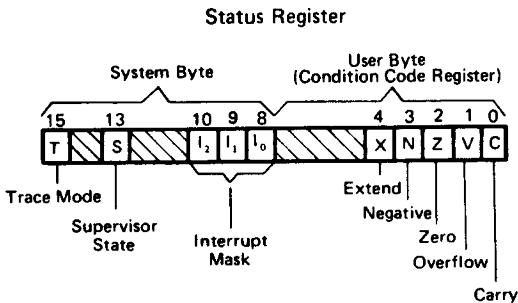
Figure 9. Bus Arbitration Timing Diagram — Multiple Bus Requests

■ INTRODUCTION

As shown in the programming model, the 68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0 ~ D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0 ~ A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All 17 registers may be used as index registers.

The status register contains the interrupt mask (eight levels available) as well as the condition codes; extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in a trace (T) mode and/or in a supervisor (S) state.

Status Register



Unused, read as zero.

● DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:
(1) Bits
(2) BCD Digits (4 bits)
(3) Bytes (8 bits)
(4) Word (16 bits)
(5) Long Words (32 bits)

In addition, operations on other data types such as memory address, status word data, etc., are provided for in the instruction set.

The 14 addressing modes, shown in Table 1, includes six basic types:
(1) Register Direct
(2) Register Indirect
(3) Absolute
(4) Immediate
(5) Program Counter Relative
(6) Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting and indexing. Program counter relative mode can also be modified via indexing and offsetting.

Programming Model

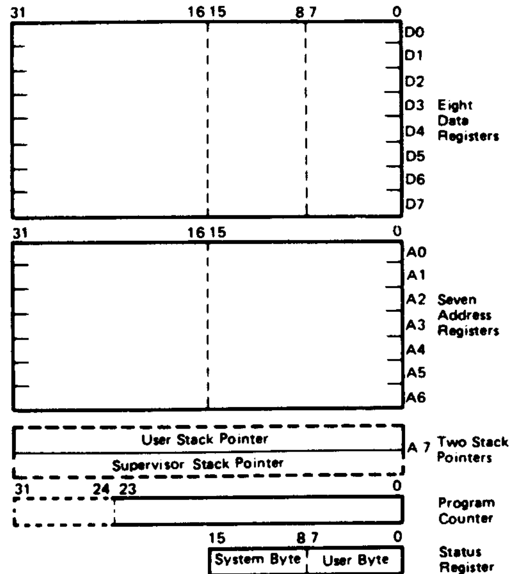


Table 1 Addressing Modes

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = (PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (AN), An ← An + N
Predecrement Register Indirect	An ← An − N, EA = (An)
Register Indirect with Offset	EA = (An) + d_{16}
Indexed Register Indirect with Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SP, PC

(NOTES)
EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
( ) = Contents of
d_8 = Eight-bit Offset (displacement)
d_{16} = Sixteen-bit Offset (displacement)
N = 1 for Byte, 2 for Words and 4 for Long Words. If An is the stack pointer and the operand size is byte, N=2 to keep the stack pointer on a word boundary.
← = Replaces

• INSTRUCTION SET OVERVIEW

The 68000 instruction set is shown in Table 2. Some additional instructions are variations, or subsets, of these and they appear in Table 3. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic and expanded operations (through traps).

Table 2 Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
B_{CC}	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Compare
DB_{CC}	Test Condition, Decrement and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide

Mnemonic	Description
EOR	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right
MOVE	Move
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NOP	No Operation
NOT	One's Complement
OR	Logical Or

Mnemonic	Description
PEA	Push Effective Address
RESET	Reset External Devices
ROL	Rotate Left without Extend
ROR	Rotate Right without Extend
ROXL	Rotate Left with Extend
ROXR	Rotate Right with Extend
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
S_{CC}	Set Conditional
STOP	Stop
SUB	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink

Table 3 Variations of Instruction Types

Instruction Type	Variation	Description
ADD	ADD	Add
	ADDA	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX	Add with Extend
AND	AND	Logical And
	ANDI	And Immediate
	ANDI to CCR	And Immediate to Condition Codes
	ANDI to SR	And Immediate to Status Register
CMP	CMP	Compare
	CMPA	Compare Address
	CMPM	Compare Memory
	CMPI	Compare Immediate
EOR	EOR	Exclusive Or
	EORI	Exclusive Or Immediate
	EORI to CCR	Exclusive Or Immediate to Condition Codes
	EORI to SR	Exclusive Or Immediate to Status Register

Instruction Type	Variation	Description
MOVE	MOVE	Move
	MOVEA	Move Address
	MOVEQ	Move Quick
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE to CCR	Move to Condition Codes
	MOVE USP	Move User Stack Pointer
NEG	NEG	Negate
	NEGX	Negate with Extend
OR	OR	Logical Or
	ORI	Or Immediate
	ORI to CCR	Or Immediate to Condition Codes
	ORI to SR	Or Immediate to Status Register
SUB	SUB	Subtract
	SUBA	Subtract Address
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX	Subtract with Extend

■ REGISTER DESCRIPTION AND DATA ORGANIZATION

The following paragraphs describe the registers and data organization of the 68000.

• OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Implict instructions support some subset of all three sizes.

• DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the active stack pointer support address operands of 32 bits.

DATA REGISTERS

Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

When a data register is used as either a source or destination operand, only the appropriate low-order portion is changed; the remaining high-order portion is neither used nor changed.

ADDRESS REGISTERS

Each address register and the stack pointer is 32 bits wide and holds a full 32 bit address. Address registers do not support byte sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

• DATA ORGANIZATION IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 10. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even), then the second word of that datum is located at address n + 2.

The data types supported by the 68000 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 11. The numbers indicate the order in which the data would be accessed from the processor.

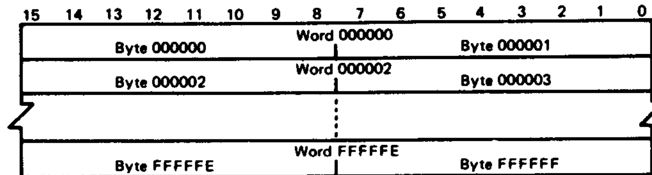


Figure 10 Word Organization in Memory

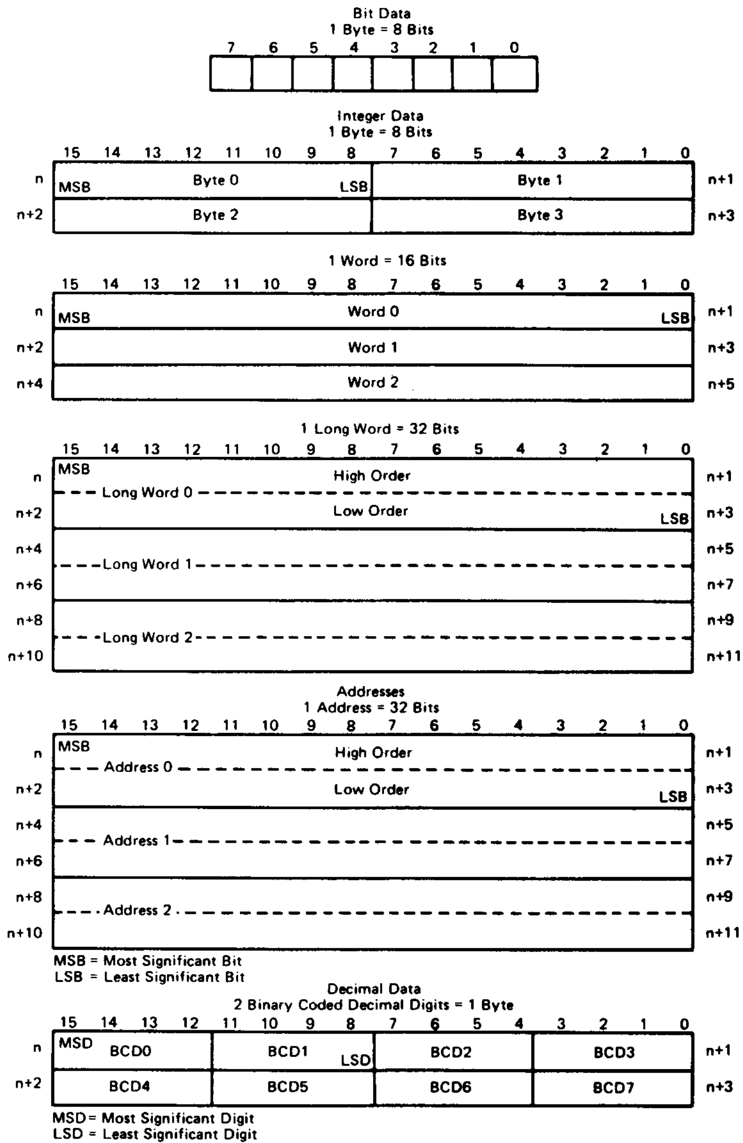


Figure 11 Data Organization in Memory

• ADDRESSING

Instructions for the 68000 contain two kinds of information: the type of function to be performed, and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification – the number of the register is given in the register field of the instruction.

Effective Address – use of the different effective address modes.

Implicit Reference – the definition of certain instructions implies the use of specific registers.

• INSTRUCTION FORMAT

Instructions are from one to five words in length, as shown in Figure 12. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

• PROGRAM/DATA REFERENCES

The 68000 separates memory references into two classes: program references, and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Operand reads are from the data space except in the case of the program counter relative addressing mode. All operand writes are to the data space.

• REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

• EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 13 shows the general format of the single effective address instruction operation word. The effective address is composed of two 3-bit fields: the mode field, and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 12. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

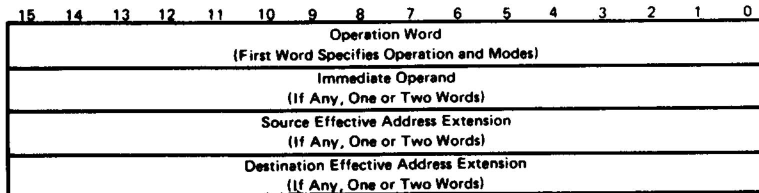


Figure 12 Instruction Format

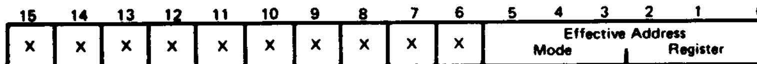


Figure 13 Single-Effective-Address Instruction Operation Word General Format

REGISTER DIRECT MODES

These effective addressing modes specify that the operand is in one of the 16 multifunction registers.

Data Register Direct

The operand is in the data register specified by the effective address register field.

EXAMPLE

MPU

MEMORY

0000ABCD D0

$001F00 ABCD

OWL 31C0

OWL + 2 1F00

MOVE D0, $1F00

COMMENTS
- EA = Dn
- Machine Level Coding

MOVE D0, $1F00

0011 0001 1100 0000

Move Word

Absolute Short

Data Register Direct

Reg #0

Address Register Direct

The operand is in the address register specified by the effective address register field.

EXAMPLE

MPU

MEMORY

00001234 A4

$201000 1234

OWL 33CC

OWL + 2 0020

OWL + 4 1000

MOVE A4, $201000

COMMENTS
- EA = An
- Machine Level Coding

MOVE A4, $201000

0011 0011 1100 1100

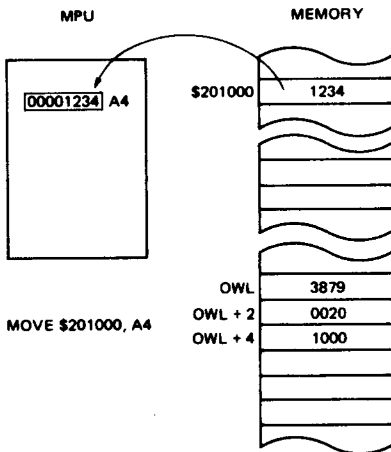
Move Word

Absolute Long

Address Register Direct

Reg #4

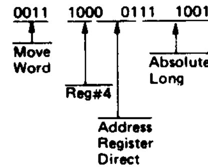
EXAMPLE



COMMENTS

- EA = An
- Address Register Sign Extended
- Machine Level Coding

MOVE $201000, A4



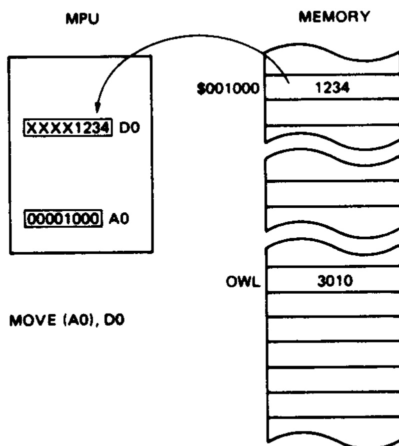
MEMORY ADDRESS MODES

These effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

Address Register Indirect

The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

EXAMPLE

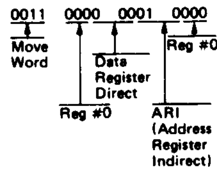


COMMENTS

- EA = (An)

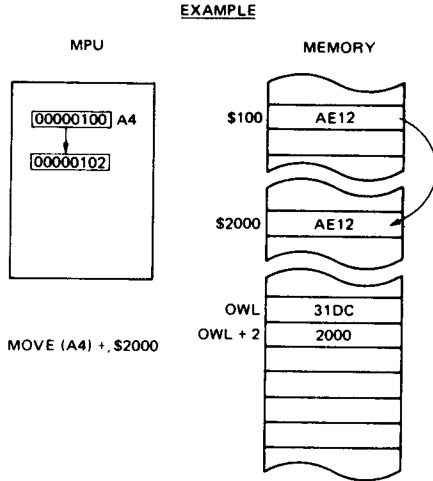
- Machine Level Coding

MOVE (A0), D0



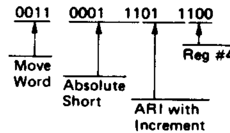
Address Register Indirect With Postincrement

The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whether the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.



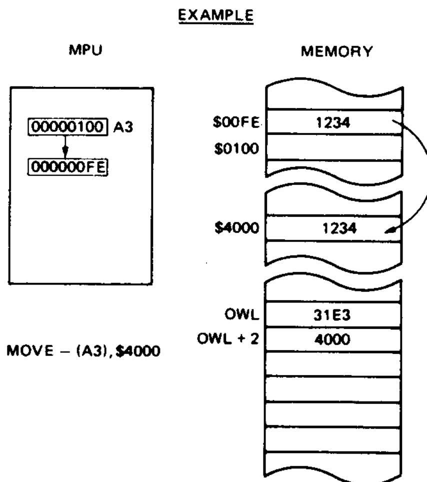
COMMENTS
- EA = (An); An + M ⟶ An
  Where An ⟶ Address Register
  M ⟶ 1, 2, or 4 (Depending Whether Byte, Word, or Long Word)
- Machine Level Coding

MOVE (A4) +, $2000



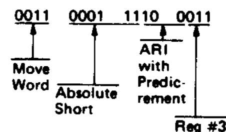
Address Register Indirect With Predecrement

The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.



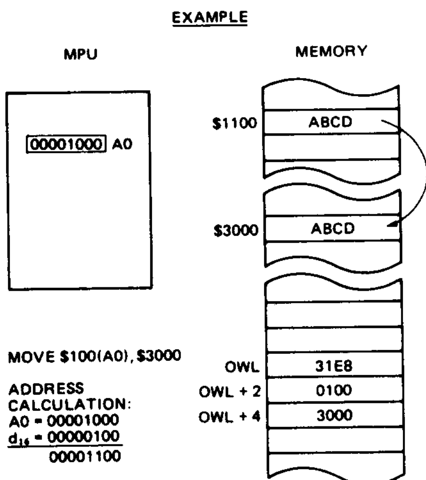
COMMENTS
- An – M ⟶ An; EA = (An)
  Where An ⟶ Address Register
  M ⟶ 1, 2, or 4 (Depending Whether Byte, Word, or Long Word)
- Machine Level Coding

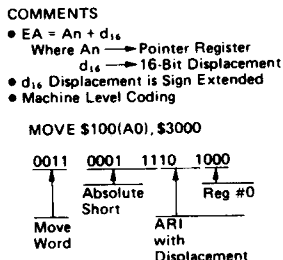
MOVE – (A3), $4000



Address Register Indirect With Displacement

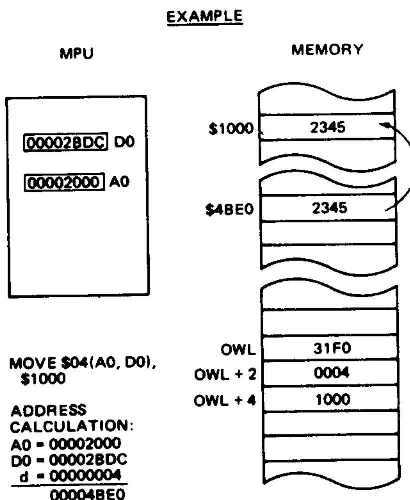
This address mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump to subroutine instructions.

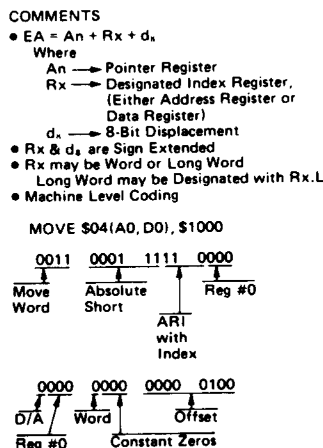




Address Register Indirect With Index

This address mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.





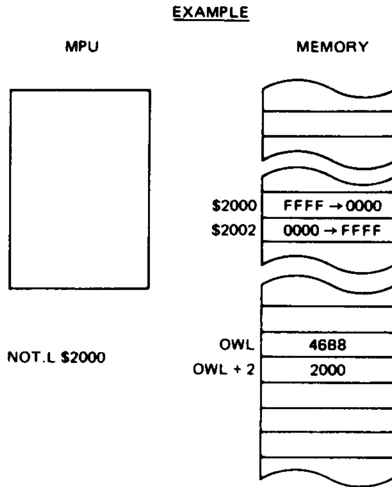
4

SPECIAL ADDRESS MODE

The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

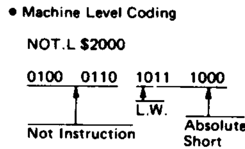
Absolute Short Address

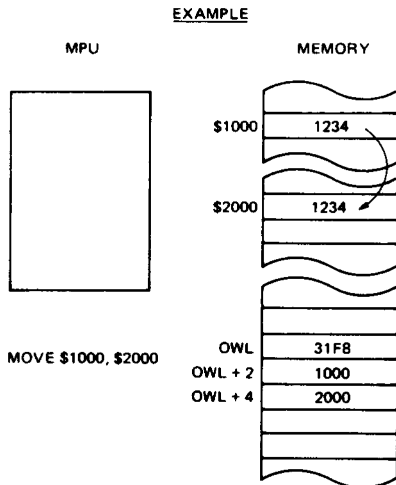
This address mode requires one word of extension. The address of the operand is the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.



COMMENTS
- EA = (Next Word)
- 16-Bit Word is Sign Extended

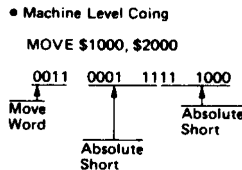
- Machine Level Coding





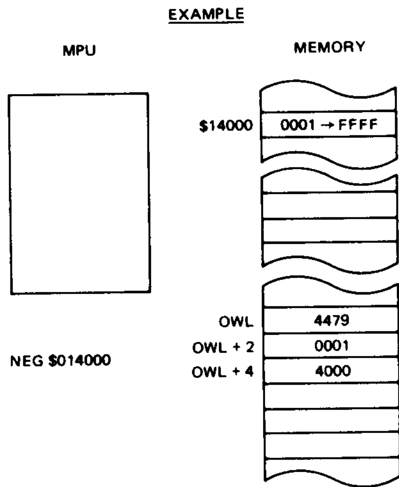
COMMENTS
- EA = (Next Word)
- 16-Bit Word is Sign Extended

- Machine Level Coing



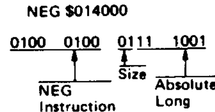
Absolute Long Address

This address mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high-order part of the address is the first extension word; the low-order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.



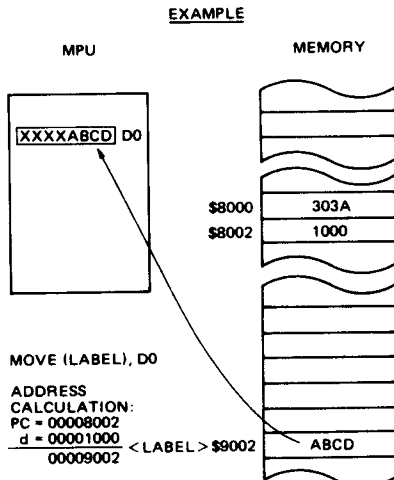
COMMENTS
- EA = (Next Two Words)
- Machine Level Coding

NEG $014000



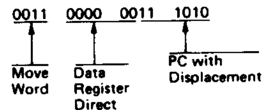
Program Counter With Displacement

This address mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.



COMMENTS
- EA = (PC) + d_{16}
- d_{16} is Sign Extended
- Machine Level Coding

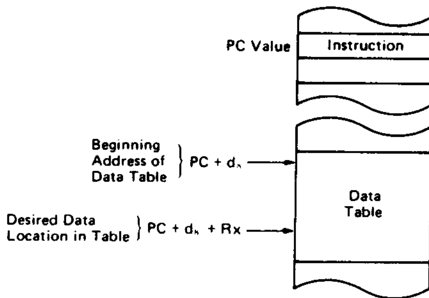
MOVE (LABEL), D0



Program Counter With Index

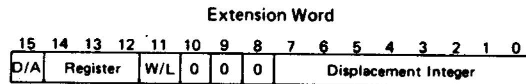
This address mode requires one word of extension. This address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference is classified as a program reference.

EA = (PC) + (Rx) + d_n



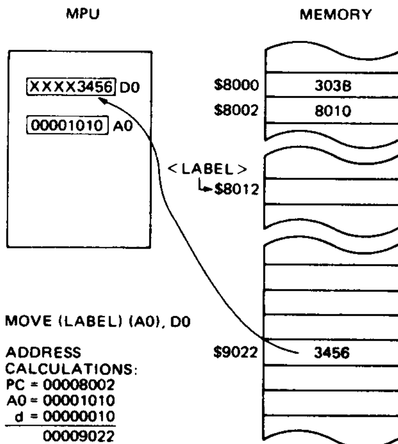
(NOTE)

Extension Word



D/A : Data Register = 0, Address Register = 1
Register : Index Register Number
W/L : Sign-extented, low order Word integer in Index Register = 0
Long Word in Index Register = 1

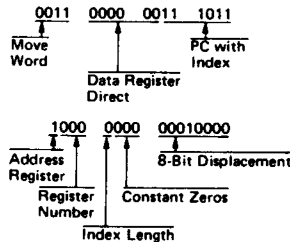
EXAMPLE



COMMENTS

- EA = (PC) + (Rx) + d_8
  Where
  PC → Current Program Counter
  Rx → Designated Index Register (Either Data or Address Register)
  d_8 → 8-Bit Displacement
- Rx and d_8 are Sign Extended
- Rx may be Word or Long Word
  Long Word is Designated with Rx.L
- Machine Level Coding

MOVE (LABEL) (A0), D0



Immediate Data

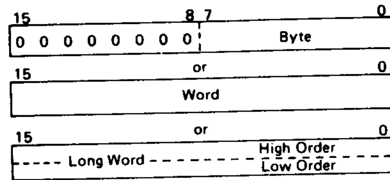
This address mode requires either one or two words of extension depending on the size of the operation.

Byte operation – operand is low order byte of extension word

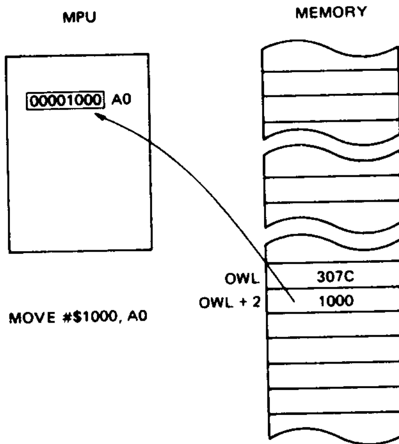
Word operation – operand is extension word

Long word operation – operand is in the two extension words, high-order 16 bits are in the first extension word, low-order 16 bits are in the second extension word.

Extension Word



EXAMPLE

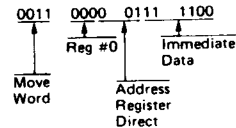


COMMENTS

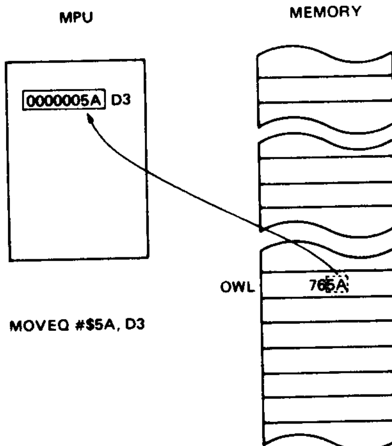
- Data = Next Word(s)
- Data is Sign Extended for Address Register but not Data Register

- Machine Level Coding

MOVE #$1000, A0



EXAMPLE

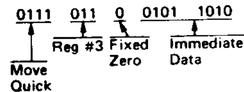


COMMENTS

- Inherent Data
- Data is Sign Extended to Long Word
- Destination must be a Data Register

- Machine Level Coding

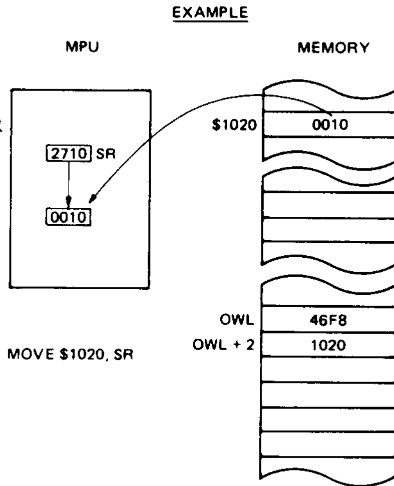
MOVEQ #$5A, D3



Condition Codes or Status Register

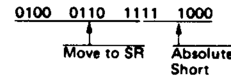
A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR
ANDI to SR
EORI to CCR
EORI to SR
ORI to CCR
ORI to SR
MOVE to CCR
MOVE to SR
MOVE from SR



COMMENTS
- EA = (Next Word)
- Note: This Example is a Privileged Instruction
- Machine Level Coding

MOVE $1020, SR



• EFFECTIVE ADDRESS ENCODING SUMMARY

Table 4 is a summary of the effective addressing modes discussed in the previous paragraphs.

Table 4 Effective Address Encoding Summary

Addressing Mode	Mode	Register
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement	101	register number
Address Register Indirect with Index	110	register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

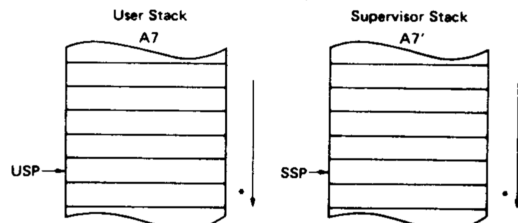
• IMPLICIT REFERENCE

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR).

SYSTEM STACK

The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S-bit in the status register. If the S-bit indicates supervisor state, SSP is the active system stack pointer, and the USP cannot be referenced as an address register. If the S-bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

SYSTEM STACK POINTERS



- Accessed when S = 0
- PC is Stacked on Subroutine Calls in User State

* Increasing Addresses

- Accessed when S = 1
- PC is Stacked on Subroutine Calls in Supervisor State
- Used for Exception Processing

The address mode SP@- creates a new item on the active system stack, and the address mode SP@+ deletes an item from the active system stack.

The program counter is saved on the active system stack on subroutine calls, and restored from the active system stack on returns. On the other hand, both the program counter and the status register are saved on the supervisor stack during the processing of traps and interrupts. Thus, the correct execution of the supervisor state code is not dependent on the behavior of user code and user programs may use the user stack pointer arbitrarily.

In order to keep data on the system stack aligned properly, data entry on the stack is restricted so that data is always put in the stack on a word boundary. Thus byte data is pushed on or pulled from the system stack in the high order half of the word; the lower half is unchanged.

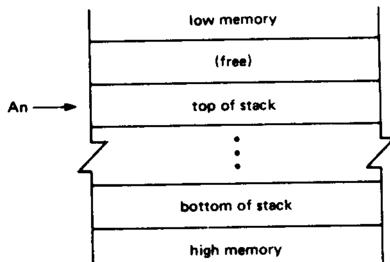
USER STACKS

User stacks can be implemented and manipulated by employing the address register indirect with postincrement and predecrement addressing modes. Using an address register (on of A0 through A6), the user may implement stacks which are filled either from high memory to low memory, or vice versa. The important things to remember are:

- using predecrement, the register is decremented before its contents are used as the pointer into the stack,
- using postincrement, the register is incremented after its contents are used as the pointer into the stack,
- byte data must be put on the stack in pairs when mixed with word or long data so that the stack will not get misaligned when the data is retrieved. Word and long accesses must be on word boundary (even) addresses.

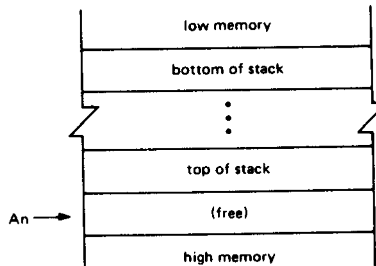
Stack growth from high to low memory is implemented with
An@- to push data on the stack,
An@+ to pull data from the stack.

After eigher a push or a pull operation, register An points to the last (top) item on the stack. This is illustrated as:



Stack growth from low to high memory is implemented with
An@+ to push data on the stack,
An@- to pull data from the stack.

After either a push or a pull operation, register An points to the next available space on the stack. This is illustrated as:

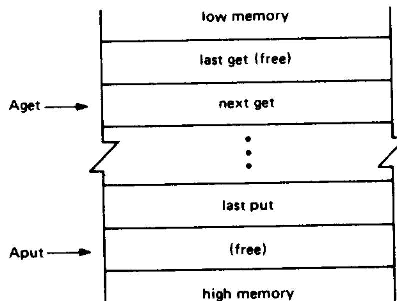


QUEUES

User queues can be implemented and manipulated with the address register indirect with postincrement or predecrement addressing modes. Using a pair of address registers (two of A0 through A6), the user may implement queues which are filled either from high memory to low memory, or vice versa. Because queues are pushed from one end and pulled from the other, two registers are used: the put and get pointers.

Queue growth from low to high memory is implemented with
Aput@+ to put data into the queue,
Aget@+ to get data from the queue.

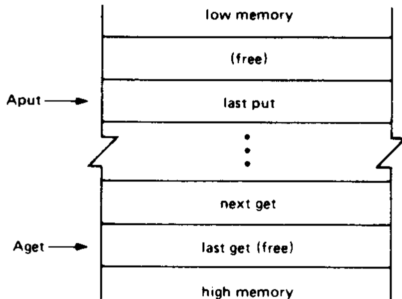
After a put operation, the put address register points to the next available space in the queue and the unchanged get address register points to the next item to remove from the queue. After a get operation, the get address register points to the next item to remove from the queue and the unchanged put address register points to the next available space in the queue. This is illustrated as:



If the queue is to be implemented as a circular buffer, the address register should be checked and, if necessary, adjusted before the put or get operation is performed. The address register is adjusted by subtracting the buffer length (in bytes).

Queue growth from high to low memory is implemented with
Aput@- to put data into the queue,
Aget@ - to get data from the queue.

After a put operation, the put address register points to the last item put in the queue, and the unchanged get address register points to the last item removed from the queue. After a get operation, the get address register points to the last item removed from the queue and the unchanged put address register points to the last item put in the queue. This is illustrated as:



If the queue is to be implemented as a circular buffer, the get or put operation should be performed first, and then the address register should be checked and, if necessary, adjusted. The address register is adjusted by adding the buffer length (in bytes).

■INSTRUCTION SET SUMMARY

The following paragraphs contain an overview of the form and structure of the 68000 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

- Data Movement
- Integer Arithmetic
- Logical
- Shift and Rotate
- Bit Manipulation
- Binary Coded Decimal
- Program Control
- System Control

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

• DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data move instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to memory, and register to register. Address move instructions allow word and long word operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), and move quick (MOVEQ). Table 5 is a summary of the data movement operations.

Table 5 Data Movement Operations

Instruction	Operand Size	Operation
EXG	32	Rx ↔ Ry
LEA	32	EA → An
LINK	–	An → –(SP) SP → An; SP + d → SP
MOVE	8, 16, 32	(EA)s → EAd
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA
MOVEP	16, 32	(EA) → Dn Dn → EA
MOVEQ	8	#xxx → Dn
PEA	32	EA → –(SP)
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	An → Sp; (SP) + → An

(NOTES)
s = source
d = destination
[ ] = bit numbers
–( ) = indirect with predecrement
( ) + = indirect with postincrement
\# = immediate data

• INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotien with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 6 is a summary of the integer arithmetic operations.

Table 6 Integer Arithmetic Operations

Instruction	Operand Size	Operation
ADD	8, 16, 32 16, 32	Dn + (EA) → Dn (EA)+ Dn → EA (EA) + #xxx → EA AN + (EA) → An
ADDX	8, 16, 32 16, 32	Dx + Dy + X → Dx –(Ax) + – (Ay) + X → (Ax)
CLR	8, 16, 32	(EA) → MPU 0 → EA
CMP	8, 16, 32 16, 32	Dn - (EA) (EA) - #xxx (Ax) + – (Ay) + An - (EA)
DIVS	32 ÷ 16	Dn÷(EA)→ Dn
DIVU	32 ÷ 16	Dn÷(EA)→ Dn
EXT	8 → 16 16 → 32	$(Dn)_8 \rightarrow Dn_{16}$ $(Dn)_{16} \rightarrow Dn_{32}$
MULS	16×16 → 32	Dn×(EA) → Dn
MULU	16×16 → 32	Dn×(EA) → Dn
NEG	8, 16, 32	0 - (EA) → EA
NEGX	8, 16, 32	0 - (EA) - X - EA
SUB	8, 16, 32 16, 32	Dn - (EA) → Dn (EA) - Dn → EA (EA) - #xxx → EA An - (EA) → An
SUBX	8, 16, 32	Dx - Dy - X → Dx –(Ax) – – (Ay) – X → (Ax)
TAS	8	(EA) - 0, 1 → EA[7]
TST	8, 16, 32	(EA) - 0

(NOTE) [ ] = bit number
– ( ) = indirect with predecrement
( ) + = indirect with postincrement
= immediate data

• LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 7 is a summary of the logical operations.

Table 7 Logical Operations

Instruction	Operand Size	Operation
AND	8, 16, 32	Dn∧(EA) → Dn (EA)∧ Dn → EA (EA)∧ #xxx → EA
OR	8, 16, 32	Dn v (EA) → Dn (EA) v Dn → EA (EA) v #xxx → EA
EOR	8, 16, 32	(EA)⊕Dy → EA (EA)⊕ #xxx → EA
NOT	8, 16, 32	~ (EA) → EA

[NOTE] ~ = invert
v = logical OR
# = immediate data
∧ = logical AND
⊕ = exclusive OR

• SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by the arithmetic instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in the instruction of one to eight bits, or 0 to 63 specified in a data register.

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates. Table 8 is a summary of the shift and rotate operations.

Table 8 Shift and Rotate Operations

Instruction	Operand Size	Operation
ASL	8, 16, 32	X/C ← [←] ← 0
ASR	8, 16, 32	[→] → X/C
LSL	8, 16, 32	X/C ← [←] ← 0
LSR	8, 16, 32	0 → [→] → X/C
ROL	8, 16, 32	C ← [←]
ROR	8, 16, 32	[→] → C
ROXL	8, 16, 32	C ← [←] ← X
ROXR	8, 16, 32	X → [→] → C

• BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 9 is a summary of the bit manipulation operations. (Bit 2 of the status register is Z.)

Table 9 Bit Manipulation Operations

Instruction	Operand Size	Operation
BTST	8, 32	~ bit of (EA) → Z
BSET	8, 32	~ bit of (EA) → Z; 1 → bit of EA
BCLR	8, 32	~ bit of (EA) → Z; 0 → bit of EA
BCHG	8, 32	~ bit of (EA) → Z; ~ bit of (EA) → bit of EA

(Note) ~ = invert

• BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 10 is a summary of the binary coded decimal operations.

Table 10 Binary Coded Decimal Operations

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $-(Ax)_{10} + -(Ay)_{10} + X \rightarrow (Ax)$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $-(Ax)_{10} - -(Ay)_{10} - X \rightarrow (Ax)$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$

– ( ) = indirect with predecrement

• PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 11.

The conditional instructions provide setting and branching for the following conditions:

CC	– carry clear	LS	– low or same
CS	– carry set	LT	– less than
EQ	– equal	MI	– minus
F	– never true	NE	– not equal
GE	– greater or equal	PL	– plus
GT	– greater than	T	– always true
HI	– high	VC	– no overflow
LE	– less or equal	VS	– overflow

Table 11 Program Control Operations

Instruction	Operation
Conditional	
B_{CC}	Branch conditionally (14 conditions) 8- and 16-bit displacement
DB_{CC}	Test condition, decrement, and branch 16-bit displacement
S_{CC}	Set byte conditionally (16 conditions)
Unconditional	
BRA	Branch always 8-and 16-bit displacement
BSR	Branch to subroutine 8- and 16-bit displacement
JMP	Jump
JSR	Jump to subroutine
Returns	
RTR	Return and restore condition codes
RTS	Return from subroutine

• SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 12.

Table 12 System Control Operations

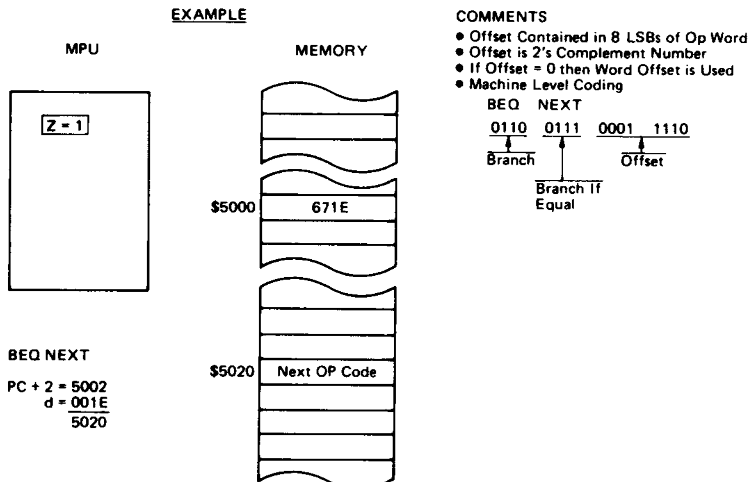
Instruction	Operation
Privileged	
RESET	Reset external devices
RTE	Return from exception
STOP	Stop program execution
ORI to SR	Logical OR to status register
MOVE USP	Move user stack pointer
ANDI to SR	Logical AND to status register
EORI to SR	Logical EOR to status register
MOVE EA to SR	Load new status register
Trap Generating	
TRAP	Trap
TRAPV	Trap on overflow
CHK	Check register against bounds
Status Register	
ANDI to CCR	Logical AND to condition codes
EORI to CCR	Logical EOR to condition codes
MOVE EA to CCR	Load new condition codes
ORI to CCR	Logical OR to condition codes
MOVE SR to EA	Store status register

• BRANCH INSTRUCTION ADDRESSING

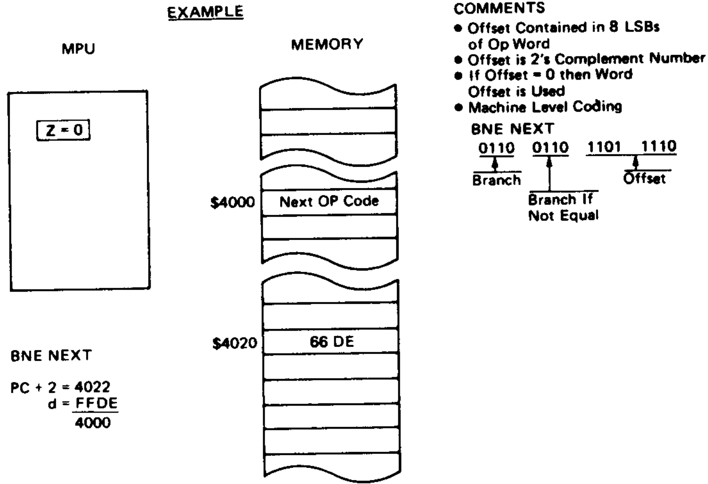
BRANCH INSTRUCTION FORMAT

	15 — 8	7 — 0
Operation Word	Operation Code	8 bit Displacement
Extension Word	16 bit Displacement if 8 bit Displacement = 0	

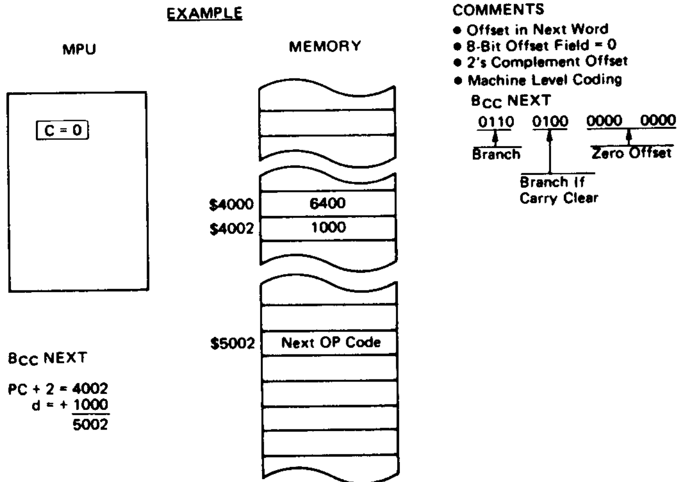
RELATIVE, FORWARD REFERENCE, 8-BIT OFFSET



RELATIVE, BACKWARD REFERENCE 8-BIT OFFSET



RELATIVE, FORWARD REFERENCE, 16-BIT OFFSET



▪ SIGNAL AND BUS OPERATION DESCRIPTION

The following paragraphs contain a brief description of the input and output signals. A discussion of bus operation during the various machine cycles and operations is also given.

(NOTE) The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true independent of whether that voltage is low or high. The term negate or negation is used to indicate that a signal is inactive or false.

• SIGNAL DESCRIPTION

The input and output signals can be functionally organized into the groups shown in Figure 14. The following paragraphs provide a brief description of the signals and also a reference (if applicable) to other paragraphs that contain more detail about the function being performed.

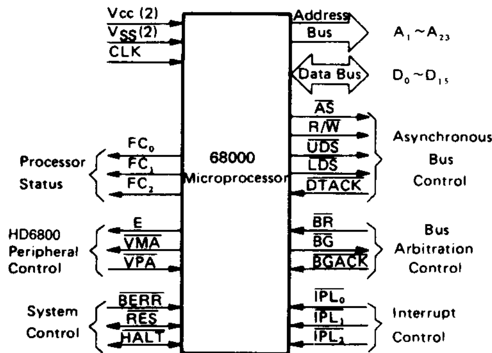


Figure 14 Input and Output Signals

ADDRESS BUS (A_1 through A_{23})

This 23-bit, unidirectional, three-state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A_1, A_2, and A_3. Provide information about what level interrupt is being serviced while address lines A_4 through A_{23} are all set to a logic high.

DATA BUS (D_0 through D_{15})

This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte length. During an interrupt acknowledge cycle, an external device supplies the vector number on data lines D_0 through D_7.

ASYNCHRONOUS BUS CONTROL

Asynchronous data transfer are handled using the following control signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.

Address Strobe ($\overline{AS}$)

This signal indicates that there is a valid address on the address bus.

Read/Write (R/$\overline{W}$)

This signal defines the data bus transfer as a read or write cycle. The R/$\overline{W}$ signal also works in conjunction with the upper and lower data strobes as explained in the following paragraph.

Upper and Lower Data Strobes ($\overline{UDS}$, $\overline{LDS}$)

These signals control the data on the data bus, as shown in Table 13. When the R/$\overline{W}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{W}$ line is low, the processor will write to the data bus as shown.

Table 13 Data Strobe Control of Data Bus

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	$D_8 \sim D_{15}$	$D_0 \sim D_7$
High	High	–	No valid data	No valid data
Low	Low	High	Valid data bits 8 ~ 15	Valid data bits 0 ~ 7
High	Low	High	No valid data	Valid data bits 0 ~ 7
Low	High	High	Valid data bits 8 ~ 15	No valid data
Low	Low	Low	Valid data bits 8 ~ 15	Valid data bits 0 ~ 7
High	Low	Low	Valid data bits 0 ~ 7*	Valid data bits 0 ~ 7
Low	High	Low	Valid data bits 8 ~ 15	Valid data bits 8 ~ 15*

* These conditions are a result of current implementation and may not appear on future devices.

Data Transfer Acknowledge ($\overline{DTACK}$)

This input indicates that the data transfer is completed. When the processor recognizes $\overline{DTACK}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle is terminated. (Refer to ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION)

BUS ARBITRATION CONTROL

These three signals form a bus arbitration circuit to determine which device will be the bus master device.

Bus Request ($\overline{BR}$)

This input is wire ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

Bus Grant ($\overline{BG}$)

This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

Bus Grand Acknowledge ($\overline{BGACK}$)

This input indicates that some other device has become the bus master. This signal cannot be asserted until the following four conditions are met:

(1) A Bus Grant has been received
(2) Address Strobe is inactive which indicates that the microprocessor is not using the bus
(3) Data Transfer Acknowledge is inactive which indicates

that neither memory nor peripherals are using the bus
(4) Bus Grant Acknowledge is inactive which indicates that no other device is still claiming bus mastership.

INTERRUPT CONTROL ($\overline{IPL_0}$, $\overline{IPL_1}$, $\overline{IPL_2}$)

These input pins indicate the encoded priority lèvel of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. Level seven can not be masked. The least significant bit is given in $\overline{IPL_0}$ and the most significant bit is contained in $\overline{IPL_2}$. These lines must remain stable until the processor signals interrupt acknowledge (FC_0 ~ FC_2 are all high) to insure that the interrupt is recognized.

SYSTEM CONTROL

The system control inputs are used to either reset or halt the processor and to indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

Bus Error ($\overline{BERR}$)

This input informs the processor that there is a problem with the cycle currently being executed. Problems may be a result of:

(1) Nonresponding devices
(2) Interrupt vector number acquisition failure
(3) Illegal access request as determined by a memory management unit
(4) Other application dependent errors.

The bus error signal interacts with the halt signal to determine if exception processing should be performed or if the current bus cycle should be retried.

Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction of the bus error and halt signals.

Reset ($\overline{RES}$)

This bidirectional signal line acts to reset (initiate a system initialization sequence) the processor in response to an external reset signal. An internally generated reset (result of a RESET instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external HALT and RESET signals applied at the same time. Refer to **RESET OPERATION** paragraph for additional information about reset operation.

Halt ($\overline{HALT}$)

When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state. Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction between the halt and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition, the halt line is driven by the processor to indicate to external devices that the processor has stopped.

HMCS6800 PERIPHERAL CONTROL

These control signals are used to allow the interfacing of synchronous HD6800 peripheral devices with the asynchronous 68000. These signals are explained in the following paragraphs.

Enable (E)

This signal is the standard enable signal common to all HD6800 type peripheral devices. The period for this output is ten 68000 clock periods (six clocks low; four clocks high). Enable is generated by an internal ring counter which may come up in any state (i.e., at power on, it is impossible to guarantee phase relationship of E to CLK), E is a free-running clock and runs regardless of the state of the bus on the MPU.

Valid Peripheral Address ($\overline{VPA}$)

This input indicates that the device or region addressed is a HD6800 family device and that data transfer should be synchronized with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt. Refer to **INTERFACE WITH HD6800 PERIPHERALS. ALS.**

Valid Memory Address ($\overline{VMA}$)

This output is used to indicate to HD6800 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is a HD6800 family device.

PROCESSOR STATUS (FC_0, FC_1, FC_2)

These function code outputs indicate the state (user or supervisor) and the cycle type currently being executed, as shown in Table 14. The information indicated by the function code outputs is valid whenever address strobe ($\overline{AS}$) is active.

Table 14 Function Code Outputs

FC_2	FC_1	FC_0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Superviser Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

CLOCK (CLK)

The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input should not be gated off at any time, and the clock signal must conform to minimum and maximum pulse width time.

SIGNAL SUMMARY

Table 15 is a summary of all the signals discussed in the previous paragraphs.

• BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

Table 15 Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	Three State	
				On $\overline{BGACK}$	On $\overline{HALT}$
Address Bus	$A_1 \sim A_{23}$	output	high	yes	yes
Data Bus	$D_0 \sim D_{15}$	input/output	high	yes	yes
Address Strobe	$\overline{AS}$	output	low	yes	no
Read/Write	$R/\overline{W}$	output	read-high write-low	yes	no
Upper and Lower Data Strobes	$\overline{UDS}$, $\overline{LDS}$	output	low	yes	no
Data Transfer Acknowledge	$\overline{DTACK}$	input	low	no	no
Bus Request	$\overline{BR}$	input	low	no	no
Bus Grant	$\overline{BG}$	output	low	no	no
Bus Grant Acknowledge	$\overline{BGACK}$	input	low	no	no
Interrupt Priority Level	$\overline{IPL_0}$, $\overline{IPL_1}$, $\overline{IPL_2}$	input	low	no	no
Bus Error	$\overline{BERR}$	input	low	no	no
Reset	$\overline{RES}$	input/output	low	no*	no*
Halt	$\overline{HALT}$	input/output	low	no*	no*
Enable	E	output	high	no	no
Valid Memory Address	$\overline{VMA}$	output	low	yes	no
Valid Peripheral Address	$\overline{VPA}$	input	low	no	no
Function Code Output	FC_0, FC_1, FC_2	output	high	yes	no
Clock	CLK	input	high	no	no
Power Input	V_{CC}	input	–	–	–
Ground	V_{SS}	input	–	–	–

* Open drain

DATA TRANSFER OPERATIONS

Transfer of data between devices involve the following leads:

(1) Address Bus A_1 through A_{23}

(2) Data Bus D_0 through D_{15}

(3) Control Signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the 68000 for interlocked multiprocessor communications.

Read Cycle

During a read cycle, the processor receives data from memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the processor reads both upper and lower bytes simultaneously by asserting both upper and lower data strobes. When the instruction specifies byte operation, the processor uses an internal Ao bit to determine which byte to read and then issues the data strobe required for that byte. For bytes operations, when the Ao bit equals zero, the upper data strobe is issued. When the Ao bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally.

A word read cycle flow chart is given in Figure 15. A byte read cycle flow chart is given in Figure 16. Read cycle timing is given in Figure 17. Figure 18 details word and byte read cycle operations. Refer to these illustrations during the following detailed.

At state zero (S0) in the read cycle, the address bus (A_1 through A_{23}) is in the high impedance state. A function code is asserted on the function code output line (FC_0 through FC_2). The read/write (R/W) signal is switched high to indicate a read cycle. One half clock cycle later, at state 1, the address bus is released from the high impedance state. The function code outputs indicate which address space that this cycle will operate on.

In state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus and the upper and lower data strobe ($\overline{UDS}$, $\overline{LDS}$) is asserted as required. The memory or peripheral device uses the address bus and the address strobe to determine if it has been selected. The selected device uses the read/write signal and the data strobe to place its information on the data bus. Concurrent with placing data on the data bus, the selected device asserts data transfer acknowledge ($\overline{DTACK}$).

Data transfer acknowledge must be present at the processor at the start of state 5 or the processor will substitute wait states for states 5 and 6. State 5 starts the synchronization of the returning data transfer acknowledge. At the end of state 6 (beginning of state 7) incoming data is latched into an internal data bus holding register.

During state 7, address strobe and the upper and/or lower data strobes are negated. The address bus is held valid through state 7 to allow for static memory operation and signal skew.

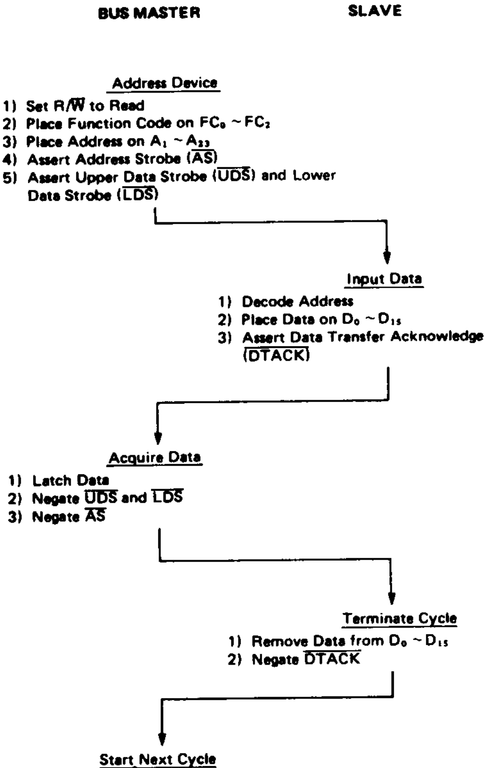


Figure 15 Word Read Cycle Flow Chart

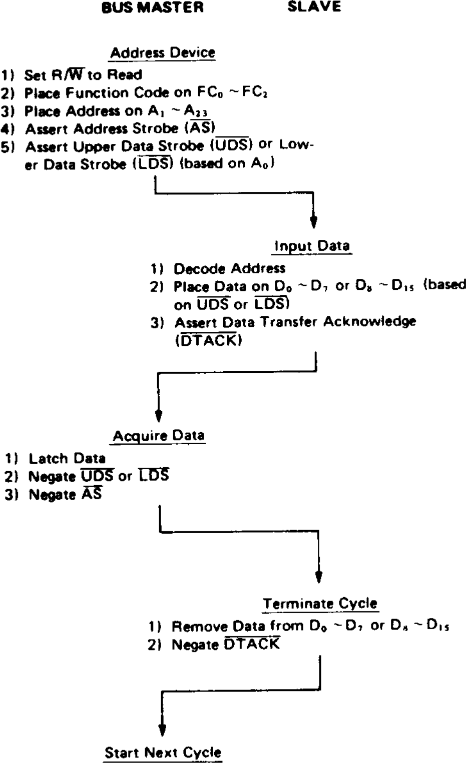


Figure 16 Byte Read Cycle Flow Chart

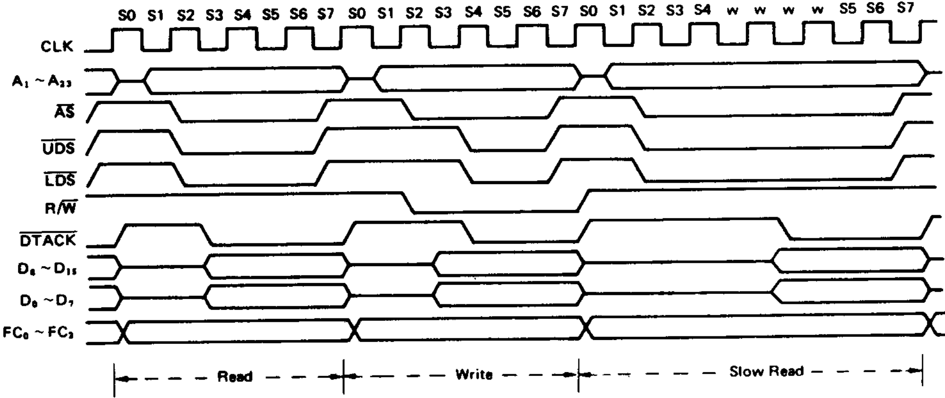


Figure 17 Read and Write Cycle Timing Diagram

4

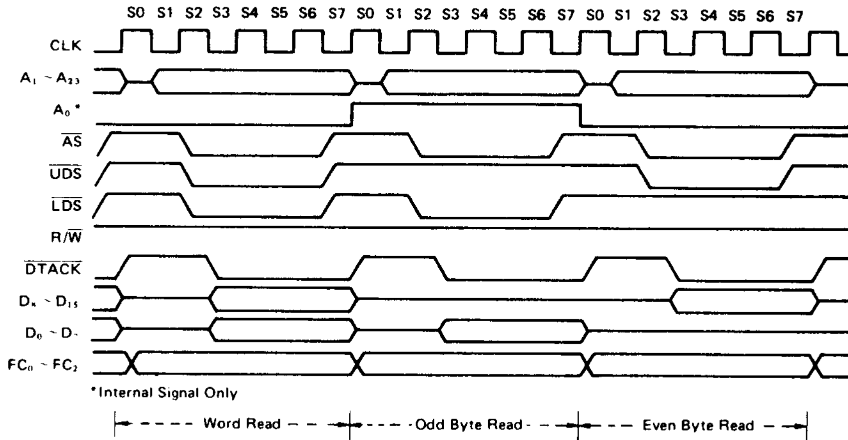


Figure 18 Word and Byte Read Cycle Timing Diagram

The read/write signal and the function code outputs also remain valid through state 7 to ensure a correct transfer operation. The slave device keeps its data asserted until it detects the negation of either the address strobe or the upper and/or lower data strobe. The slave device must remove its data and data transfer acknowledge within one clock period of recognizing the negation of the address or data strobes. Note that the data bus might not become free and data transfer acknowledge might not be removed until state 0 or 1.

When address strobe is negated, the slave device is released. Note that a slave device must remain selected as long as address strobe is asserted to ensure the correct functioning of the read-modify-write cycle.

Write Cycle

During a write cycle, the processor sends data to memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A_0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A_0 bit equals zero, the upper data strobe is issued. When the A_0 bit equals one, the lower data strobe is issued. A word write cycle flow chart is given in Figure 19. A byte write cycle flow chart is given in Figure 20. Write cycle timing is given in Figure 17. Figure 21 details word and byte write cycle operation. Refer to these illustrations during the following detailed discussion.

At state zero (S0) in the write cycle, the address bus (A_1 through A_{23}) is in the high impedance state. A function code is asserted on the function code output line (FC_0 through FC_2).

(NOTE) The read/write (R/$\overline{W}$) signal remains high until state 2 to prevent bus conflicts with preceding read cycles. The data bus is not driven until state 3.

One half clock later, at state 1, the address bus is released from the high impedance state. The function code outputs indicate which address space that this cycle will operate on.

In state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus. The memory or peripheral device uses the address bus and the address strobe to determine if it has been selected. During state 2, the read/write signal is switched low to indicate a write cycle. When external processor data bus buffers are required, the read/write line provides sufficient directional control. Data is not asserted during this state to allow sufficient turn around time for external data buffers (if used). Data is asserted onto the data bus during state 3.

In state 4, the data strobes are asserted as required to indicate that the data bus is stable. The selected device uses the read/write signal and the data strobes to take its information from the data bus. The selected device asserts data transfer acknowledge ($\overline{DTACK}$) when it has successfully stored the data.

Data transfer acknowledge must be present at the processor at the start of state 5 or the processor will substitute wait states for states 5 and 6. State 5 starts the synchronization of the returning data transfer acknowledge.

During state 7, address strobe and the upper and/or lower data strobes are negated. The address and data buses are held valid through state 7 to allow for static memory operation and signal skew. The read/write signal and the function code outputs also remain valid through state 7 to ensure a correct transfer operation. The slave device keeps its data transfer acknowledge asserted until it detects the negation of either the address strobe or the upper and/or lower data strobe. The slave device must remove its data transfer acknowledge within one clock period after recognizing the negation of the address or data strobes. Note that the processor releases the data bus at the end of state 7 but that data transfer acknowledge might not be removed until state 0 or 1. When address strobe is negated, the slave device is released.

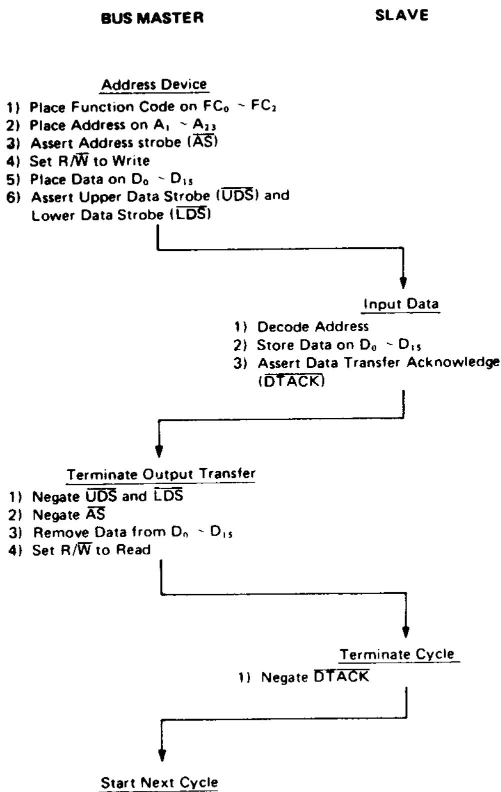


Figure 19 Word Write Cycle Flow Chart

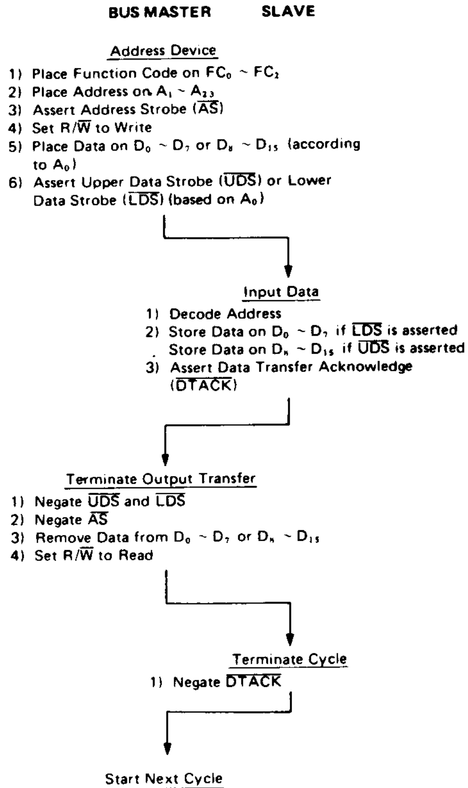


Figure 20 Byte Write Cycle Flow Chart

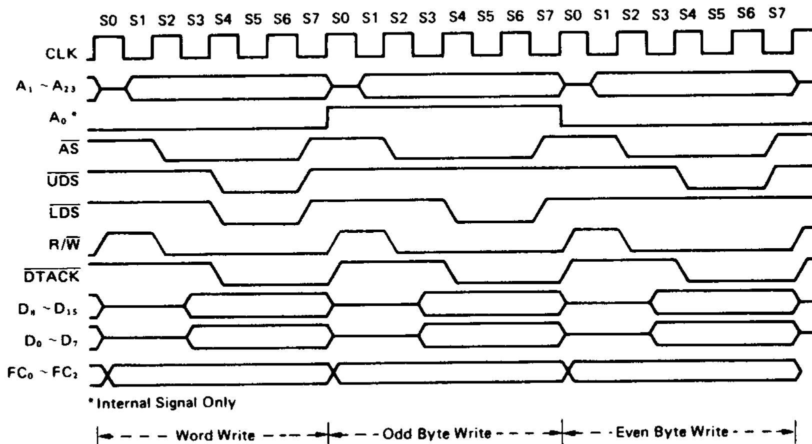


Figure 21 Word and Byte Write Cycle Timing Diagram

Read-Modify-Write Cycle

The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the 68000 this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycle and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flow chart is given in Figure 22 and a timing diagram is given in Figure 23. Refer to these illustrations during the following detailed discussions.

At state zero (S0) in the read-modify-write cycle, the address bus (A_1 through A_{23}) is in the high impedance state. A function code is asserted on the function code output line (FC_0 through FC_2). The read/write ($R/\overline{W}$) signal is switched high to indicate a read cycle. One half clock cycle later, at state 1, the address bus is released from the high impedance state. The function code outputs indicate which address space that this cycle will operate on.

In state 2, the address strobe (AS) is asserted to indicate that there is a valid address on the address bus and the upper or lower data strobe ($\overline{UDS}$, $\overline{LDS}$) is asserted as required. The memory or peripheral device uses the address bus and the address strobe to determine if it has been selected. The selected device uses the read/write signal and the data strobe to place its information on the data bus. Concurrent with placing data on the data bus, the selected device asserts data transfer acknowledge ($\overline{DTACK}$).

Data transfer acknowledge must be present at the processor at the start of state 5 or the processor will substitute wait states for states 5 and 6. State 5 starts the synchronization of the returning data transfer acknowledge. At the end of state 6 (beginning of state 7) incoming data is latched into an internal data bus holding register.

During state 7, the upper or lower data strobe is negated. The address bus, address strobe, read/write signal, and function code outputs remain as they were in preparation for the write portion of the cycle. The slave device keeps its data asserted until it detects the negation of the upper or lower data strobe. The slave device must remove its data and data transfer acknowledge within one clock period of recognizing the negation of the data strobes. Internal modification of data may occur from state 8 to state 11.

(NOTE) The read/write signal remains high until state 14 to prevent bus conflicts with the preceding read portion of the cycle and the data bus is not asserted by the processor until state 15.

In state 14, the read/write signal is switched low to indicate a write cycle. When external processor data bus buffers are required, the read/write line provides sufficient directional control. Data is not asserted during this state to allow sufficient turn around time for external data buffers (if used). Data is asserted onto the data bus during state 15.

In state 16, the data strobe is asserted as required to indicate that the data bus is stable. The selected device uses the read/write signal and the data strobe to take its information from the data bus. The selected device asserts data transfer acknowledge ($\overline{DTACK}$) when it has successfully stored its data.

Data transfer acknowledge must be present at the processor at the start of state 17 or the processor will substitute wait states for states 17 and 18. State 17 starts the synchronization of the returning data transfer acknowledge for the write portion of the cycle. The bus interface circuitry issues requests for subsequent internal cycles during state 18.

During state 19, address strobe and the upper or lower data strobe is negated. The address and data buses are held valid through state 19 to allow for static memory operation and signal skew. The read/write signal and the function code outputs also remain valid through state 19 to ensure a correct transfer operation. The slave device keeps its data transfer acknowledge asserted until it detects the negation of either the address strobe or the upper or lower data strobe. The slave device must remove its data transfer acknowledge within once clock period after recognizing the negation of the address or data strobes. Note that the processor releases the data bus at the end of state 19 but that data transfer acknowledge might not be removed until state 0 or 1. When address strobe is negated the slave device is released.

BUS ARBITRATION

Bus arbitration is a technique used by master-type devices to request, be granted, and acknowledge bus mastership. In its simplest form, it consists of:

(1) Asserting a bus mastership request.
(2) Receiving a grant that the bus is available at the end of the current cycle.
(3) Acknowledging that mastership has been assumed.

Figure 24 is a flow chart showing the detail involved in a request from a single device. Figure 25 is a timing diagram for the same operations. This technique allows processing of bus requests during data transfer cycles.

The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the processor and one device capable of bus mastership. In systems having a number of devices capable of bus mastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge ($\overline{BGACK}$) signal.

However, if the bus requests are still pending, the processor will assert another bus grant within a few clock cycles after it was negated. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has completed its requirements. The following paragraphs provide additional information about the three steps in the arbitration process.

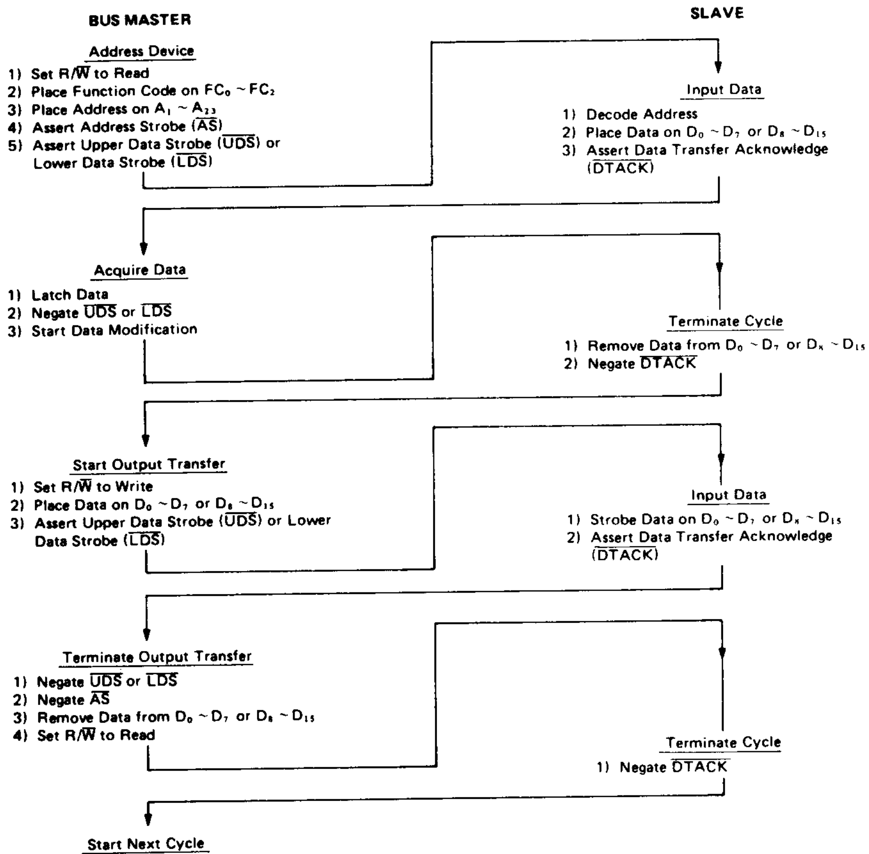


Figure 22 Read-Modify-Write Cycle Flow Chart

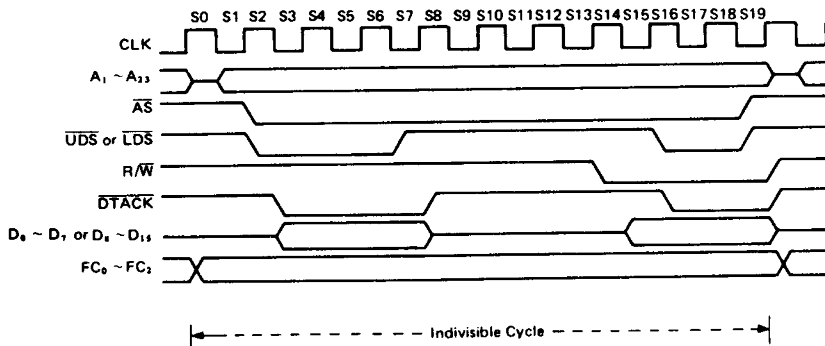


Figure 23 Read-Modify-Write Cycle Timing Diagram

PROCESSOR **REQUESTING DEVICE**

Request the Bus
1) Assert Bus Request (BR)

Grant Bus Arbitration
1) Assert Bus Grant (BG)

Acknowledge Bus Mastership
1) External arbitration determines next bus master
2) Next bus master waits for current cycle to complete
3) Next bus master asserts Bus Grant Acknowledge (BGACK) to become new master
4) Bus master negates BR

Terminate Arbitration
1) Negate BG (and wait for BGACK to be negated)

Operate as Bus Master
1) Perform Data Transfers (Read and Write cycles) according to the same rules the processor uses.

Release Bus Mastership
1) Negate BGACK

Re-Arbitrate or Resume Processor Operation

Figure 24 Bus Arbitration Cycle Flow Chart

Requesting the Bus

External devices capable of becoming bus masters request the bus by asserting the bus request (BR) signal. This is a wire ORed signal (although it need not be constructed from open collector devices) that indicates to the processor that some external device requires control of the external bus. The processor is effectively at a lower bus priority level that the external device and will relinquish the bus after it has completed the last bus cycle it has started.

When no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

Receiving the Bus Grant

The processor asserts bus grant (BG) as soon as possible. Normally this is immediately after internal synchronization. The only exception to this occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe (AS) signal. In this case, bus grant will not be asserted until one clock after address strobe is asserted to indicate to external devices that a bus cycle is being executed.

The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

Acknowledgement of Mastership

Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowledge, and bus grant acknowledge are negated before issuing its own BGACK. The negation of the address strobe indicates that the previous master has completed its cycle, the negation of bus grant acknowledge indicates that the previous master has released the bus. (While address strobe is asserted no device is allowed to "break into" a cycle.) The negation of data transfer acknowledge indicates the previous slave has terminated its connection to the previous master. Note that in some applications data

CLK
$A_1 \sim A_{23}$
AS
LDS/UDS
R/W
DTACK
$D_0 \sim D_{15}$
$FC_0 \sim FC_2$
BR
BG
BGACK
Processor — DMA Device — Processor — DMA Device

Figure 25 Bus Arbitration Cycle Timing Diagram

transfer acknowledge might not enter into this function. General purpose devices would then be connected such that they were only dependent on address strobe. When bus grant acknowledge is issued the device is bus master until it negates bus grant acknowledge. Bus grant acknowledge should not be negated until after the bus cycle(s) is (are) completed. Bus mastership is terminated at the negation of bus grant acknowledge.

The bus request from the granted device should be dropped after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of bus grant. Refer to Bus Arbitration Control section. Note that the processor does not perform any external bus cycles before it re-asserts bus grant.

BUS ARBITRATION CONTROL

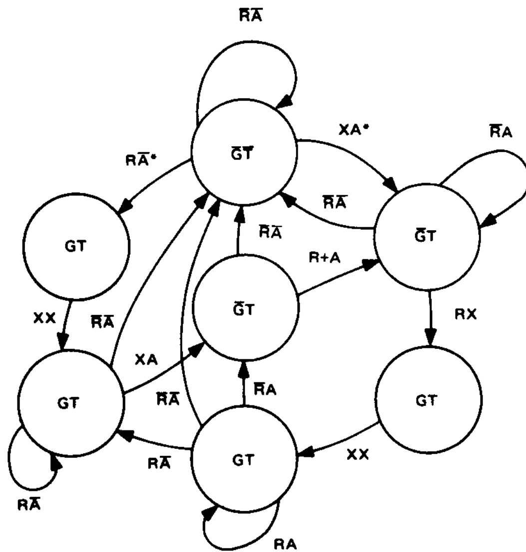
The bus arbitration control unit in the 68000 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 26. All asynchronous signals to the 68000 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 27). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.

As shown in Figure 26, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is lebeled G and the internal three-state control signal T. If T is true, the address, data, function code line, and control buses are placed in a high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high) regardless of their true active voltage level.

State changes (valid outputs) occur on the next rising edge after the internal signal is valid.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 28. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 29.

If a bus request is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0), $\overline{BG}$ will not be asserted on the next rising edge. Instead, $\overline{BG}$ will be delayed until the second rising edge following it's internal assertion. This sequence is shown in Figure 30.



R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic**
X = Don't Care

* State machine will not change state if bus is in S0. Refer to BUS ARBITRATION CONTROL for additional information.
** The address bus will be placed in the high impedance state if T is asserted and $\overline{AS}$ is negated.

Figure 26 State Diagram of 68000 Bus Arbitration Unit

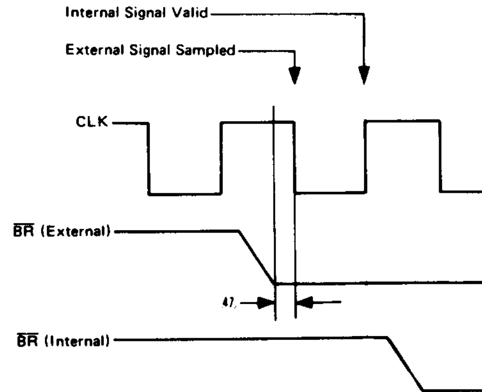


Figure 27 Timing Relationship of External Asynchronous Inputs to Internal Signals

4

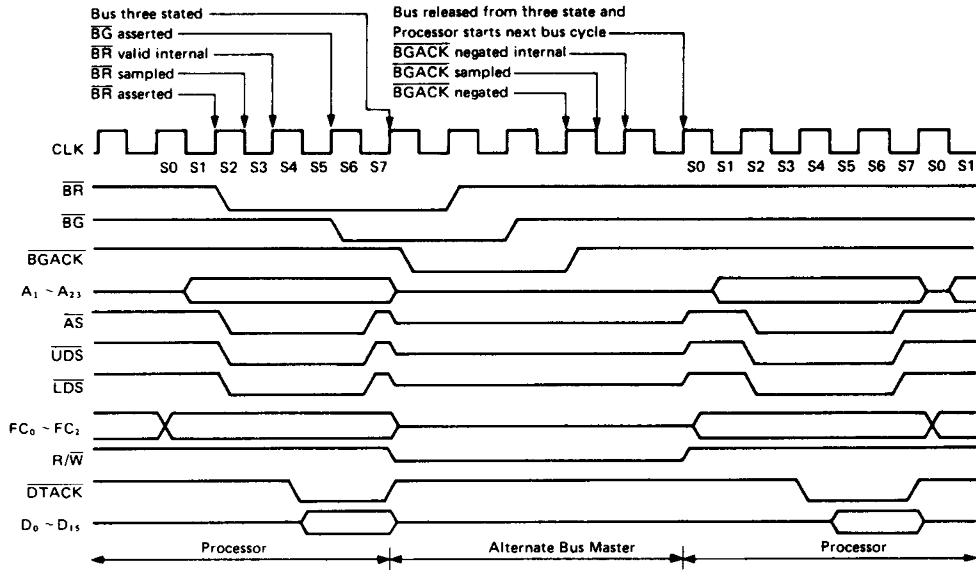


Figure 28 Bus Arbitration During Processor Bus Cycle

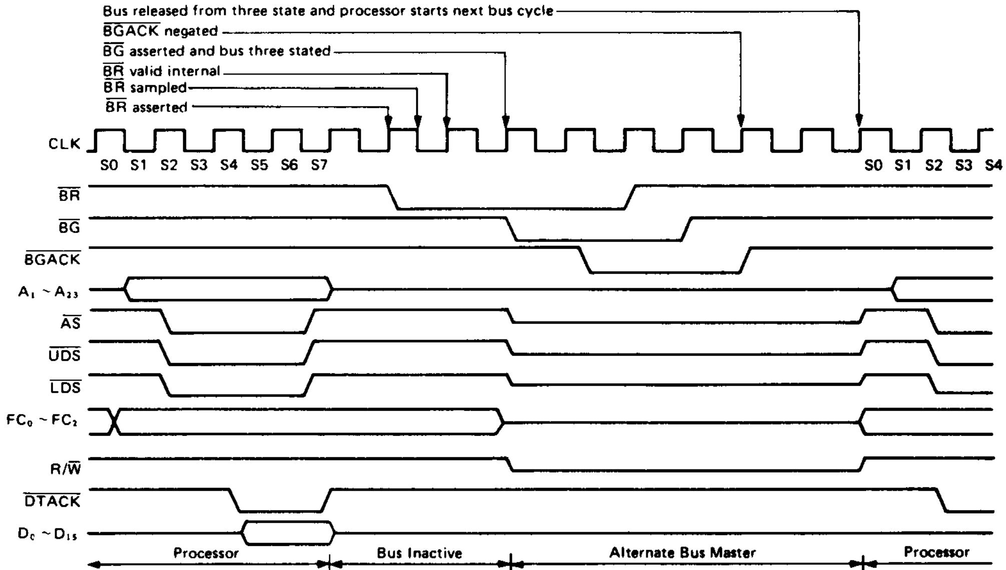


Figure 29 Bus Arbitration with Bus Inactive

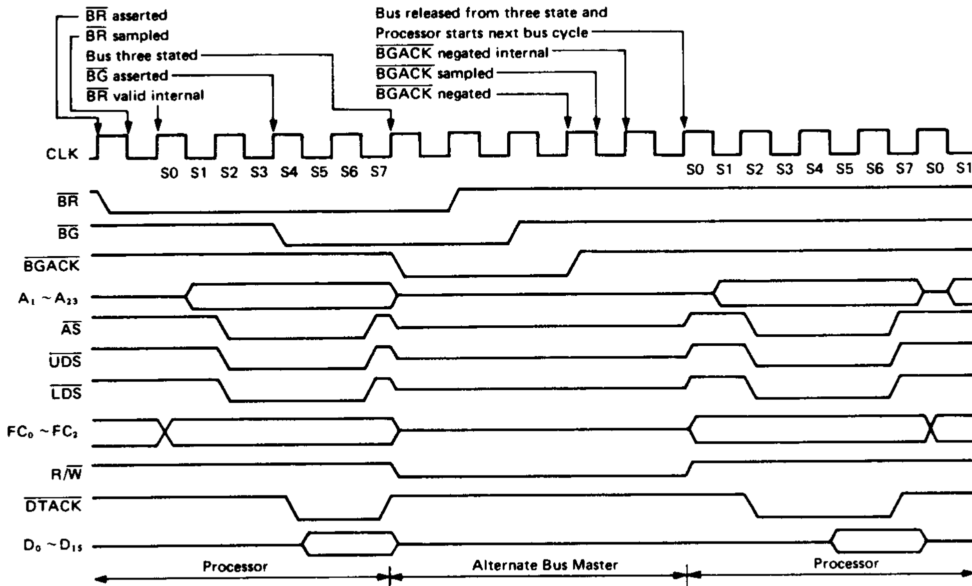


Figure 30 Bus Arbitration During Processor Bus Cycle Special Case

BUS ERROR AND HALT OPERATION

In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided. External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error signal is received, the processor has two options initiate a bus error exception sequence or try running the bus cycle again.

Exception Sequence

When the bus error signal is asserted, the current bus cycle is terminated. If BERR is asserted before the falling edge of S2, AS will be negated in S7 in either a read or write cycle. As long as BERR remains asserted, the data and address buses will be in the high-impedance state. When BERR is negated, the processor will begin stacking for exception processing. Figure 31 is a timing diagram for the exception sequence. The sequence is composed of the following elements.

(1) Stacking the program counter and status register
(2) Stacking the error information
(3) Reading the bus error vector table entry
(4) Executing the bus error handler routine

The stacking of the program counter and the status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address $000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to **EXCEPTION PROCESSING** for additional information.

Re-Running the Bus Cycle

When, during a bus cycle, the processor receives a bus error signal and the halt pin is being driven by an external device, the processor enters the re-run sequence. Figure 32 is a timing diagram for re-running the bus cycle.

The processor terminates the bus cycle, then puts the address and data output lines in the high-impedance state. The processor remains "halted," and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed at least one clock cycle before the halt signal is removed.

(NOTE) The processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a Test-and-Set operation is performed without ever releasing AS. If BERR and HALT are asserted during a read-modify-write bus cycle, a bus error operation results.

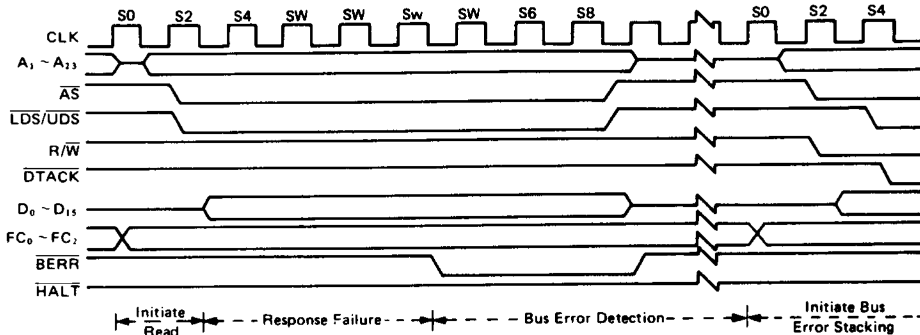


Figure 31 Bus Error Timing Diagram

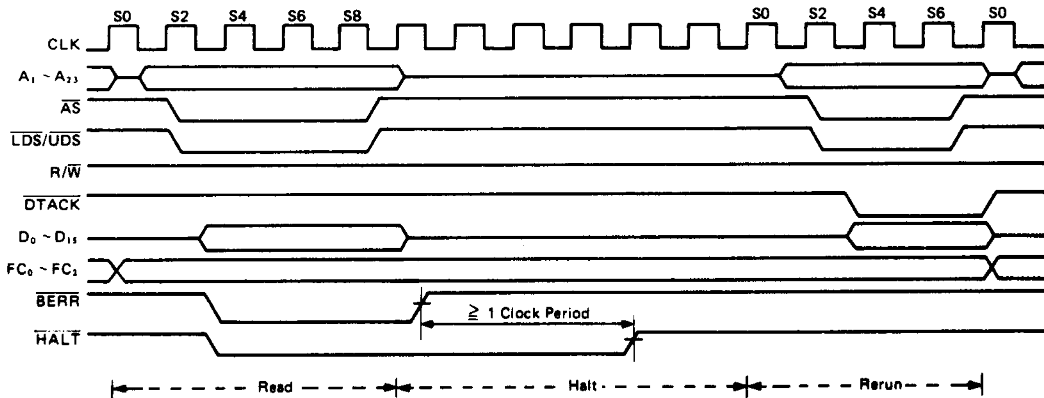


Figure 32 Re-Run Bus Cycle Timing Information

Halt Operation with No Bus Error

The halt input signal to the 68000 perform a Halt/Run/Single-Step function in a similar fashion to the HD6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does nothing) and when the halt signal is constantly inactive the processor "runs" (does something).

The single-step mode is derived from correctly timed transitions on the halt signal input. It forces the processor to execute a single bus cycle by entering the "run" mode until the processor starts a bus cycle then changing to the "halt" mode. Thus, the single-step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 33 details the timing required for correct single-step operations and Figure 34 shows a simple circuit for providing the single-step function. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

When the processor completes a bus cycle after recognizing that the halt signal is active, most three-state signals are put in the high-impedance state. These include:

(1) Address lines
(2) Data lines

This is required for correct performance of the re-run bus cycle operation.

While the processor is honoring the halt request, bus arbitration performs as usual. That is, halting has no effect on bus arbitration. It is the bus arbitration function that removes the control signals from the bus.

The halt function and the hardware trace capability allow the hardware debugger to trace single bus cycles or single instructions at a time. These processor capabilities, along with a software debugging package, give total debugging flexibility.

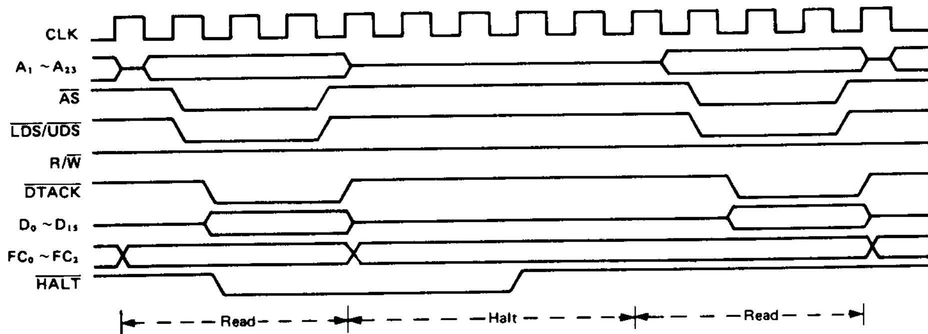


Figure 33 Halt Signal Timing Characteristics

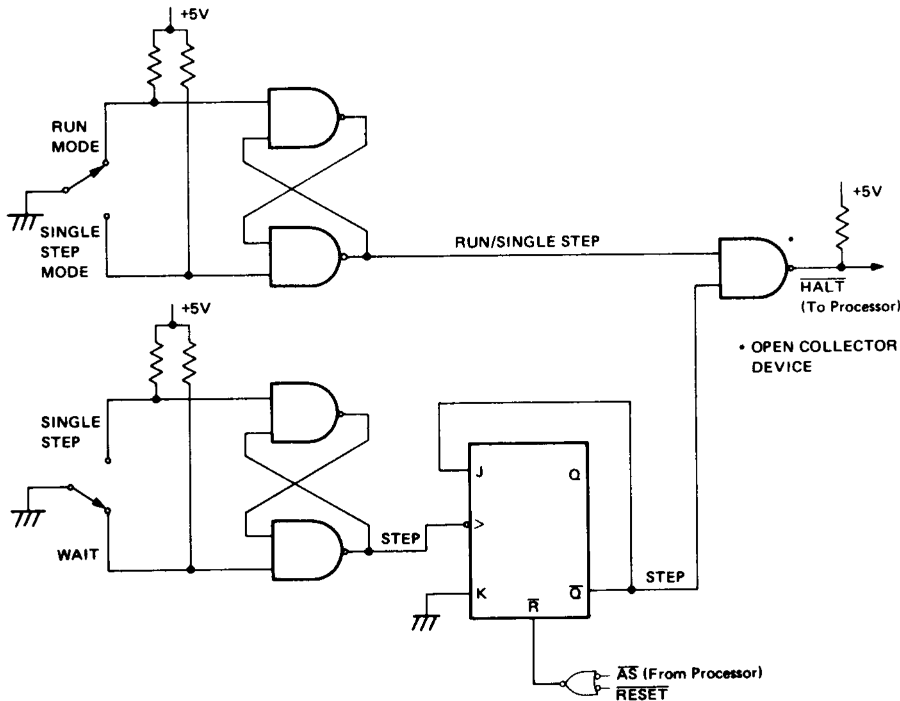


Figure 34 Simplified Single-Step Circuit

Double Bus Faults

When a bus error exception occurs, the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus errors in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs, the processor will halt. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

Note that a bus cycle which is re-run does not constitute a bus error exception, and does not contribute to a double bus fault. Note also that this means that as long as the external hardware requests it, the processor will continue to re-run the same bus cycle.

The bus error pin also has an effect on processor operation after the processor receives an external reset input. The processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed), the processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

RESET OPERATION

The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 35 is a timing diagram for the reset operations. Both the halt and reset lines must be asserted to ensure total reset of the processor.

When the reset and halt lines are driven by an external device, it is recognized as an entire system reset, including the processor. The processor responds by reading the reset vector table entry (vector unumber zero, address $000000) and loads it into the supervisor stack pointer (SSP). Vector table entry number one at address $000004 is read next and loaded into the program counter. The processor initializes the status register to an interrupt level of seven. No other registers are affected by the reset sequence.

When a RESET instruction is executed, the processor drives the reset pin for 124 clock periods. In this case, the processor is trying to reset the rest of the system. Therefore, there is no effect on the internal state of the processor. All of the processor's internal registers and the status register are unaffected by the execution of a RESET instruction. All external devices connected to the reset line should be reset at the completion of the RESET instruction.

Asserting the Reset and Halt pins for 10 clock cycles will cause a processor reset, except when V_{CC} is initially applied to the processor. In this case, an external reset must be applied for 100 milliseconds.

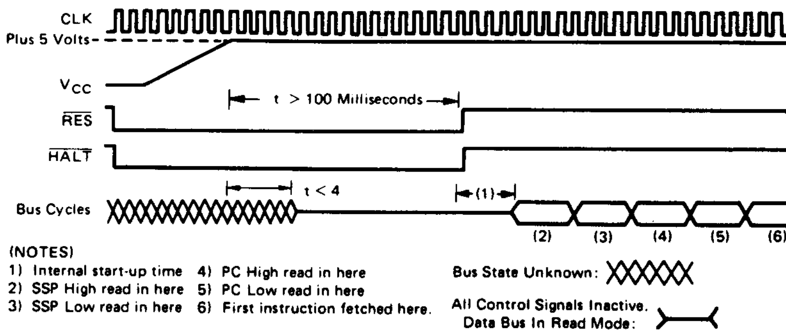


Figure 35 Reset Operation Timing Diagram

THE RELATIONSHIP OF $\overline{\mathrm{DTACK}}$, $\overline{\mathrm{BERR}}$, AND $\overline{\mathrm{HALT}}$

In order to properly control termination of a bus cycle for a re-run or a bus error condition, $\overline{\mathrm{DTACK}}$, $\overline{\mathrm{BERR}}$, and $\overline{\mathrm{HALT}}$ should be asserted and negated on the rising edge of the 68000 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the 68000. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 16):

Normal Termination: $\overline{\mathrm{DTACK}}$ occurs first (case 1).

Halt Termination: $\overline{\mathrm{HALT}}$ is asserted at the same time or before $\overline{\mathrm{DTACK}}$ and $\overline{\mathrm{BERR}}$ remains negated (cases 2 and 3).

Bus Error Termination: $\overline{\mathrm{BERR}}$ is asserted in lieu of, at the same time, or before $\overline{\mathrm{DTACK}}$ (case 4); $\overline{\mathrm{BERR}}$ is negated at the same time or after $\overline{\mathrm{DTACK}}$.

Re-Run Termination: $\overline{\mathrm{HALT}}$ and $\overline{\mathrm{BERR}}$ are asserted in lieu of, at the same time, or before $\overline{\mathrm{DTACK}}$ (cases 6 and 7); $\overline{\mathrm{HALT}}$ must be held at least one cycle after $\overline{\mathrm{BERR}}$. Case 5 indicates $\overline{\mathrm{BERR}}$ may precede $\overline{\mathrm{HALT}}$ which allows fully asynchronous assertion.

Table 16 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 17 ($\overline{\mathrm{DTACK}}$ is assumed to be negated normally in all cases; for best results, both $\overline{\mathrm{DTACK}}$ and $\overline{\mathrm{BERR}}$ should be negated when address strobe is negated.)

Example A: A system uses a watch-dog timer to terminate accesses to un-populated address space. The timer asserts $\overline{\mathrm{DTACK}}$ and $\overline{\mathrm{BERR}}$ simultaneously after time-out. (case 4)

Example B: A system uses error detection on RAM contents. Designer may (a) delay $\overline{\mathrm{DTACK}}$ until data verified, and return $\overline{\mathrm{BERR}}$ and $\overline{\mathrm{HALT}}$ simultaneously to re-run error cycle (case 6), or if valid, return $\overline{\mathrm{DTACK}}$; (b) delay $\overline{\mathrm{DTACK}}$ until data verified, and return $\overline{\mathrm{BERR}}$ at same time as $\overline{\mathrm{DTACK}}$ if data in error (case 4); (c) return $\overline{\mathrm{DTACK}}$ prior to data verification, as described in previous section. If data invalid, $\overline{\mathrm{BERR}}$ is asserted (case 1) in next cycle. Error-handling software must know how to recover error cycle.

Table 16 DTACK, BERR, HALT Assertion Results

Case No.	Control Signal	Asserted on Rising Edge of State		Result
		N	N + 2	
1	DTACK	A	S	Normal cycle terminate and continue.
	BERR	NA	X	
	HALT	NA	X	
2	DTACK	A	S	Normal cycle terminate and halt. Continue when HALT removed.
	BERR	NA	X	
	HALT	A	S	
3	DTACK	NA	A	Normal cycle terminate and halt. Continue when HALT removed.
	BERR	NA	NA	
	HALT	A	S	
4	DTACK	X	X	Terminate and take bus error trap.
	BERR	A	S	
	HALT	NA	NA	
5	DTACK	NA	X	Terminate and re-run.
	BERR	A	S	
	HALT	NA	A	
6	DTACK	X	X	Terminate and re-run when HALT removed.
	BERR	A	S	
	HALT	A	S	
7	DTACK	NA	X	Terminate and re-run when HALT removed.
	BERR	NA	A	
	HALT	A	S	

gend:
N – The number of the current even bus state (e.g., S4, S6, etc.)
A – Signal is asserted in this bus state
NA – Signal is not asserted in this state
X – Don't care
S – Signal was asserted in previous state and remains asserted in this state

Table 17 BERR and HALT Negation Results

Conditions of Termination in Table A	Control Signal	Negated on Rising Edge of State		Results – Next Cycle
		N	N + 2	
Bus Error	BERR	● or	●	Takes bus error trap.
	HALT	● or	●	
Re-run	BERR	● or	●	Illegal sequence; usually traps to vector number 0.
	HALT	●		
Re-run	BERR	●		Re-runs the bus cycle.
	HALT		●	
Normal	BERR	●		May lengthen next cycle.
	HALT	● or	●	
Normal	BERR		●	If next cycle is started it will be terminated as a bus error.
	HALT	● or	none	

ASYNCHRONOUS VERSUS SYNCHRONOUS OPERATION

Asynchronous Operation

To achieve clock frequency independence at a system level, the 68000 can be used in an asynchronous manner. This entails using only the bus handshake lines (AS, UDS, LDS, DTACK, BERR, HALT, and VPA) to control the data transfer. Using this method, AS signals the start of a bus cycle and the data strobes are used as a condition for valid data on a write cycle. The slave device (memory or peripheral) then responds by placing the requested data on the data bus for a read cycle or latching data on a write cycle and asserting the data transfer acknowledge signal (DTACK) to terminate the bus cycle. If no slave responds or the access is invalid, external control logic asserts the BERR, or BERR and HALT, signal to abort or re-run the bus cycle.

The DTACK signal is allowed to be asserted before the data from a slave device is valid on a read cycle. The length of time that DTACK may precede data is given as parameter #31 and it must be met in any asynchronous system to insure that valid data is latched into the processor. Notice that there is no maximum time specified from the assertion of AS to the assertion of DTACK. This is because the MPU will insert wait cycles of one clock period each until DTACK is recognized.

The BERR signal is allowed to be asserted after the DTACK signal is asserted. BERR must be asserted within the time given as parameter #48 after DTACK is asserted in any asynchronous

system to insure proper operation. If this maximum delay time is violated, the processor may exhibit erratic behavior.

Synchronous Operation

To allow for those systems which use the system clock as a signal to generate $\overline{DTACK}$ and other asynchronous inputs, the asynchronous input setup time is given as parameter #47. If this setup is met on an input, such as $\overline{DTACK}$, the processor is guaranteed to recognize that signal on the next falling edge of the system clock. However, the converse is not true – if the input signal does not meet the setup time it is not guaranteed not to be recognized. In addition, if $\overline{DTACK}$ is recognized on a falling edge, valid data will be latched into the processor (on a read cycle) on the next falling edge provided that the data meets the setup time given as parameter #27. Given this, parameter #31 may be ignored. Note that if $\overline{DTACK}$ is asserted, with the required setup time, before the falling edge of S4, no wait status will be incurred and the bus cycle will run at its maximum speed of four clock periods.

In order to assure proper operation in a synchronous system when $\overline{BERR}$ is asserted after $\overline{DTACK}$, $\overline{BERR}$ must meet the setup time parameter #27A prior to the falling edge of the clock one clock cycle after $\overline{DTACK}$ was recognized. This setup time is critical to proper operation, and the HD68000 may exhibit erratic behavior if it is violated.

(NOTE)

During an active bus cycle, $\overline{VPA}$ and $\overline{BERR}$ are sampled on every falling edge of the clock starting with S0. $\overline{DTACK}$ is sampled on every falling edge of the clock starting with S4 and data is latched on the falling edge of S6 during a read. The bus cycle will then be terminated in S7 except when $\overline{BERR}$ is asserted in the absence of $\overline{DTACK}$, in which case it will terminate one clock cycle later in S9.

■ PROCESSING STATES

This section describes the actions the 68000 which are outside the normal processing associated with the execution of instructions. The functions of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions is detailed.

The 68000 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a STOP instruction is executed. In this state, no further memory references are made.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

PROCESSING STATES

NORMAL	INSTRUCTION EXECUTION (INCLUDING STOP)
EXCEPTION	INTERRUPTS TRAPS TRACING ETC.
HALTED	HARDWARE HALT DOUBLE BUS FAULT

● PRIVILEGE STATES

The processor operates in one of two states of privilege: the "user" state or the "supervisor" state. The privilege state determines which operations are legal, are used to choose between the supervisor stack pointer and the user stack pointer in instruction references, and may be used by an external memory management device to control and translate accesses.

The privileges state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas, and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user state programs.

SUPERVISOR STATE

The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S-bit of the status register; if the S-bit is asserted (high), the processor is in the supervisor state. All instructions can be executed in the supervisor state. The bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the setting of the S-bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

USER STATE

The user state is the lower state of privilege. For instruction execution, the user state is determined by the S-bit of the status register; if the S-bit is negated (low), the processor is executing instructions in the user state.

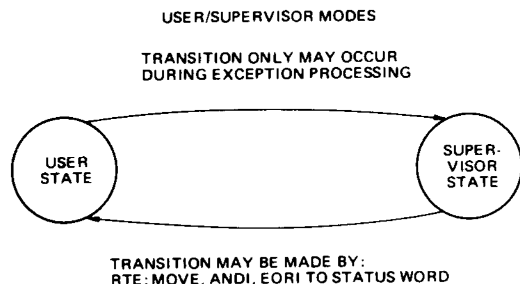
Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the STOP instruction, or the

RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole status register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE to USP) and move from user stack pointer (MOVE from USP) instructions are also privileged.

The bus cycles generated by an instruction executed in user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly, or address register seven explicitly, access the use stack pointer.

PRIVILEGE STATE CHANGES

Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current setting of the S-bit of the status register is saved and the S-bit is asserted, putting the processing in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.



REFERENCE CLASSIFICATION

When the processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge. Table 18 lists the classification of references.

Table 18 Reference Classification

Function Code Output			Reference Class
FC_2	FC_1	FC_0	
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

• EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made, and the status register is set for exception processing. In the second step the exception vector is determined, and the third step is the saving of the current processor context. In the fourth step a new context is obtained, and the processor switches to instruction processing.

EXCEPTION VECTORS

Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 36), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an eight-bit number which, when multiplied by four, gives the address of an exception vector. Vector numbers are generated internally or externally depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 37) to the processor on data bus lines D_0 through D_7. The processor translates the vector number into a full 24-bit address, as shown in Figure 38. The memory layout for exception vectors is given in Table 19.

As shown in Table 19, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

KINDS OF EXCEPTIONS

Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts and the bus error and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from address error or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK) and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses and privilege violations cause exceptions. Tracing behaves like a very high priority, internally generated interrupt after each instruction execution.

EXCEPTION PROCESSING SEQUENCE

Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S-bit is asserted, putting the processor into the supervisor privilege state. Also, the T-bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

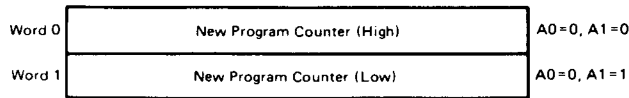
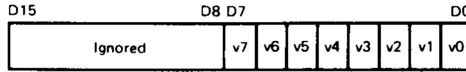


Figure 36 Exception Vector Format



Where:
v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

Figure 37 Peripheral Vector Number Format

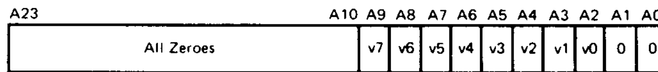


Figure 38 Address Translated From 8-Bit Vector Number

Table 19 Exception Vector Assignment

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
–	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16 ~ 23*	64	040	SD	(Unassigned, reserved)
	95	05F		
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32 ~ 47	128	080	SD	TRAP Instruction Vectors
	191	0BF		
48 ~ 63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		
64 ~ 255	256	100	SD	User Interrupt Vectors
	1023	3FF		

SP: Supervisor program, SD: Supervisor data

* Vector numbers 12, 13, 14, 16 through 23 and 48 through 63 are reserved for future enhancements by Hitachi. No user peripheral devices should be assigned these numbers.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer as shown in Figure 39. The program counter value stacked usually points to the next unexecuted instruction, however for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. Then instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

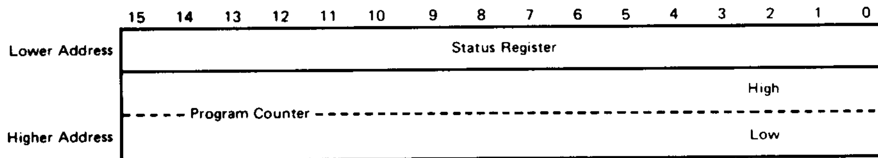


Figure 39 Exception Stack Order (Group 1, 2)

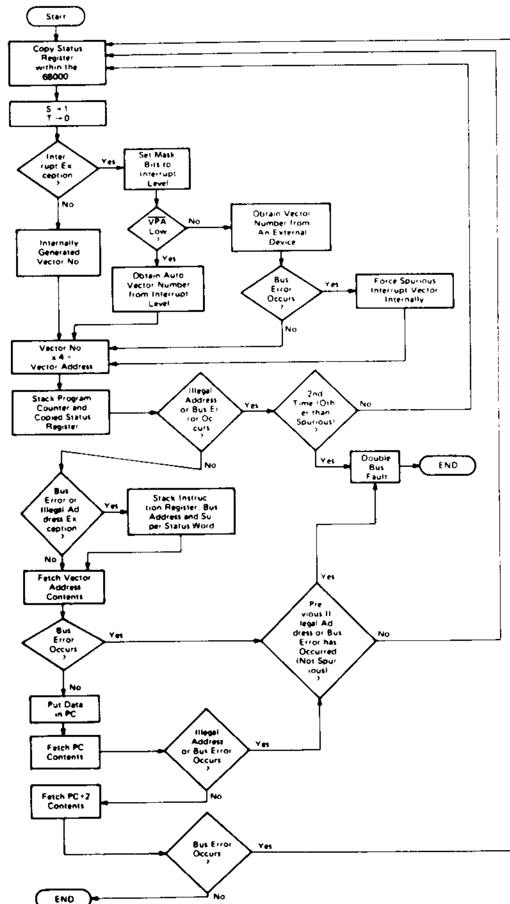


Figure 40 Exception Processing Sequence (Not Reset)

MULTIPLE EXCEPTIONS

These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The Group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted, and the exeception processing to commence within two clock cycles. The Group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The Group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while Group 2 exceptions have lowest priority. Within Group 0, reset has highest priority, followed by address error and then bus error. Within Group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within Group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit is asserted, the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 20.

Table 20 Exception Grouping and Priority

Group	Exception	Processing
0	Reset Address Error Bus Error	Exception processing begins within two clock cycles.
1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction
2	TRAP, TRAPV CHK, Zero Divide	Exception processing is started by normal instruction execution

RECOGNITION TIMES OF EXCEPTIONS, HALT, AND BUS ARBITRATION

END OF A CLOCK CYCLE
- RESET

END OF A BUS CYCLE
- ADDRESS ERROR
- BUS ERROR
- HALT
- BUS ARBITRATION

END OF AN INSTRUCTION CYCLE
- TRACE EXCEPTION
- INTERRUPT EXCEPTIONS
- ILLEGAL INSTRUCTION
- UNIMPLEMENTED INSTRUCTION
- PRIVILEGE VIOLATION

WITHIN AN INSTRUCTION CYCLE
- TRAP, TRAPV
- CHK
- ZERO DIVIDE

• EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources, and each exception has processing which is peculiar to it. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

RESET

The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, and the trace state is forced off. The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The power-up/restart code should be pointed to by the initial program counter.

The RESET instruction does not cause loading of the reset vector, but does assert the reset line to reset external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

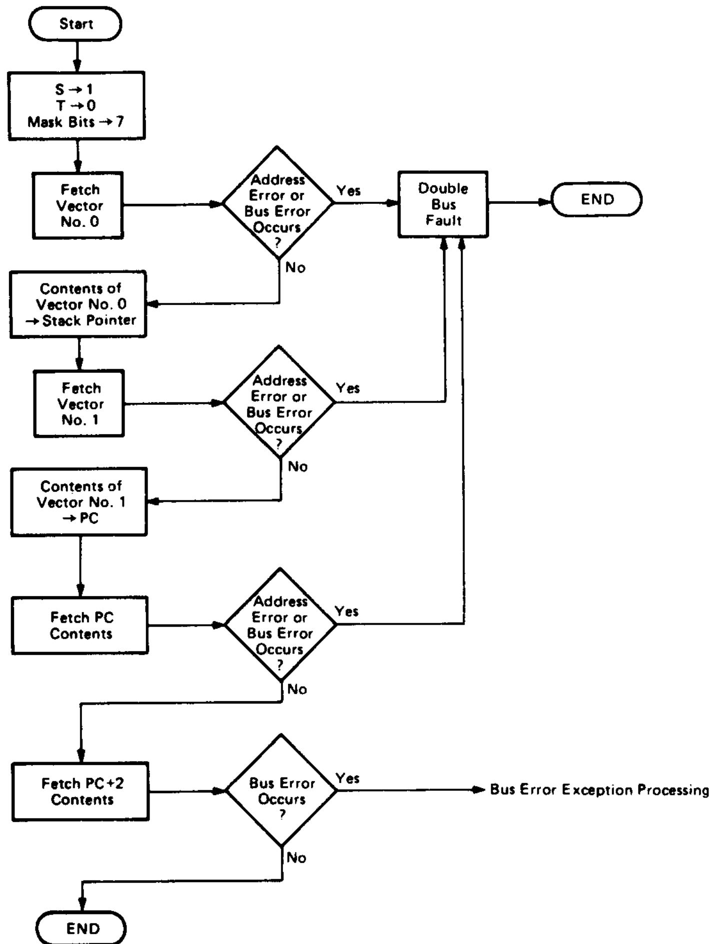


Figure 41 Reset Exception Processing

INTERRUPTS

Seven levels of interrupt priorities are provided. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels are numbered from one to seven, with level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority, and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained in a following paragraph.)

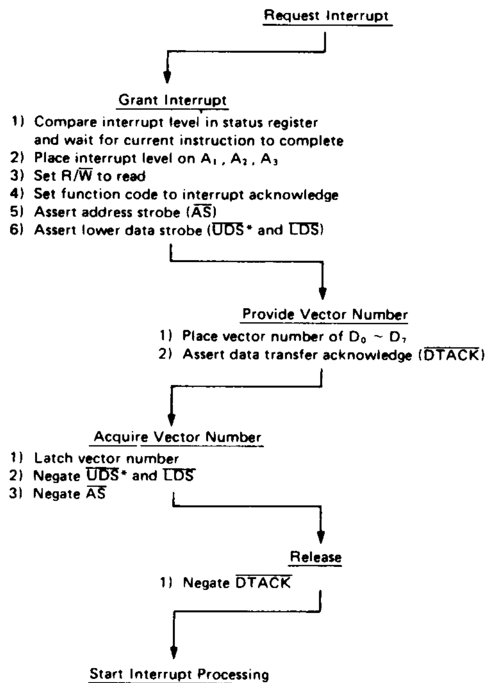
If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. First a copy of the status register is saved, and the privilege state is set to supervisor, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of

the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flow chart for the interrupt acknowledge sequence is given in Figure 42, a timing diagram is given in Figure 43, and the interrupt exception timing sequence is shown in Figure 44.

Table 21 Internal Interrupt Level

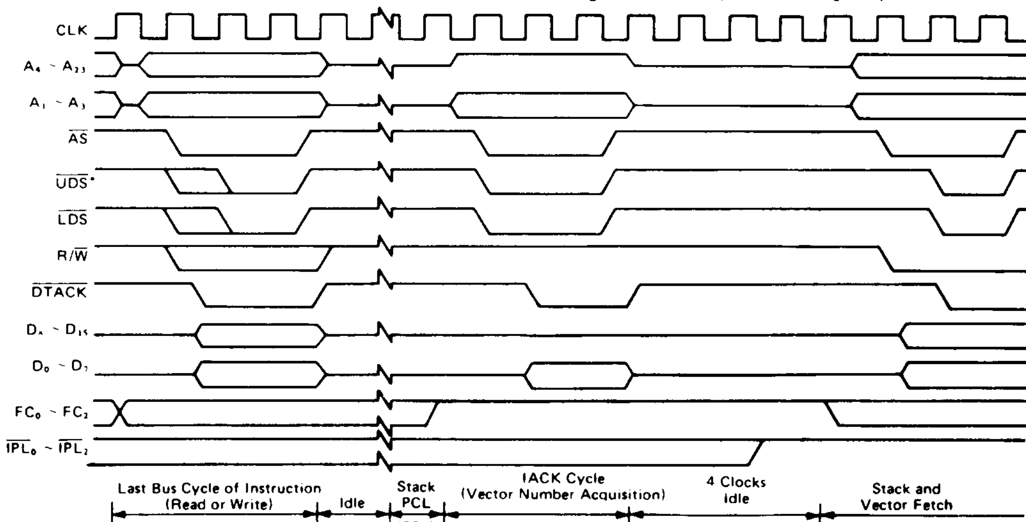
Level	I2	I1	I0	Interrupt
7	1	1	1	Non-Maskable Interrupt
6	1	1	0	Maskable Interrupt
5	1	0	1	
4	1	0	0	
3	0	1	1	
2	0	1	0	
1	0	0	1	
0	0	0	0	No Interrupt

(NOTE) The internal interrupt mask level (I2, I1, I0) are inverted to the logic level applied to the pins ($\overline{IPL_2}$, $\overline{IPL_1}$, $\overline{IPL_0}$).



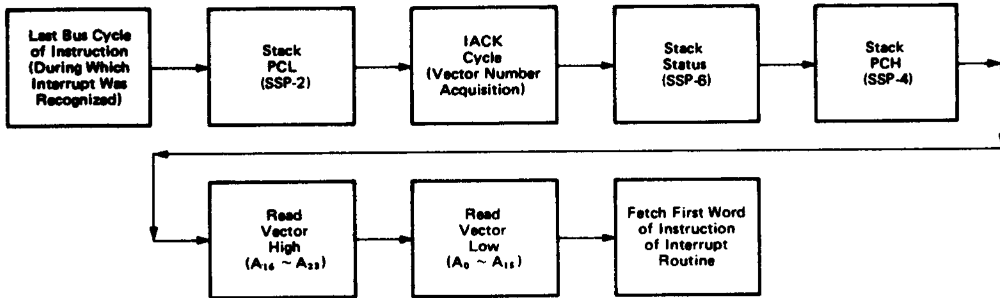
* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D_8 through D_{15} at this time.

Figure 42 Interrupt Acknowledge Sequence Flow Chart



* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D_8 through D_{15} at this time.

Figure 43 Interrupt Acknowledge Sequence Timing Diagram



Note: SSP refers to the value of the supervisor stack pointer before the interrupt occurs.

Figure 44 Interrupt Exception Timing Sequence

Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

UNINITIALIZED INTERRUPT

An interrupting device asserts $\overline{VPA}$ or provides an interrupt vector during an interrupt acknowledge cycle to the 68000.
If the vector register has not been initialized, the responding HD68000 Family peripheral will provide vector 15, the unitialized interrupt vector. This provides a uniform way to recover from a programming error.

SPURIOUS INTERRUPT

If during the interrupt acknowledge cycle no device responds by asserting $\overline{DTACK}$ or $\overline{VPA}$, the bus error line should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

INSTRUCTION TRAPS

Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception, and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow or a subscript out of bounds.

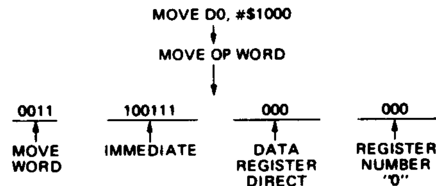
The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

ILLEGAL AND UNIMPLEMENTED INSTRUCTIONS

Illegal instruction is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs.

Word patterns with bits 15 through 12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

ILLEGAL INSTRUCTION EXAMPLE



PRIVILEGE VIOLATIONS

In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an exception. The privileged instruction are:

STOP	AND (word) Immediate to SR
RESET	EOR (word) Immediate to SR
RTE	OR (word) Immediate to SR
MOVE to SR	MOVE USP

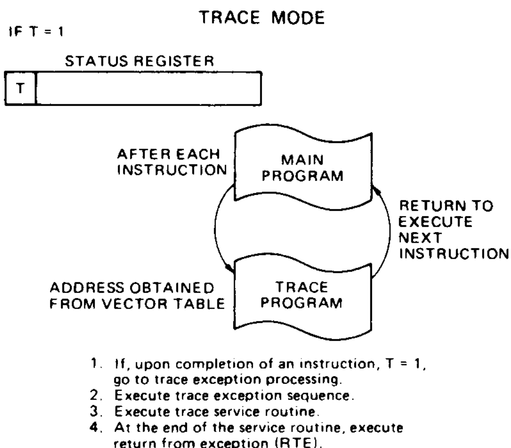
TRACING

To aid in program development, the 68000 includes a facility to allow instruction by instruction tracing. In the trace state, after each instruction is executed an exceptions is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T-bit in the supervisor portion of the status register. If the T-bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T-bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed. either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus

error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction, an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.



1. If, upon completion of an instruction, T = 1, go to trace exception processing.
2. Execute trace exception sequence.
3. Execute trace service routine.
4. At the end of the service routine, execute return from exception (RTE).

BUS ERROR

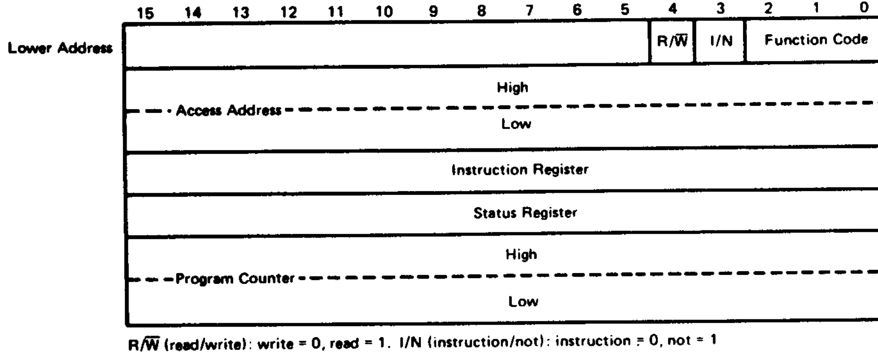
Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus error exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, one to five words beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed, and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or a write, whether the processor was processing an instruction or not, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a Group 2 exception; the processor is not processing an instruction if it is processing a Group 0 or a Group 1 exception. Figure 45 illustrates how this information is organized on the supervisor stack. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted, and all processing cases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroy all memory contents. Only the $\overline{RES}$ pin can restart a halted processor.

ADDRESS ERROR

Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in Figure 46, an address error will execute a short bus cycle followed by exception processing.



R/W̄ (read/write): write = 0, read = 1. I/N (instruction/not): instruction = 0, not = 1

Figure 45 Exception Stack Order (Group 0)

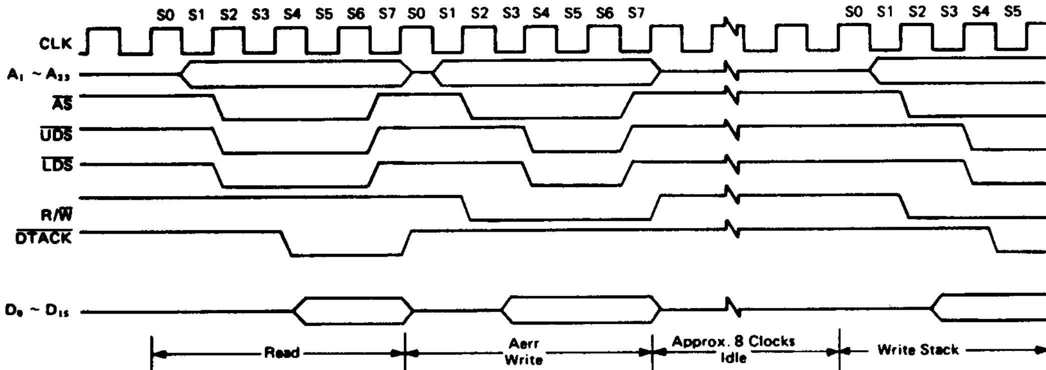


Figure 46 Address Error Timing

■ INTERFACE WITH HD6800 PERIPHERALS

Hitachi's extensive line of HD6800 peripherals are directly compatible with the 68000. Some of these devices that are particularly useful are:

HD6821	Peripheral Interface Adapter
HD6840	Programmable Timer Module
HD6843	Floppy Disk Controller
HD6845S	CRT Controller
HD46508	Analog Data Acquisition Unit
HD6850	Asynchronous Communication Interface Adapter
HD6852	Synchronous Serial Data Adapter

To interface the synchronous HD6800 peripherals with the asynchronous 68000, the processor modifies its bus cycle to meet the HD6800 cycle requirements whenever an HD6800 device address is detected. This is possible since both processors use memory mapped I/O. Figure 48 is a flow chart of the interference operation between the processor and HD6800 devices.

• DATA TRANSFER OPERATION

Three signals on the processor provide the HD6800 interface. They are enable (E), valid memory address (VMA), and valid peripheral address (VPA). Enable corresponds to the E or ϕ_2 signal in existing HD6800 systems. The bus frequency is one tenth of the incoming 68000 clock frequency. The timing of E allows 1 MHz peripherals to be used with an 8 MHz 68000. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do successive VPA accesses on successive E pulses.

HD6800 cycle timing is given in Figures 49 and 50. At state zero (S0) in the cycle, the address bus is in the high-impedance state. A function code is asserted on the function code output lines. One-half clock later, in state 1 the address bus is released from the high-impedance state.

During state 2, the address strobe (AS) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle, the upper and/or lower data strobes are also asserted in state 2. If the bus cycle is a write cycle,

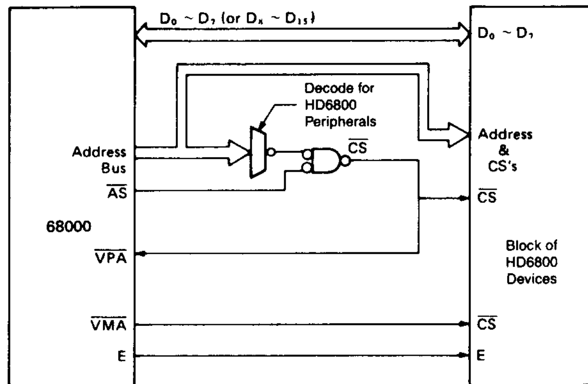


Figure 47 Connection of HD6800 Peripherals

the read/write (R/$\overline{W}$) signal is switched to low (write) during state 2. One half clock later, in state 3, the write data is placed on the data bus, and in state 4 the data strobes are issued to indicate valid data on the data bus. The processor now inserts wait states until it recognizes the assertion of $\overline{VPA}$.

The $\overline{VPA}$ input signals the processor that the address on the bus is the address of an HD6800 device (or an area reserved for HD6800 devices) and that the bus should conform to the ϕ_2 transfer characteristics of the HD6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by address strobe. Chip select for the HD6800 peripherals should be derived by decoding the address bus conditioned by $\overline{VMA}$.

After the recognition of $\overline{VPA}$, the processor assures that the Enable (E) is low, by waiting if necessary, and subsequently asserts $\overline{VMA}$. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the HD6800 peripherals are selected and deselected at the correct time. The peripheral now runs in cycle during the high portion of the E signal. Figures 49 and 50 depict the best and worst case HD6800 cycle timing. This cycle length is dependent strictly upon when $\overline{VPA}$ is asserted in relationship to the E clock.
dependent strictly upon when $\overline{VPA}$ is asserted in relationship to the E clock.

If we assume that external circuitry asserts $\overline{VPA}$ as soon as possible after the assertion of $\overline{AS}$, then $\overline{VPA}$ will be recognized as being asserted on the falling edge of S4. In this case, no "extra" wait cycles will be inserted prior to the recognition of $\overline{VPA}$ asserted and only the wait cycles inserted to synchronize with the E clock will determine the total length of the cycle. In any case, the synchronization delay will be some integral number of clock cycles within the following two extremes:

1. Best Case – $\overline{VPA}$ is recognized as being asserted on the falling edge three clock cycles before E rises (or three clock cycles after E falls).
2. Worst Case -- $\overline{VPA}$ is recognized as being asserted on the falling edge two clock cycles before E rises(or four clock cycles after E falls).

During a read cycle, the processor latches the peripheral data in state 6. For all cycles, the processor negates the address and data strobes one half clock cycle later in state 7, and the Enable signal goes low at this time. Another half clock later, the address bus is put in the high-impedance state. During a write cycle, the data bus is put in the high-impedance state

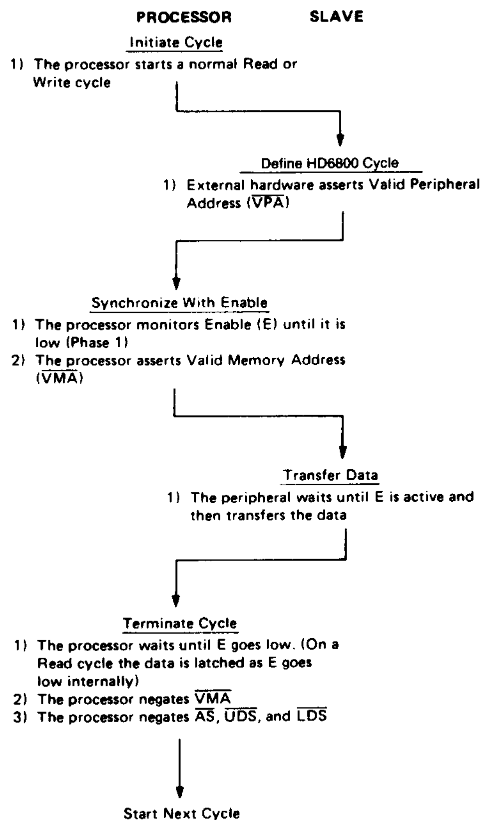


Figure 48 HD6800 Interface Flow Chart

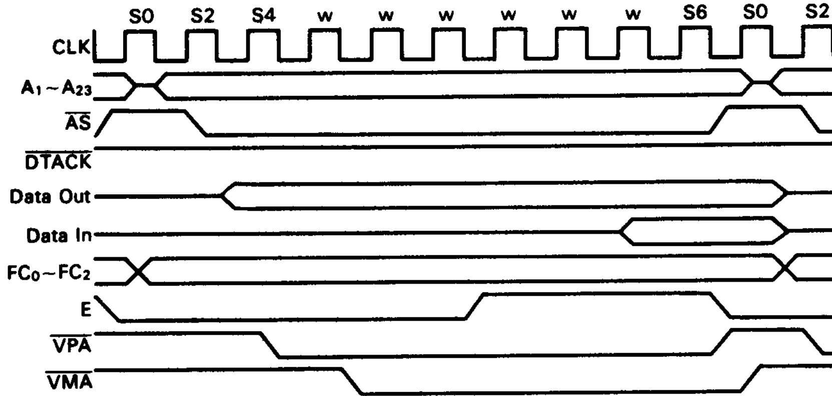


Figure 49 68000 to HD6800 Peripheral Timing—Best Case

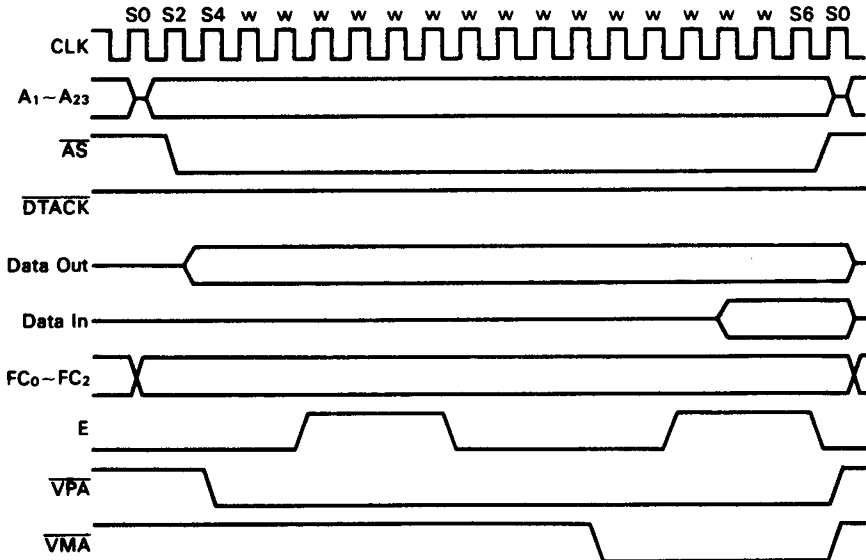


Figure 50 68000 to HD6800 Peripheral Timing—Worst Case

HMCS6800 E Clock Freq.	Type
2.0 MHz	B
1.5 MHz	A
1.0 MHz	Std

* Times are expressed for different device clock frequencies.

Figure 51 68000 to HD6800 Peripheral Timing Diagram

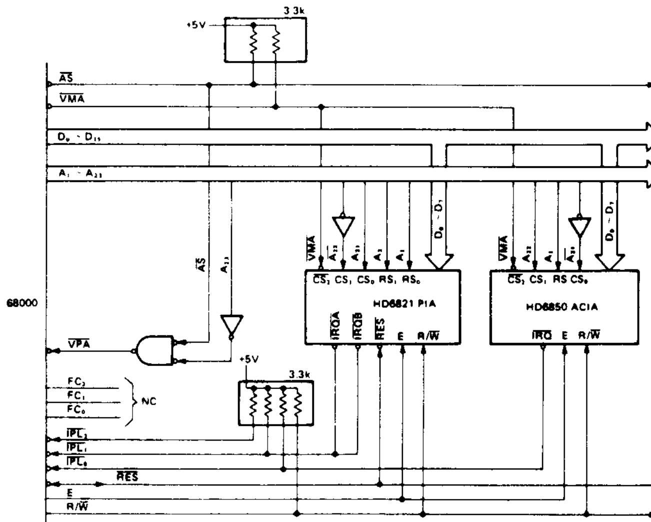


Figure 52 HD6800 Interface—Example 1

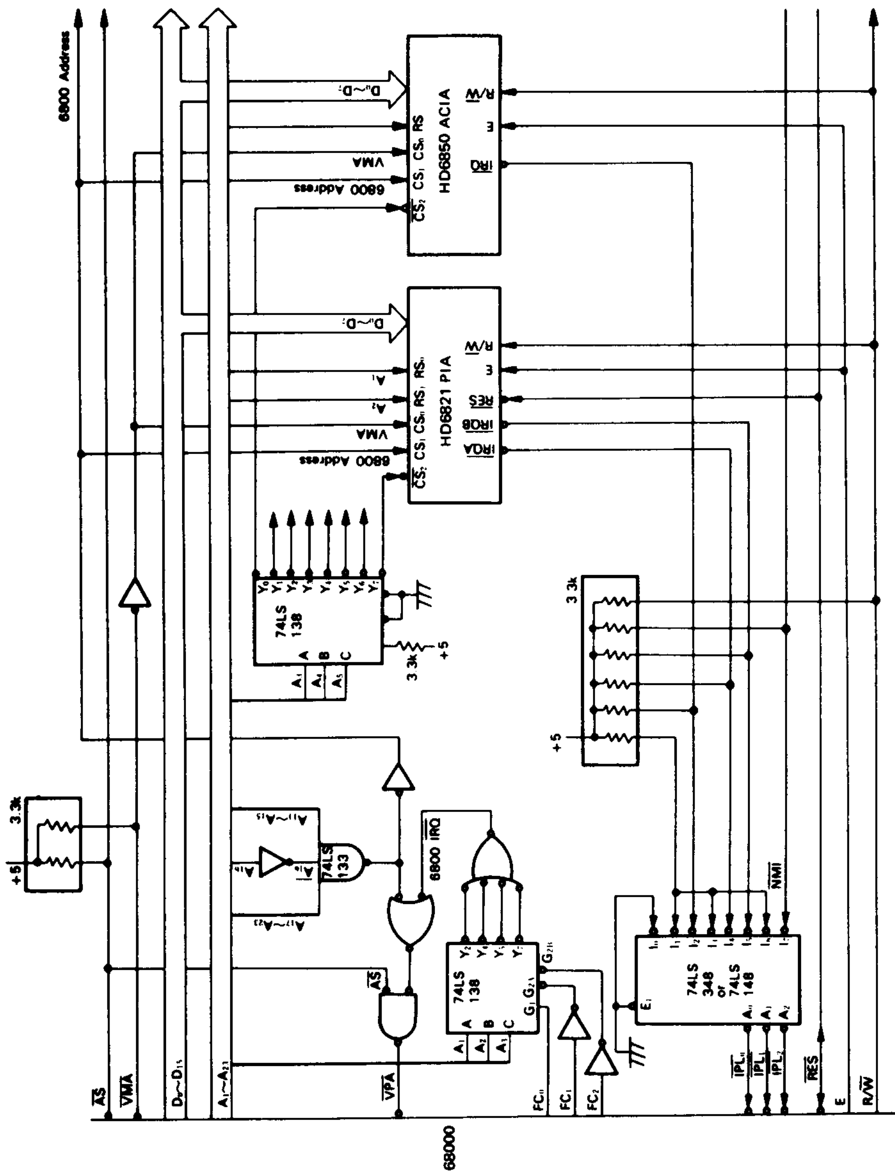


Figure 53 HD6800 Interface—Example 2

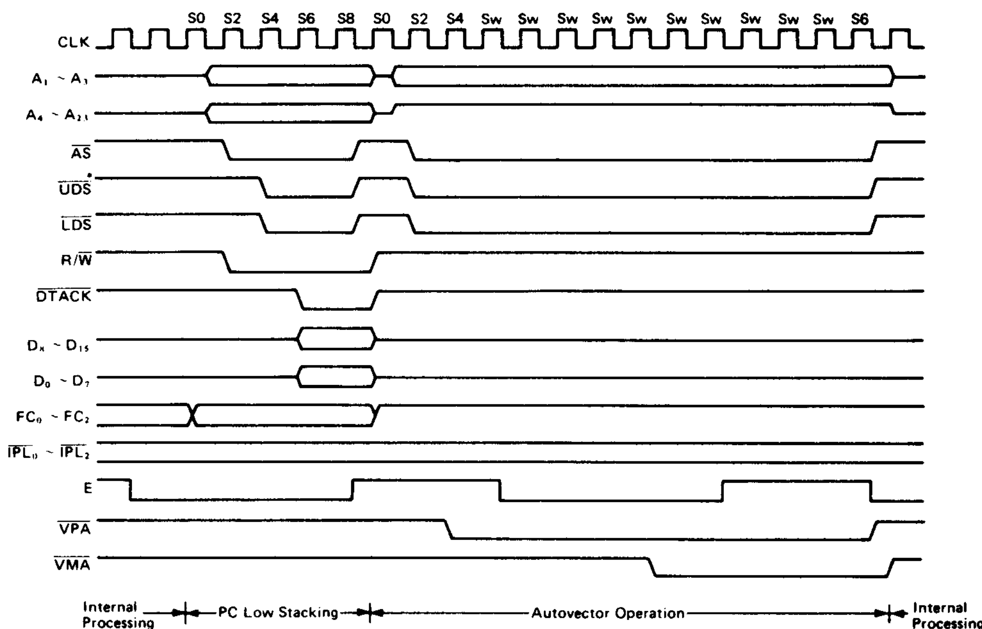
and the read/write signal is switched high. The peripheral logic must remove $\overline{VPA}$ within one clock after address strobe is negated.

Figure 51 shows the timing required by HD6800 peripherals, the timing specified for HD6800, and the corresponding timing for the 68000. Two example systems with HD6800 peripherals are shown in Figures 52 and 53. The system in Figure 52 reserves the upper eight megabytes of memory for HD6800 peripherals. The system in Figure 53 is more efficient with memory and easily expandable, but more complex.

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Notice that the 68000 $\overline{VMA}$ is active low, contrasted with the active high HD6800 VMA. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting peripherals.

• INTERRUPT OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, if $\overline{VPA}$ is asserted, the 68000 will assert $\overline{VMA}$ and complete a normal HD6800 read cycle as shown in Figure 54. The processor will then use an internally generated



* Although a vector number is one byte, both data strobes are asserted due to the microcode used for exception processing. The processor does not recognize anything on data lines D_8 through D_{15} at this time.

Figure 54 Autovector Operation Timing Diagram

vector that is a function of the interrupt being serviced. This process is known as autovectoring. The seven autovectors are vector numbers 25 through 31 (decimal).

This operates in the same fashion (but is not restricted to) HD6800 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the HD6800 and the 68000's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted autovectoring, the HD6800 peripheral address decoding should prevent unintended accesses.

■ CONDITION CODES COMPUTATION

This provides a discussion of how the condition codes were developed, the meanings of each bit, how they are computed, and how they are represented in the instruction set details.

• CONDITION CODE REGISTER

The condition code register portion of the status register contains five bits:

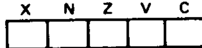
N – Negative
Z – Zero
V – Overflow
C – Carry
X – Extend

The first four bits are true condition code bits in that they reflect the condition of the result of a processor operation. The X-bit is an operand for multiprecision computations. The carry bit (C) and the multiprecision operand extend bit (X) are separate in the 68000 to simplify the programming model.

- **CONDITION CODE REGISTER NOTATION**

In the instruction set details, the description of the effect on the condition codes is given in the following form:

Condition Codes:



Where

N (negative) set if the most significant bit of the result is set. Cleared otherwise.

Z (zero) set if the result equals zero. Cleared otherwise.

V (overflow) set if there was an arithmetic overflow. This implies that the result is not representable in the operand size. Cleared otherwise.

C (carry) set if a carry is generated out of the most significant bit of the operands for an addition. Also set if a borrow is generated in a subtraction. Cleared otherwise.

X (extend) transparent to data movement. When affected, it is set the same as the C-bit.

The notational convention that appears in the representation of the condition code registers is:

- * set according to the result of the operation
- – not affected by the operation
- 0 cleared
- 1 set
- U undefined after the operation

- **CONDITION CODE COMPUTATION**

Most operations take a source operand and a destination operand, compute, and store the result in the destination location. Unary operations take a destination operand, compute, and store the result in the destination location. Table 22 details how each instruction sets the condition codes.

Table 22 Condition Code Computations

Operations	X	N	Z	V	C	Special Definition
ABCD	*	U	?	U	?	C = Decimal Carry $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
ADD, ADDI, ADDQ	*	*	*	?	?	$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + \overline{Rm} \cdot Dm + Sm \cdot \overline{Rm}$
ADDX	*	*	?	?	?	$V = Sm \cdot Dm \cdot \overline{Rm} + \overline{Sm} \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot Dm + \overline{Rm} \cdot Dm + Sm \cdot \overline{Rm}$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
AND, ANDI, EOR, EORI, MOVEQ, MOVE, OR, ORI, CLR, EXT, NOT, TAS, TST	–	*	*	0	0	
CHK	–	*	U	U	U	
SUB, SUBI SUBQ	*	*	*	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$
SUBX	*	*	?	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
CMP, CMPI, CMPM	–	*	*	?	?	$V = \overline{Sm} \cdot Dm \cdot \overline{Rm} + Sm \cdot \overline{Dm} \cdot Rm$ $C = Sm \cdot \overline{Dm} + Rm \cdot \overline{Dm} + Sm \cdot Rm$
DIVS, DIVU	–	*	*	?	0	V = Division Overflow
MULS, MULU	–	*	*	0	0	
SBCD, NBCD	*	U	?	U	?	C = Decimal Borrow $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
NEG	*	*	*	?	?	$V = Dm \cdot Rm, C = Dm + Rm$
NEGX	*	*	?	?	?	$V = Dm \cdot Rm, C = Dm + Rm$ $Z = Z \cdot \overline{Rm} \cdot \ldots \cdot \overline{R0}$
BTST, BCHG, BSET, BCLR	–	–	?	–	–	$Z = \overline{Dn}$
ASL	*	*	*	?	?	$V = Dm \cdot (\overline{D_{m-1}} + \ldots + \overline{D_{m-r}}) + \overline{Dm} \cdot (D_{m-1} + \ldots + D_{m-r})$ $C = D_{m-r+1}$
ASL (r = 0)	–	*	*	0	0	
LSL, ROXL	*	*	*	0	?	$C = D_{m-r+1}$
LSR (r = 0)	–	*	*	0	0	
ROXL (r = 0)	–	*	*	0	?	C = X
ROL	–	*	*	0	?	$C = D_{m-r+1}$
ROL (r = 0)	–	*	*	0	0	
ASR, LSR, ROXR	*	*	*	0	?	$C = D_{r-1}$
ASR, LSR (r = 0)	–	*	*	0	0	
ROXR (r = 0)	–	*	*	0	?	C = X
ROR	–	*	*	0	?	$C = D_{r-1}$
ROR (r = 0)	–	*	*	0	0	

– Not affected
U Undefined
? Other— see Special Definition

* General Case:
X = C
N = Rm
$Z = \overline{Rm} \cdot \ldots \cdot \overline{R0}$

Sm – Source operand most significant bit
Dm – Destination operand most significant bit
Rm – Result bit most significant bit
n – bit number
r – shift amount

• CONDITIONAL TESTS

Table 23 lists the condition names, encodings, and tests for the conditional branch and set instructions. The test associated with each condition is a logical formula based on the current state of the condition codes. If this formula evaluates to 1, the condition succeeds, or is true. If the formula evaluates to 0, the condition is unsuccessful, or false. For example, the T condition always succeeds, while the EQ condition succeeds only if the Z bit is currently set in the condition codes.

Table 23 Conditional Tests

Mnemonic	Condition	Encoding	Test
T	true	0000	1
F	false	0001	0
HI	high	0010	$\overline{C} \cdot \overline{Z}$
LS	low or same	0011	$C + Z$
CC	carry clear	0100	$\overline{C}$
CS	carry set	0101	C
NE	not equal	0110	$\overline{Z}$
EQ	equal	0111	Z
VC	overflow clear	1000	$\overline{V}$
VS	overflow set	1001	V
PL	plus	1010	$\overline{N}$
MI	minus	1011	N
GE	greater or equal	1100	$N \cdot V + \overline{N} \cdot \overline{V}$
LT	less than	1101	$N \cdot \overline{V} + \overline{N} \cdot V$
GT	greater than	1110	$N \cdot V \cdot \overline{Z} + \overline{N} \cdot \overline{V} \cdot \overline{Z}$
LE	less or equal	1111	$Z + N \cdot \overline{V} + \overline{N} \cdot V$

■ INSTRUCTION SET

The following paragraphs provide information about the addressing categories and instruction set of the 68000.

• ADDRESSING CATEGORIES

Effective address modes may be categorized by the ways in which they may used. The following classifications will be used in the instruction definitions.

- Data — If an effective address mode may be used to refer to data operands, it is considered a data addressing effective address mode.
- Memory — If an effective address mode may be used to refer to memory operands, it is considered a memory addressing effective address mode.
- Alterable — If an effective address mode may be used to refer to alterable (writeable) operands, it is considered an alterable addressing effective address mode.
- Control — If an effective address mode may be used to refer to memory operands without an associated size, it is considered a control addressing effective address mode.

Table 24 shows the various categories to which each of the effective address modes belong. Table 25 is the instruction set summary.

The status register addressing mode is not permitted unless it is explicitly mentioned as a legal addressing mode.

These categories may be combined so that additional, more restrictive, classifications may be defined. For example, the instruction descriptions use such classifications as alterable memory or data alterable. The former refers to those addressing modes which are both alterable and memory addresses, and the latter refers to addressing modes which are both data and alterable.

• INSTRUCTION PRE-FETCH

The 68000 uses a 2-word tightly-coupled instruction prefetch mechanism to enhance performance. This mechanism is described in terms of the microcode operations involved. If the execution of an instruction is defined to begin when the microroutine for that instruction is entered, some features of the prefetch mechanism can be described.

1) When execution of an instruction begins, the operation word and the word following have already been fetched. The operation word is in the instruction decoder.
2) In the case of multi-word instructions, as each additional word of the instruction is used internally, a fetch is made to the instruction stream to replace it.
3) The last fetch from the instruction stream is made when the operation word is discarded and decoding is started on the next instruction.
4) If the instruction is a single-word instruction causing a branch, the second word is not used. But because this word is fetched by the preceding instruction, it is impossible to avoid this superfluous fetch. In the case of an interrupt or trace exception, both words are not used.
5) The program counter usually points to the last word fetched from the instruction stream.

Table 24 Effective Addressing Mode Categories

Effective Address Modes	Mode	Register	Data	Addressing Categories		
				Memory	Control	Alterable
Dn	000	register number	X	–	–	X
An	001	register number	–	–	–	X
An@	010	register number	X	X	X	X
An@+	011	register number	X	X	–	X
An@-	100	register number	X	X	–	X
An@(d)	101	register number	X	X	X	X
An@(d, ix)	110	register number	X	X	X	X
xxx.W	111	000	X	X	X	X
xxx.L	111	001	X	X	X	X
PC@(d)	111	010	X	X	X	–
PC@(d, ix)	111	011	X	X	X	–
#xxx	111	100	X	X	–	–

The following example illustrates many of the features of instruction prefetch. The contents of memory are assumed to be as illustrated in Figure 55.

```
            ORG      0                       DEFINE RESTART VECTOR
            DC.L     INISSP                  INITIAL SYSTEM STACK POINTER
            DC.L     RESTART                 RESTART SYSTEM ENTRY POINT
            ORG      INTVECTOR               DEFINE AN INTERRUPT VECTOR
            DC.L     INTHANDLER              HANDLER ADDRESS FOR THIS VECTOR
            ORG                              SYSTEM RESTART CODE
RESTART:
            NOP                              NO OPERATION EXAMPLE
            BRA.S    LABEL                   SHORT BRANCH
            ADD.W    D0, D1                  ADD REGISTER TO REGISTER
LABEL:
            SUB.W    DISP(A0), A1            SUBTRACT REGISTER INDIRECT WITH OFFSET
            CMP.W    D2, D3                  COMPARE REGISTER TO REGISTER
            SGE.B    D7                      Scc TO REGISTER
            ...
            ...
INTHANDLER:
            MOVE.W   LONGADR1, LONGADR2      MOVE WORD FROM AND TO LONG ADDRESS
            NOP                              NO OPERATION
            SWAP.W                           REGISTER SWAP
```

Figure 55 Instruction Prefetch Example, Memory Contents

The sequence we shall illustrate consists of the power-up reset, the execution of NOP, BRA, SUB, the taking of an interrupt, and the execution of the MOVE.W xxx.L to yyy.L. The order of operations described within each microroutine is not exact, but is intended for illustrative purpose only.

4

Microroutine	Operation	Location	Operand
Reset	Read	0	SSP High
	Read	2	SSP Low
	Read	4	PC High
	Read	6	PC Low
	Read	(PC)	NOP
	Read	+(PC)	BRA
	<begin NOP>		
NOP	Read	+(PC)	ADD
	<begin BRA>		
BRA	PC=PC+d		
	Read	(PC)	SUB
	Read	+(PC)	DISP
	<begin SUB>		
SUB	Read	+(PC)	CMP
	Read	DISP(A0)	<src>
	Read	+(PC)	SGE
	<begin CMP>	<take INT>	
INTERRUPT	Write	-(SSP)	PC Low
	Read	<INT ACK>	Vector #
	Write	-(SSP)	SR
	Write	-(SSP)	PC High
	Read	(VR)	PC High
	Read	+(VR)	PC Low
	Read	(PC)	MOVE
	Read	+(PC)	xxx High
	<begin MOVE>		
MOVE	Read	+(PC)	xxx Low
	Read	+(PC)	yyy High
	Read	xxx	<src>
	Read	+(PC)	yyy Low
	Write	yyy	<dest>
	Read	+(PC)	NOP
	Read	+(PC)	SWAP
	<begin NOP>		

Figure 56 Instruction Prefetch Example

- **DATA PREFETCH**

Normally the 68000 prefetches only instructions and not data. However, when the MOVEM instruction is used to move data from memory to registers, the data stream is prefetched in order to optimize performance. As a result, the processor reads one extra word beyond the higher end of the source area. For example, the instruction sequence in Figure 57 will operate as shown in Figure 58.

	...		MOVE TWO
	MOVEM.L	A, D0/D1	LONGWORDS INTO REGISTERS
	...		
A	DC.W	1	WORD 1
B	DC.W	2	WORD 2
C	DC.W	3	WORD 3
D	DC.W	4	WORD 4
E	DC.W	5	WORD 5
F	DC.W	6	WORD 6

Figure 57 MOVEM Example, Memory Contents

Assume Effective Address Evaluation is Already Done

Microroutine	Operation	Location	Other Operations
	...		
MOVEM	Read	A	
			Prepare to Fill D0
	Read	B	A → D0H
	Read	C	B → D0L
			Prepare to Fill D1
	Read	D	C → D1H
	Read	E	D → D1L
			Detect Register List Complete

Figure 58 MOVEM Example, Operation Sequence

Table 25 Instruction Set

Mnemonic / Operation	Size	Addr Mode	Dn #	Dn ~	An #	An ~	(An) #	(An) ~	(An)+ #	(An)+ ~	-(An) #	-(An) ~	d(An) #	d(An) ~	d(An, Xi) #	d(An, Xi) ~	Abs W #	Abs W ~	Abs L #	Abs L ~	d(PC) #	d(PC) ~	d(PC, Xi) #	d(PC, Xi) ~	#=Immed / #→SR/CCR #	~	Opcode Bit Pattern 15..8	Opcode Bit Pattern 7..0	Operation	Condition Codes XNZVC
ABCD Add Digits	B	s Dn d	2	6																							1100 RRR1	0000 0rrr	[illegible]	* * *
		s -(An) d									2	18															1100 RRR1	0000 1rrr		
ADD Add Binary	B/W	s-Dn d					2	12	2	12	2	14	4	16	4	18	4	16	6	20							1101 DDD1	SSEE EEEE	d+Dn→d	*****
		d-Dn s	2	4	2*	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1101 DDD0	SSee eeee	Dn+s→Dn	
	L	s-Dn d					2	20	2	20	2	22	4	24	4	26	4	24	6	28							1101 DDD1	10EE EEEE	d+Dn→d	
		d-Dn s	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1101 DDD0	10ee eeee	Dn+s→Dn	
ADDA Add Address	W	d An s	2	8	2	8	2	12	2	12	2	14	4	16	4	18	4	16	6	20	4	16	4	18	4	12	1101 AAA0	11ee eeee	[illegible]→An	
	L	d An s	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1101 AAA1	11ee eeee		
ADDI Add Immed	B/W	s Imm d	4	8	ADDA		4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 0110	SSEE EEEE	d+#→d	*****
	L	s Imm d	6	16	ADDA		6	28	6	28	6	30	8	32	8	34	8	32	10	36										
ADDQ Add Quick	B/W	s Imm3 d	2	4	2*	4	2	12	2	12	2	14	4	16	4	18	4	16	6	20							0101 QQQ0	SSEE EEEE	[illegible]	*****
	L	s Imm3 d	2	8	2	8	2	20	2	20	2	22	4	24	4	26	4	24	6	28										
ADDX Add Multi-precision	B/W	s-Dn d	2	4																							1101 RRR1	SS00 0rrr	d+s+X→d	*****
		s-(An) d									2	18															1101 RRR1	SS00 1rrr		
	L	s-Dn d	2	8																							1101 RRR1	1000 0rrr		
		s-(An) d									2	30															1101 RRR1	1000 1rrr		
AND Logic And	B/W	s Dn d					2	12	2	12	2	14	4	16	4	18	4	16	6	20							1100 DDD1	SSEE EEEE	[illegible]	**
		d Dn s	2	4			2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1100 DDD0	SSee eeee	Dn<and>s→Dn	
	L	s Dn d					2	20	2	20	2	22	4	24	4	26	4	24	6	28							1100 DDD1	10EE EEEE	d<and>Dn→d	
		d Dn s	2	8			2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1100 DDD0	10ee eeee	Dn<and>s→Dn	
ANDI And Immed	B/W	s Imm d	4	8			4	16	4	16	4	18	6	20	6	22	6	20	8	24					4	20	0000 0010	SSEE EEEE	d<and>#→d	-**00
	L	s Imm d	6	16			6	28	6	28	6	30	8	32	8	34	8	32	10	36										
ASL, ASR Arithmetic Shift	B/W	count Dn d	2	6+2n																							1110 rrr[illegible]	SS[illegible]0 0DDD	Left / Right (diagram)	*****
		count #1-8 d	2	6+2n																							1110 QQQ[illegible]	SS00 0DDD		
	L	count Dn d	2	8+2n																							1110 rrr[illegible]	1010 0DDD		
		count #1-8 d	2	8+2n																							1110 QQQ[illegible]	1000 0DDD		
Memory	W	count 1 d					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 000[illegible]	11EE EEEE		
BCHG Test and Change	B	bit#-Dn d					2	12	2	12	2	14	4	16	4	18	4	16	6	20							0000 rrr1	01EE EEEE	~(bit)# of d→Z, ~(bit)# of d→(bit)# of d	--*--
		bit#-Imm d					4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 1000	01EE EEEE		
	L	bit#-Dn d	2	<8																							0000 rrr1	01EE EEEE		
		bit#-Imm d	4	<12																							0000 1000	01EE EEEE		
BCLR Test and Clear	B	bit# Dn d					2	12	2	12	2	14	4	16	4	18	4	16	6	20							0000 rrr1	10EE EEEE	~(bit)# of d→Z, 0→(bit)# of d	--*--
		bit# Imm d					4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 1000	10EE EEEE		
	L	bit# Dn d	2	<10																							0000 rrr1	10EE EEEE		
		bit# Imm d	4	<14																							0000 1000	10EE EEEE		
BSET Test and Set	B	bit#-Dn d					2	12	2	12	2	14	4	16	4	18	4	16	6	20							0000 rrr1	11EE EEEE	~(bit)# of d→Z, 1→(bit)# of d	--*--
		bit#-Imm d					4	16	4	16	4	18	6	20	6	22	6	20	8	24							0000 1000	11EE EEEE		
	L	bit#-Dn d	2	<8																							0000 rrr1	11EE EEEE		
		bit#-Imm d	4	<12																							0000 1000	11EE EEEE		
BTST Bit Test	B	bit# Dn d					2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	10	0000 rrr1	00EE EEEE	~(bit)# of d→Z	--*--
		bit# Imm d					4	12	4	12	4	14	6	16	6	18	6	16	8	20	6	16	6	18			0000 1000	00EE EEEF		
	L	bit# Dn d	2	6																							0000 rrr1	00EE EEEE		
		bit# Imm d	4	10																							0000 1000	00EE EEEE		
CHK Check Register Against Bounds	W	s-Dn (bound)	2	<40 10	←trap→		2	<44 14	2	<44 14	2	<46 16	4	<48 18	4	<50 20	4	<48 18	6	<52 22	4	<48 18	4	<50 20	4	<44 14	0100 DDD1	10ee eeee	If Dn<0 or Dn>(bound), then trap	-*UUU
CLR Clear Operand	B/W	d	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0010	SSEE EEEE	0→d	-0100
	L	d	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
CMP Compare Binary	B/W	s-Dn	2	4	2*	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1011 DDD0	SSee eeee	Dn−s	-****
	L	s-Dn	2	6	2	6	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14				
CMPA Compare Address	W	d An s	2	6	2	6	2	10	2	10	2	12	4	14	4	16	4	14	6	18	4	14	4	16	4	10	1011 AAA0	11ee eeee	An−s	-****
	L	d An s	2	6	2	6	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1011 AAA1	11ee eeee		
CMPI Compare Imm	B/W	s Imm d	4	8	CMPA		4	12	4	12	4	14	6	16	6	18	6	16	8	20							0000 1100	SSEE EEEE	d−#	-****
	L	s Imm d	6	14	CMPA		6	20	6	20	6	22	8	24	8	26	8	24	10	28										
CMPM Compare Memory	B/W	s (An)+ d							2	12																	[illegible] RRR[illegible]	SS00 [illegible]	[illegible]	-****
	L	s (An)+ d							2	20																				
DIVS Divide Signed	W	s-Dn	2	<158			2	<162	2	<162	2	<164	4	<166	4	<168	4	<166	6	<170	4	<166	4	<168	4	<162	1000 DDD1	11ee eeee	Dn32/s16→Dn(r:q)	-***0
DIVU Divide Unsigned	W	d Dn s	2	140			2	144	2	144	2	146	4	148	4	150	4	148	6	152	4	148	4	150	4	144	1000 DDD0	11ee eeee	Dn32/s16→Dn(r:q)	-***0
EOR Exclusive OR Logical	B/W	s-Dn d	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							1011 rrr1	SSEE EEEE	d⊕Dn→d	-**00
	L	s-Dn d	2	8			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
EORI Exclusive OR Immediate	B/W	s Imm d	4	8			4	16	4	16	4	18	6	20	6	22	6	20	8	24					4	20	0000 1010	SSEE EEEE	d⊕#→d	-**00
	L	s Imm d	6	16			6	28	6	28	6	30	8	32	8	34	8	32	10	36										
EXG Exchange Registers	L	s-Dn d	2	6																							1100 DDD1	0100 0DDD	s↔d	-----
		s-An d			2	6																					1100 AAA1	0100 1AAA		
																											1100 DDD1	1000 1AAA		
EXT Sign Extend	W	d	2	4																							0100 1000	1000 0DDD	bit 7→bit 8–15	
	L	d	2	4																							0100 1000	1100 0DDD	bit 15→bit 16–31	-**00
LEA Load Effective Address	L	s-An					2	4					4	8	4	12	4	8	6	12	4	8	4	12			0100 AAA1	11ee eeee	s→An	-----
LINK Link and Allocate		disp Imm s			4	16																					0100 1110	0101 0AAA	An→-(SP), SP→An, SP+disp→SP	

Note: Refer to Condition Code Computations for condition Code

* Word only

< Maximum value

\# Number of Program Bytes

~ Number of Clock Periods

Opcode Bit Pattern Key

A Address Register #

C Test Condition

D Data Register #

e Source Effective Address

E Destination Effective Address

d Direction 0: Right 1: Left

M Destination EA Mode

P Displacement

Q Quick Immediate Data

r Source Register

R Destination Register

Size [illegible]: Byte, Word, Long Word

(to be continued)

Mnemonic Operation	Size	Addr. Mode	Dn		An		(An)		(An) +		− (An)		d(An)		d(An, Xi)		Abs. W		Abs. L		d(PC)		d(PC, Xi)		s = Immed d = SR/CC		Opcode Bit Pattern 1111 11 5432 1098	7654 3210	Boolean	Condition Codes XNZVC
			#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~	#	~				
LSL, LSR Logical Shift	B W	count Dn d	2	6 + 2n																							1110 rrrf	SS10 1DDD	Left: C, X ← ▭ ← 0; Right: 0 → ▭ → C, X	***0*
		count #1-8 d	2	6 + 2n																							1110 QQQf	SS00 1DDD		
	L	count Dn d	2	8 + 2n																							1110 rrrf	1010 1DDD		
		count #1-8 d	2	8 + 2n																							1110 QQQf	1000 1DDD		
Memory	W	count 1 d					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 001f	11EE EEEE		
MOVE Move Data	B W	s Dn d	2	4	MOVEA		2	8	2	8	2	8	4	12	4	14	4	12	6	16							00SS RRRM	MMss ssss	s→d	-**00
		s An d	2	4	MOVEA		2	8	2	8	2	8	4	12	4	14	4	12	6	16										
		s (An) d	2	8	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s (An)+ d	2	8	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s −(An) d	2	10	MOVEA		2	14	2	14	2	14	4	18	4	20	4	18	6	22										
		s d(An) d	4	12	MOVEA		4	16	4	16	4	16	6	20	6	22	6	20	8	24										
		s d(An,X) d	4	14	MOVEA		4	18	4	18	4	18	6	22	6	24	6	22	8	26										
		s Abs W d	4	12	MOVEA		4	16	4	16	4	16	6	20	6	22	6	20	8	24										
		s Abs L d	6	16	MOVEA		6	20	4	20	6	20	8	24	8	26	8	24	10	28										
		s d(PC) d	4	12	MOVEA		4	16	4	16	4	16	6	20	6	22	6	20	8	24										
		s d(PC,X) d	4	14	MOVEA		4	18	4	18	4	18	6	22	6	24	6	22	8	26										
		s Imm d	4	8	MOVEA		4	12	4	12	4	12	6	16	6	18	6	16	8	20										
	L	s Dn d	2	4	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s An d	2	4	MOVEA		2	12	2	12	2	12	4	16	4	18	4	16	6	20										
		s (An) d	2	12	MOVEA		2	20	2	20	2	20	4	24	4	26	4	24	6	28										
		s (An)+ d	2	12	MOVEA		2	20	2	20	2	20	4	24	4	26	4	24	6	28										
		s −(An) d	2	14	MOVEA		2	22	2	22	2	22	4	26	4	28	4	26	6	30										
		s d(An) d	4	16	MOVEA		4	24	4	24	4	24	6	28	6	30	6	28	8	32										
		s d(An,X) d	4	18	MOVEA		4	26	4	26	4	28	6	30	6	32	6	30	8	34										
		s Abs W d	4	16	MOVEA		4	24	4	24	4	24	6	28	6	30	6	28	8	32										
		s Abs L d	6	20	MOVEA		6	[illegible]	4	28	6	28	8	32	8	34	8	32	10	36										
		s d(PC) d	4	16	MOVEA		4	24	4	24	4	24	6	28	6	30	6	28	8	32										
		s d(PC,X) d	4	18	MOVEA		4	26	4	26	4	26	6	30	6	32	6	30	8	34										
		s Imm d	6	12	MOVEA		6	20	6	20	6	20	8	24	8	26	8	24	10	28										
MOVE Move to Condition Codes	W	d CCR s	2	12			2	16	2	16	2	18	4	20	4	22	4	20	6	24	4	20	4	22	4	16	0100 0100	11ee eeee	s→CCR	*****
MOVE Move to from Status Reg	W	d SR s	2	12			2	16	2	16	2	18	4	20	4	22	4	20	6	24	4	20	4	22	4	16	0100 0110	11ee eeee	s→SR	*****
		s SR d	2	6			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0000	11EE EEEE	d→MPU SR→d	-----
MOVE Move to from User SP (A7)	L	s USP d			2	4																					0100 1110	0110 1AAA	USP→An	-----
		d USP s			2	4																					0100 1110	0110 0AAA	An→USP	
MOVEA Move Address	W	d An	2	4	2	4	2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	0011 AAA0	01ee eeee	s→An	-----
	L	d An	2	4	2	4	2	12	2	12	2	14	4	16	4	18	4	16	6	20	4	16	4	18	6	12	0010 AAA0	01ee eeee		
MOVEM Move Multiple Registers	W	s Xn d					4	8+4n			4	8+4n	6	12+4n	6	14+4n	6	12+4n	8	16+4n							0100 1000 a7~a0	1EEE EEEE d7~d0	s→d	-----
		d Xn s					4	12+4n	4	12+4n			6	16+4n	6	18+4n	6	16+4n	8	20+4n	6	16+4n	6	18+4n			0100 1100 a7~a0	10ee eeee d7~d0	s→d**	
	L	s Xn d					4	8+8n			4	8+8n	6	12+8n	6	14+8n	6	12+8n	8	16+8n							0100 1000 a7~a0	11EE EEEE d7~d0	s→d	
		d Xn s					4	12+8n	4	12+8n			6	16+8n	6	18+8n	6	16+8n	8	20+8n	6	16+8n	6	18+8n			0100 1100 a7~a0	11ee eeee d7~d0	s→d**	
MOVEP Move Peripheral	W	s Dn d d(An)											4	16													0000 DDD1	1000 1AAA	Dn→d by bytes	-----
		s d(An) d Dn	4	16																							0000 DDD1	0000 1AAA	s→Dn by bytes	
		s Dn d											4	24													0000 DDD1	1100 1AAA	Dn→d by bytes	
		s d(An) d	4	24																							0000 DDD1	0100 1AAA	s→Dn by bytes	
MOVEQ Move Quick	L	s Imm8 d	2	4																							0111 DDD0	QQQQ QQQQ	#→Dn	-**00
MULS Multiply Signed	W	d Dn s	2	<70			2	<74	2	<74	2	<76	4	<78	4	<80	4	<78	6	<82	4	<78	4	<80	4	<74	1100 DDD1	11ee eeee	Dn×s→Dn	-**00
MULU Multiply Unsigned	W	d Dn s	2	<70			2	<74	2	<74	2	<76	4	<78	4	<80	4	<78	6	<82	4	<78	4	<80	4	<74	1100 DDD0	11ee eeee	Dn×s→Dn	-**00
NBCD Negate Digit	B	d	2	6			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 1000	00EE EEEE	0−d10−X→d	*U*U*
NEG Negate Binary	B W	d	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0100	SSEE EEEE	0−d→d	*****
	L	d	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
NEGX Negative Multi precision	B W	d	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0000	SSEE EEEE	0−d−X→d	*****
	L	d	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
NOT Logical Complement	B W	d	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20							0100 0110	SSEE EEEE	~d→d	-**00
	L	d	2	6			2	20	2	20	2	22	4	24	4	26	4	24	6	28										
OR Inclusive OR Logical	B W	s Dn d					2	12	2	12	2	14	4	16	4	18	4	16	6	20							1000 DDD1	SSEE EEEE	d<or>Dn→d	-**00
		d Dn s	2	4			2	8	2	8	2	10	4	12	4	14	4	12	6	16	4	12	4	14	4	8	1000 DDD0	SSee eeee	Dn<or>s→Dn	
	L	s Dn d					2	20	2	20	2	22	4	24	4	26	4	24	6	28							1000 DDD1	10EE EEEE	d<or>Dn→d	
		d Dn s	2	8			2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1000 DDD0	10ee eeee	Dn<or>s→Dn	
ORI OR Immediate	B W	s Imm d	4	8			4	16	4	16	4	18	6	20	6	22	6	20	8	24					4	20	0000 0000	SSEE EEEE	d<or>#→d	-**00
	L	s Imm d	6	16			6	30	6	30	6	32	8	34	8	36	8	34	10	38										
PEA Push Effective Address	L	s					2	14					4	18	4	22	4	18	6	22	4	18	4	22			0100 1000	01ee eeee	s→−(SP)	-----
ROR, ROL Rotate without X	B W	count Dn d	2	6+2n																							1110 rrrf	1S11 1DDD	Left: C ← ▭ ←; Right: → ▭ → C	-**0*
		count #1~8 d	2	6+2n																							1110 QQQf	SS01 1DDD		
	L	count Dn d	2	8+2n																							1110 rrrf	1011 1DDD		
		count #1~8 d	2	8+2n																							1110 QQQf	1001 1DDD		
Memory	W	count 1 d					2*	12	2*	12	2*	14	4*	16	4*	18	4*	16	6*	20							1110 011f	11EE EEEE		

Note: Refer to Condition Code Computations as for condition Code
* Word only
· Maximum value
\# Number of Program Bytes
~ Number of Clock Periods

** The MPU goes through an extra null read cycle after a multiple read is done (The last EA + 2)

Opcode Bit Pattern Key

- A Address Register #
- C Test Condition
- D Data Register #
- e Source Effective Address
- E Destination Effective Address
- f Direction, 0 Right 1 Left
- M Destination EA Mode
- P Displacement
- Q Quick Immediate Data
- r Source Register
- R Destination Register
- S Size 00 - Byte, 01 Word, 10 - Long Word, 11 - Another Operation
- V Vector #

In the MOVE instruction: 01 Byte, 10 - Long Word, 11 Word

(to be continued)

Mnemonic Operation	Size	Addr Mode	Dn #	Dn ~	An #	An ~	(An) #	(An) ~	(An)+ #	(An)+ ~	-(An) #	-(An) ~	d(An) #	d(An) ~	d(An, Xi) #	d(An, Xi) ~	Abs W #	Abs W ~	Abs L #	Abs L ~	d(PC) #	d(PC) ~	d(PC, Xi) #	d(PC, Xi) ~	s = Immed d = SR/CC #	s = Immed d = SR/CC ~	Opcode Bit Pattern 1111 11 5432 1098	Opcode Bit Pattern 7654 3210	Boolean	Condition Codes X N Z V C
ROXR, ROXL Rotate through X	B W	count Dn d	2	6 + 2n																							[illegible]0 rrrf	SS[illegible] 0DDD	Right n (C, X)	*** * *
		count #1 ~ 8 d	2	6 + 2n																							1110 QQQf	SS0[illegible] 0DDD		
	L	count Dn d	2	8 + 2n																							[illegible]0 rrrf	101 0DDD	Left n (C, X)	
		count #1 ~ 8 d	2	8 + 2n																							[illegible]0 QQQf	100 0DDD		
Memory	W	count 1 d					2*	[illegible]	2*	[illegible]	2*	[illegible]	4*	[illegible]	4*	[illegible]	4*	[illegible]	6*	[illegible]							1110 010f	11EE EEEE		
SBCD Subtract digits	B	Dy, Dx d	2	[illegible]																							1000 RRR1	0000 0rrr	d10 - s10 - X → d	*U*U*
		-(Ay), -(Ax) d									2	18															1000 RRR1	0000 1rrr		
Scc Set Conditionally	B	cc d	2	6 4			2	[illegible]	2	[illegible]	2	[illegible]	4	16	4	18	4	16	6	20							0101 cccc	11EE EEEE	If cc true then 1s → d Else 0s → d	- - - - -
SUB Subtract Binary	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1001 DDD1	SSEE EEEE	d - Dn → d	*****
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1001 DDD0	SSee eeee	Dn - s → Dn	
	L	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1001 DDD1	10EE EEEE	d - Dn → d	
	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1001 DDD0	10ee eeee	Dn - s → Dn	
SUBA Subtract Address	W	d: An s	2	8	2	8	2	12	2	12	2	14	4	16	4	18	4	16	6	20	4	16	4	18	4	12	1001 AAA0	11ee eeee	An - s → An	- - - - -
	L	d: An s	2	8	2	8	2	14	2	14	2	16	4	18	4	20	4	18	6	22	4	18	4	20	6	14	1001 AAA1	11ee eeee		
SUBI Subtract Immediate	B W	[illegible]	[illegible]	[illegible]			[illegible]	[illegible]	[illegible]	16	[illegible]	18	6	20	6	22	[illegible]	20	8	24							0000 0100	SSEE EEEE	d - # → d	*****
	L	[illegible]	[illegible]	16			6	28	6	28	6	30	8	32	8	34	[illegible]	32	10	36										
SUBQ Subtract Quick	B W	s: Imm3 d	2	4	2*	4	2	[illegible]	2	[illegible]	2	14	4	16	4	[illegible]	4	[illegible]	6	20							0101 QQQ1	SSEE EEEE	d - # → d	*****
	L	s: Imm3 d	2	8	2	8	2	[illegible]	2	[illegible]	[illegible]	[illegible]	4	24	4	[illegible]	4	24	6	28										
SUBX Subtract Multiprecision	B W	[illegible]	2	[illegible]																							1001 RRR1	SS00 0rrr	d - s - X → d	*****
		[illegible]									2	18															1001 RRR1	SS00 1rrr		
	L	[illegible]	2	8																							1001 RRR1	1000 0rrr		
		-(Ay), -(Ax) [illegible]									2	30															1001 RRR1	1000 1rrr		
SWAP Swap Register Halves	W	d	2	4																							0100 1000	0100 0DDD	[illegible]	**
TAS Test and Set Operand	B	[illegible]	2	[illegible]			[illegible]	[illegible]	[illegible]	[illegible]	2	[illegible]	[illegible]	[illegible]	[illegible]	20	[illegible]	[illegible]	[illegible]	22							0100 1010	11EE EEEE	test d → cc 1 → bit 7 of d	-**00
TST Test	B W	d	2	4			2	8	2	8	2	10	4	12	4	14	4	12	6	16							0100 1010	SSEE EEEE	test d	**
	L	d	2	4			2	12	2	12	2	14	4	16	4	18	4	16	6	20										
UNLK Unlink		[illegible]			2	12																					0100 1110	0101 1AAA	An → SP (SP)+ → An	- - - - -
Bcc Branch Conditionally	B	disp																							bra taken [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
																									bra not taken 2	8				
	W	disp																							bra taken 4	10				
																									bra not taken 4	14				
BRA Branch Always	B	disp																							2	10	0110 0000	PPPP PPPP	PC + disp → PC	- - - - -
	W	disp																							4	10				
BSR Branch to Subroutine	B	disp																							2	[illegible]	0110 0001	PPPP PPPP	PC → -(SP) PC + disp → PC	
	W	disp																							4	[illegible]				
DBcc Decrement Counter & Branch Until Condition True or Count = 1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0101 CCCC	1100 1DDD	[illegible]	- - - - -
JMP Jump to		d					2	8					4	10	4	14	4	10	6	12	4	10	4	14			0100 1110	11EE EEEE	d → PC	- - - - -
JSR Jump to Subroutine		[illegible]					[illegible]	[illegible]					[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	22			0100 1110	10EE EEEE	PC → (SP) [illegible] → PC	- - - - -
NOP No Operation			2	4																							0100 1110	0111 0001	none	- - - - -
RESET Reset External Devices			[illegible]	[illegible]																							0100 1110	0111 0000	[illegible]	- - - - -
RTE Return from Exception			2	20																							0100 1110	0[illegible]	(SP)+ → SR (SP)+ → PC	*****
RTR Return from Subroutine/ Restore CC			[illegible]	[illegible]																							0100 1110	0111 0111	(SP)+ → CC (SP)+ → PC	*****
RTS Return from Subroutine			2	16																							0100 1110	0111 0101	(SP)+ → PC	- - - - -
STOP Load SR/Stop			[illegible]	[illegible]																							0100 1110	0111 0010	[illegible]	*****
TRAP Trap			2	34																							0100 1110	0100 VVVV	PC → (SSP) SR → (SSP) (Vector) → PC	- - - - -
TRAPV Trap if Overflow Set			[illegible]	[illegible]																							[illegible]	[illegible]	[illegible]	[illegible]

Note: Refer to Condition Code Computations as for condition Code
* Word only
< Maximum value
\# Number of Program Bytes
~ Number of Clock Periods

Opcode Bit Pattern Key

A Address Register #
C Test Condition
D Data Register #
e Source Effective Address
E Destination Effective Address

f Direction 0 Right 1 Left
M Destination EA Mode
P Displacement
Q Quick Immediate Data
r Source Register

R Destination Register
S Size 00 Byte
01 Word
10 Long Word
11 Another Operation
V Vector #

s the MOVE instruction
01 Byte
10 Long Word
11 Word

4

■ INSTRUCTION FORMAT SUMMARY

This provides a summary of the first word in each instruction of the instruction set. Table 26 is an operation code (op-code) map which illustrates how bits 15 through 12 are used to specify the operations. The remaining paragraph groups the instructions according to the op-code map.

where, Size; Byte = 00　　Sz; Word = 0
Word = 01　　Long Word = 1
Long Word = 10

Table 26 Operation Code Map

Bits 15 thru 12	Operation
0000	Bit Manipulation/MOVEP/Immediate
0001	Move Byte
0010	Move Long
0011	Move Word
0100	Miscellaneous
0101	ADDQ/SUBQ/S_{CC}/DB_{CC}
0110	B_{CC}
0111	MOVEQ
1000	OR/DIV/SBCD
1001	SUB/SUBX
1010	(Unassigned)
1011	CMP/EOR
1100	AND/MUL/ABCD/EXG
1101	ADD/ADDX
1110	Shift/Rotate
1111	(Unassigned)

(1) BIT MANIPULATION, MOVE PERIPHERAL, IMMEDIATE INSTRUCTIONS

Dynamic Bit

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

Static Bit

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

Bit Type Codes: TST = 00, CHG = 01, CLR = 10, SET = 11

MOVEP

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

Op-Mode; Word to Reg = 100, Long to Reg = 101, Word to Mem = 110, Long to Mem = 111

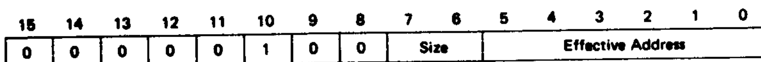
OR Immediate

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

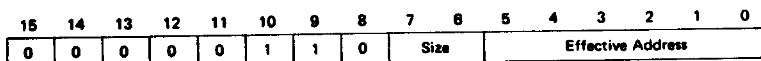
AND Immediate

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

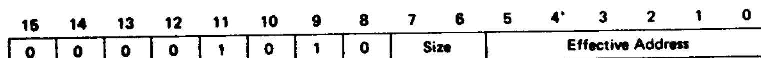
SUB Immediate



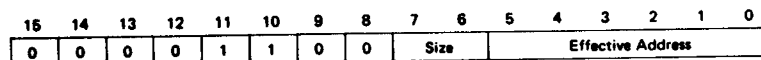
ADD Immediate



EOR Immediate

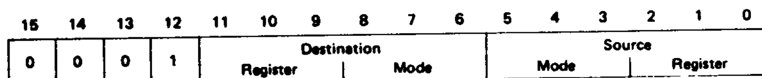


CMP Immediate



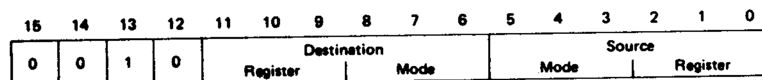
(2) MOVE BYTE INSTRUCTION

MOVE Byte



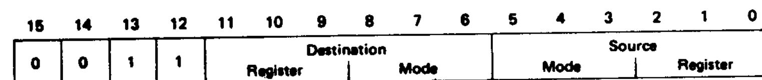
(3) MOVE LONG INSTRUCTION

MOVE Long



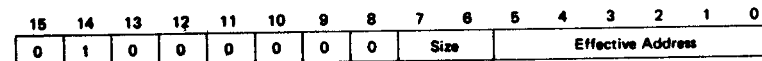
(4) MOVE WORD INSTRUCTION

MOVE Word

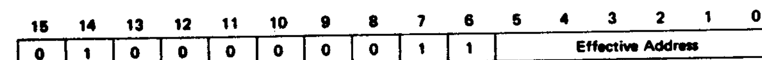


(5) MISCELLANEOUS INSTRUCTIONS

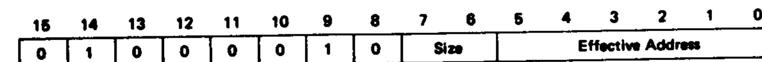
NEGX



MOVE from SR



CLR



NEG

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

MOVE to CCR

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

NOT

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

MOVE to SR

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

NBCD

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

PEA

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

SWAP

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

MOVEM Registers to EA

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

EXTW

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

EXTL

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

TST

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

TAS

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

MOVEM EA to Registers

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

TRAP

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

LINK

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

UNLK

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

MOVE to USP

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

MOVE from USP

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

RESET

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

NOP

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

STOP

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

RTE

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

RTS

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

TRAPV

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

RTR

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

JSR

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

JMP

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

CHK

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

LEA

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

(6) ADD QUICK, SUBTRACT QUICK, SET CONDITIONALLY, DECREMENT INSTRUCTIONS

ADDQ

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

SUBQ

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

S_{CC}

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

DB_{CC}

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

(7) BRANCH CONDITIONALLY, BRANCH TO SUBROUTINE INSTRUCTION

B_{CC}

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

BSR

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

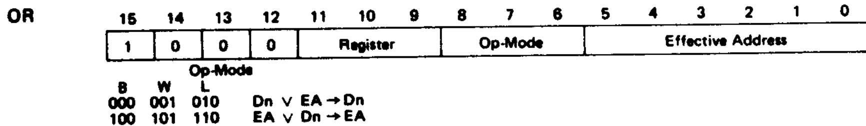
(8) MOVE QUICK INSTRUCTION

MOVEQ

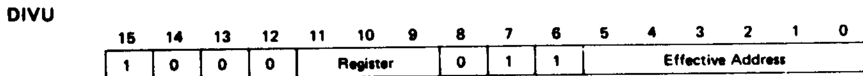
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	Register			0	Data							

(9) OR, DIVIDE, SUBTRACT DECIMAL INSTRUCTIONS

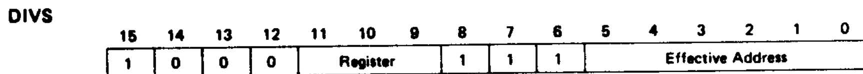
OR



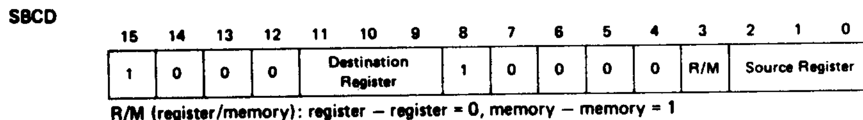
DIVU



DIVS



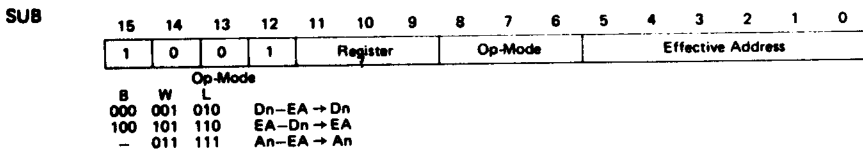
SBCD



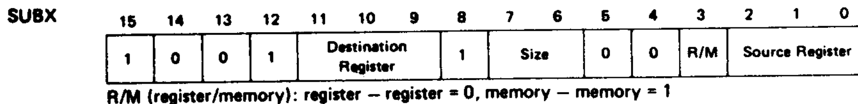
R/M (register/memory): register — register = 0, memory — memory = 1

(10) SUBTRACT, SUBTRACT EXTENDED INSTRUCTIONS

SUB



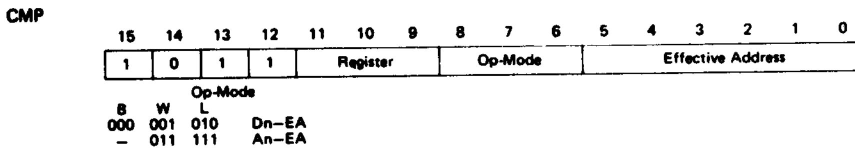
SUBX



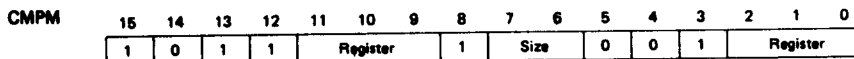
R/M (register/memory): register — register = 0, memory — memory = 1

(11) COMPARE, EXCLUSIVE OR INSTRUCTIONS

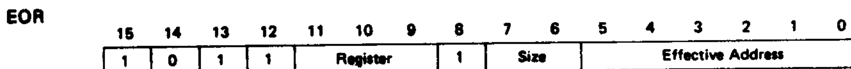
CMP



CMPM

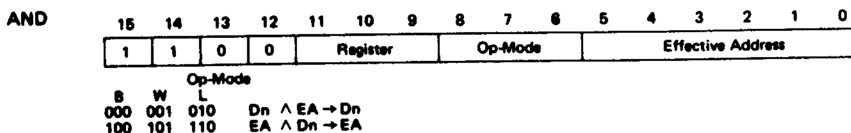


EOR



(12) AND, MULTIPLY, ADD DECIMAL, EXCHANGE INSTRUCTIONS

AND



MULU

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

MULS

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

ABCD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	Destination Register			1	0	0	0	0	R/M	Source Register		

R/M (register/memory): register – register = 0, memory – memory = 1

EXGD

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

EXGA

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

EXGM

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

(13) ADD, ADD EXTENDED INSTRUCTIONS

ADD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Register			Op-Mode			Effective Address					

Op-Mode B	W	L	
000	001	010	Dn + EA → Dn
100	101	110	EA + Dn → EA
–	011	111	An + EA → An

ADDX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	Destination Register			1	Size		0	0	R/M	Source Register		

R/M (register/memory): register – register = 0, memory – memory = 1

(14) SHIFT/ROTATE INSTRUCTIONS

Data Register Shifts

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	Count/Register			d	Size		i/r	Type		Register		

Memory Shifts

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

Shift Type Codes: AS = 00, LS = 01, ROX = 10, RO = 11
d (direction): Right = 0, Left = 1
i/r (count source): Immediate Count = 0, Register Count = 1

■ INSTRUCTION EXECUTION TIMES

The following paragraphs contain listings of the instruction execution times in terms of external clock (CLK) periods. In this timing data, it is assumed that both memory read and write cycle times are four clock periods. Any wait states caused by a longer memory cycle must be added to the total instruction time. The number of bus read and write cycles for each instruction is also included with the timing data. This data is enclosed in parenthesis following the execution periods and is shown as: (r/w) where r is the number of read cycles and w is the number of write cycles.

(NOTE) The number of periods includes instruction fetch and all applicable operand fetches and stores.

● EFFECTIVE ADDRESS OPERAND CALCULATION TIMING

Table 27 lists the number of clock periods required to compute an instruction's effective address. It includes fetching of any extension words, the address computation, and fetching of the memory operand. The number of bus read and write cycles is shown in parenthesis as (r/w). Note there are no write cycles involved in processing the effective address.

● MOVE INSTRUCTION CLOCK PERIODS

Table 28 and 29 indicate the number of clock periods for the move instruction. This data includes instruction fetch, operand reads, and operand writes. The number of bus read and write cycles is shown in parenthesis as: (r/w).

● STANDARD INSTRUCTION CLOCK PERIODS

The number of clock periods shown in Table 30 indicates the time required to perform the operations, store the results, and read the next instruction. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 30 the headings have the following meanings: An = address register operand, Dn = data register operand, ea = an operand specified by an effective address, and M = memory effective address operand.

● IMMEDIATE INSTRUCTION CLOCK PERIODS

The number of clock periods shown in Table 31 includes the time to fetch immediate operands, perform the operations, store the results, and read the next operation. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

In Table 31, the headings have the following meanings: # = immediate operand, Dn = data register operand, An = address register operand, M = memory operand, CCR = condition code register, and SR = status register.

● SINGLE OPERAND INSTRUCTION CLOCK PERIODS

Table 32 indicates the number of clock periods for the single operand instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

Table 27 Effective Address Calculation Timing

Addressing Mode		Byte, Word	Long
	Register		
Dn	Data Register Direct	0(0/0)	0(0/0)
An	Address Register Direct	0(0/0)	0(0/0)
	Memory		
An@	Address Register Indirect	4(1/0)	8(2/0)
An@ +	Address Register Indirect with Postincrement	4(1/0)	8(2/0)
An@ -	Address Register Indirect with Predecrement	6(1/0)	10(2/0)
An@(d)	Address Register Indirect with Displacement	8(2/0)	12(3/0)
An@(d, ix)*	Address Register Indirect with Index	10(2/0)	14(3/0)
xxx.W	Absolute Short	8(2/0)	12(3/0)
xxx.L	Absolute Long	12(3/0)	16(4/0)
PC@(d)	Program Counter with Displacement	8(2/0)	12(3/0)
PC@(d, ix)*	Program Counter with Index	10(2/0)	14(3/0)
#xxx	Immediate	4(1/0)	8(2/0)

* The size of the index register (ix) does not affect execution time.

Table 28 Move Byte and Word Instruction Clock Periods

Source	Destination								
	Dn	An	An@	An@+	An@-	An@(d)	An@(d, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An	4(1/0)	4(1/0)	8(1/1)	8(1/1)	8(1/1)	12(2/1)	14(2/1)	12(2/1)	16(3/1)
An@	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
An@+	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)
An@-	10(2/0)	10(2/0)	14(2/1)	14(2/1)	14(2/1)	18(3/1)	20(3/1)	18(3/1)	22(4/1)
An@(d)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
An@(d, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
xxx.W	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
xxx.L	16(4/0)	16(4/0)	20(4/1)	20(4/1)	20(4/1)	24(5/1)	26(5/1)	24(5/1)	28(6/1)
PC@(d)	12(3/0)	12(3/0)	16(3/1)	16(3/1)	16(3/1)	20(4/1)	22(4/1)	20(4/1)	24(5/1)
PC@(d, ix)*	14(3/0)	14(3/0)	18(3/1)	18(3/1)	18(3/1)	22(4/1)	24(4/1)	22(4/1)	26(5/1)
#xxx	8(2/0)	8(2/0)	12(2/1)	12(2/1)	12(2/1)	16(3/1)	18(3/1)	16(3/1)	20(4/1)

* The size of the index register (ix) does not affect execution time.

Table 29 Move Long Instruction Clock Periods

Source	Destination								
	Dn	An	An@	An@+	An@-	An@(d)	An@(d, ix)*	xxx.W	xxx.L
Dn	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An	4(1/0)	4(1/0)	12(1/2)	12(1/2)	12(1/2)	16(2/2)	18(2/2)	16(2/2)	20(3/2)
An@	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
An@+	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)
An@-	14(3/0)	14(3/0)	22(3/2)	22(3/2)	22(3/2)	26(4/2)	28(4/2)	26(4/2)	30(5/2)
An@(d)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
An@(d, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
xxx.W	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
xxx.L	20(5/0)	20(5/0)	28(5/2)	28(5/2)	28(5/2)	32(6/2)	34(6/2)	32(6/2)	36(7/2)
PC@(d)	16(4/0)	16(4/0)	24(4/2)	24(4/2)	24(4/2)	28(5/2)	30(5/2)	28(5/2)	32(6/2)
PC@(d, ix)*	18(4/0)	18(4/0)	26(4/2)	26(4/2)	26(4/2)	30(5/2)	32(5/2)	30(5/2)	34(6/2)
#xxx	12(3/0)	12(3/0)	20(3/2)	20(3/2)	20(3/2)	24(4/2)	26(4/2)	24(4/2)	28(5/2)

* The size of the index register (ix) does not affect execution time.

Table 30 Standard Instruction Clock Periods

Instruction	Size	op < ea >, An	op < ea >, Dn	op Dn, < M >
ADD	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long	6(1/0) + **	6(1/0) + **	12(1/2) +
AND	Byte, Word	–	4(1/0) +	8(1/1) +
	Long	–	6(1/0) + **	12(1/2) +
CMP	Byte, Word	6(1/0) +	4(1/0) +	–
	Long	6(1/0) +	6(1/0) +	–
DIVS	–	–	158(1/0) + *	–
DIVU	–	–	140(1/0) + *	–
EOR	Byte, Word	–	4(1/0) ***	8(1/1) +
	Long	–	8(1/0) ***	12(1/2) +
MULS	–	–	70(1/0) + *	–
MULU	–	–	70(1/0) + *	–
OR	Byte, Word	–	4(1/0) +	8(1/1) +
	Long	–	6(1/0) + **	12(1/2) +
SUB	Byte, Word	8(1/0) +	4(1/0) +	8(1/1) +
	Long	6(1/0) + **	6(1/0) + **	12(1/2) +

+ add effective address calculation time
* indicates maximum value
** total of 8 clock periods for instruction if the effective address is register direct
*** only available effective address mode is data register direct

DIVS, DIVU – The divide algorithm used by the 68000 provides less than 10% difference between the best and worst case timings.

MULS, MULU – The multiply algorithm requires 38+2n clocks when n is defined as

MULU; n = the number of ones in the < ea >

MULS; n = concatanate the < ea > with a zero as the LSB; n is the resultant number of 10 or 01 patterns in the 17-bit source; i.e. worst case happens when the source is $5555.

Table 31 Immediation Instruction Clock Periods

Instruction	Size	op #, Dn	op #, An	op #, M	op #, CCR/SR
ADDI	Byte, Word	8(2/0)	–	12(2/1) +	–
	Long	16(3/0)	–	20(3/2) +	–
ADDQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1) +	–
	Long	8(1/0)	8(1/0)	12(1/2) +	–
ANDI	Byte, Word	8(2/0)	–	12(2/1) +	20(3/0)
	Long	16(3/0)	–	20(3/1) +	–
CMPI	Byte, Word	8(2/0)	8(2/0)	8(2/0) +	–
	Long	14(3/0)	14(3/0)	12(3/0) +	–
EORI	Byte, Word	8(2/0)	–	12(2/1) +	20(3/0)
	Long	16(3/0)	–	20(3/2) +	–
MOVEQ	Long	4(1/0)	–	–	–
ORI	Byte, Word	8(2/0)	–	12(2/1) +	20(3/0)
	Long	16(3/0)	–	20(3/2) +	–
SUBI	Byte, Word	8(2/0)	–	12(2/1) +	–
	Long	16(3/0)	–	20(3/2) +	–
SUBQ	Byte, Word	4(1/0)	8(1/0)*	8(1/1) +	–
	Long	8(1/0)	8(1/0)	12(1/2) +	–

+ add effective address calculation time
* word only

Table 32 Single Operand Instruction Clock Periods

Instruction	Size	Register	Memory
CLR	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
NBCD	Byte	6(1/0)	8(1/1) +
NEG	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
NEGX	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
NOT	Byte, Word	4(1/0)	8(1/1) +
	Long	6(1/0)	12(1/2) +
Scc	Byte, False	4(1/0)	8(1/1) +
	Byte, True	6(1/0)	8(1/1) +
TAS	Byte	4(1/0)	10(1/1) +
TST	Byte, Word	4(1/0)	4(1/0) +
	Long	4(1/0)	4(1/0) +

+ add effective address calculation time

- **SHIFT/ROTATE INSTRUCTION CLOCK PERIODS**

Table 33 indicates the number of clock periods for the shift and rotate instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

- **BIT MANIPULATION INSTRUCTION CLOCK PERIODS**

Table 34 indicates the number of clock periods required for the bit manipulation instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

- **CONDITIONAL INSTRUCTION CLOCK PERIODS**

Table 35 indicates the number of clock periods required for the conditional instructions. The number of bus read and write cycles is indicated in parenthesis as: (r/w). The number of clock periods and the number of read and write cycles must be added respectively to those of the effective address calculation where indicated.

- **JMP, JSR, LEA, PEA, MOVEM INSTRUCTION CLOCK PERIODS**

Table 36 indicates the number of clock periods required for the jump, jump to subroutine, load effective address, push effective address, and move multiple registers instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 33 Shift/Rotate Instruction Clock Periods

Instruction	Size	Register	Memory
ASR, ASL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
LSR, LSL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
ROR, ROL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–
ROXR, ROXL	Byte, Word	6 + 2n(1/0)	8(1/1) +
	Long	8 + 2n(1/0)	–

Table 34 Bit Manipulation Instruction Clock Periods

Instruction	Size	Dynamic		Static	
		Register	Memory	Register	Memory
BCHG	Byte	–	8(1/1) +	–	12(2/1) +
	Long	8(1/0)*	–	12(2/0)*	–
BCLR	Byte	–	8(1/1) +	–	12(2/1) +
	Long	10(1/0)*	–	14(2/0)*	–
BSET	Byte	–	8(1/1) +	–	12(2/1) +
	Long	8(1/0)*	–	12(2/0)*	–
BTST	Byte	–	4(1/0) +	–	8(2/0) +
	Long	6(1/0)	–	10(2/0)	–

+ add effective address calculation time
* indicates maximum value

Table 35 Conditional Instruction Clock Periods

Instruction	Displacement	Trap or Branch Taken	Trap of Branch Not Taken
B_{CC}	Byte	10(2/0)	8(1/0)
	Word	10(2/0)	12(2/0)
BRA	Byte	10(2/0)	–
	Word	10(2/0)	–
BSR	Byte	18(2/2)	–
	Word	18(2/2)	–
DB_{CC}	CC_{true}	–	12(2/0)
	CC_{false}	10(2/0)	14(3/0)
CHK	–	40(5/3) + *	10(1/0) +
TRAP	–	34(4/3)	–
TRAPV	–	34(5/3)	4(1/0)

+ add effective address calculation time
* indicates maximum value

Table 36 JMP, JSR, LEA, PEA, MOMEM Instruction Clock Periods

Instr	Size	An@	An@ +	An@ -	An@(d)	An@(d, ix)*	xxx.W	xxx.L	PC@(d)	PC@(d, ix)*
JMP	–	8(2/0)	–	–	10(2/0)	14(3/0)	10(2/0)	12(3/0)	10(2/0)	14(3/0)
JSR	–	16(2/2)	–	–	18(2/2)	22(2/2)	18(2/2)	20(3/2)	18(2/2)	22(2/2)
LEA	–	4(1/0)	–	–	8(2/0)	12(2/0)	8(2/0)	12(3/0)	8(2/0)	12(2/0)
PEA	–	12(1/2)	–	–	16(2/2)	20(2/2)	16(2/2)	20(3/2)	16(2/2)	20(2/2)
MOVEM	Word	12+4n (3+n/0)	12+4n (3+n/0)	– –	16+4n (4+n/0)	18+4n (4+n/0)	16+4n (4+n/0)	20+4n (5+n/0)	16+4n (4+n/0)	18+4n (4+n/0)
M → R	Long	12+8n (3+2n/0)	12+8n (3+2n/0)	– –	16+8n (4+2n/0)	18+8n (4+2n/0)	16+8n (4+2n/0)	20+8n (5+2n/0)	16+8n (4+2n/0)	18+8n (4+2n/0)
MOVEM	Word	8+4n (2/n)	– –	8+4n (2/n)	12+4n (3/n)	14+4n (3/n)	12+4n (3/n)	16+4n (4/n)	– –	– –
R → M	Long	8+8n (2/2n)	– –	8+8n (2/2n)	12+8n (3/2n)	14+8n (3/2n)	12+8n (3/2n)	16+8n (4/2n)	– –	– –

n is the number of registers to move
* is the size of the index register (ix) does not affect the instruction's execution time

• MULTI-PRECISION INSTRUCTION CLOCK PERIODS

Table 37 indicates the number of clock periods for the multi-precision instructions. The number of clock periods includes the time to fetch both operands, perform the operations, store the results, and read the next instructions. The number of read and write cycles is shown in parenthesis as: (r/w).

In Table 37, the headings have the following meanings: Dn = data register operand and M = memory operand.

Table 37 Multi-Precision Instruction Clock Periods

Instruction	Size	op Dn, Dn	op M, M
ADDX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
CMPM	Byte, Word	–	12(3/0)
	Long	–	20(5/0)
SUBX	Byte, Word	4(1/0)	18(3/1)
	Long	8(1/0)	30(5/2)
ABCD	Byte	6(1/0)	18(3/1)
SBCD	Byte	6(1/0)	18(3/1)

• MISCELLANEOUS INSTRUCTION CLOCK PERIODS

Table 38 indicates the number of clock periods for the following miscellaneous instructions. The number of bus read and write cycles is shown in parenthesis as: (r/w). The number of clock periods plus the number of read and write cycles must be added to those of the effective address calculation where indicated.

• EXCEPTION PROCESSING CLOCK PERIODS

Table 39 indicates the number of clock periods for exception processing. The number of clock periods includes the time for all stacking, the vector fetch, and the fetch of the first instruction of the handler routine. The number of bus read and write cycles is shown in parenthesis as: (r/w).

Table 38 Miscellaneous Instruction Clock Periods

Instruction	Size	Register	Memory	Register → Memory	Memory → Register
MOVE from SR	–	6(1/0)	8(1/1) +	–	–
MOVE to CCR	–	12(2/0)	12(2/0) +	–	–
MOVE to SR	–	12(2/0)	12(2/0) +	–	–
MOVEP	Word	–	–	16(2/2)	16(4/0)
	Long	–	–	24(2/4)	24(6/0)
EXG	–	6(1/0)	–	–	–
EXT	Word	4(1/0)	–	–	–
	Long	4(1/0)	–	–	–
LINK	–	16(2/2)	–	–	–
MOVE from USP	–	4(1/0)	–	–	–
MOVE to USP	–	4(1/0)	–	–	–
NOP	–	4(1/0)	–	–	–
RESET	–	132(1/0)	–	–	–
RTE	–	20(5/0)	–	–	–
RTR	–	20(5/0	–	–	–
RTS	–	16(4/0)	–	–	–
STOP	–	4(0/0)	–	–	–
SWAP	–	4(1/0)	–	–	–
UNLK	–	12(3/0)	–	–	–

+ add effective address calculation time

Table 39 Exception Processing Clock Periods

Exception	Periods
Reset**	38.5 (6/0)
Address Error	50(4/7)
Bus Error	50(4/7)
Interrupt	44(5/3)*
Illegal Instruction	34(4/3)
Privileged Violation	34(4/3)
Trace	34(4/3)

* The interrupt acknowledge bus cycle is assumed to take four external clock periods.

** Indicates the time from when $\overline{RES}$ and $\overline{HALT}$ are first sampled as negated to when instruction execution starts.

■ **MASK VERSION**

Type No.	Mask version
HD68000-8 HD68000-10 HD68000-12	68000S1
HD68000Y-8 HD68000Y-10 HD68000Y-12	68000S1
HD68000P8 HD68000PS8 HD68000CP8	68000U

The difference of function between mask version 68000S1 and 68000U is only as following (Figure 59).

The function of HD68HC000 is as same as mask version 68000U.

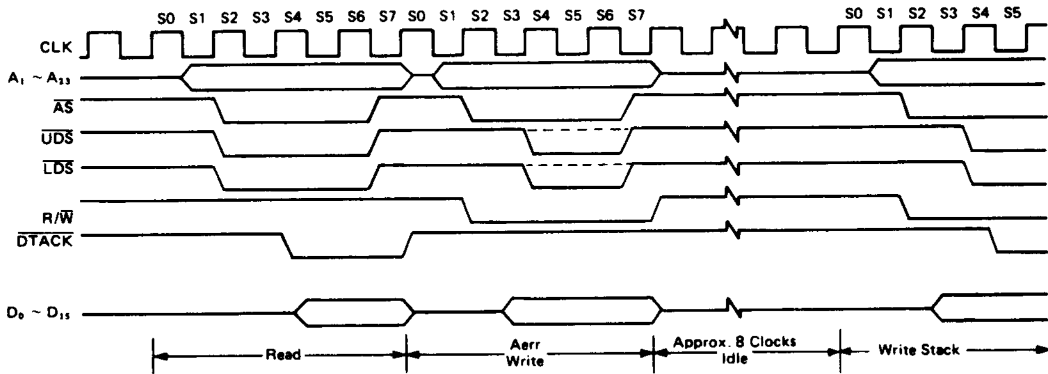
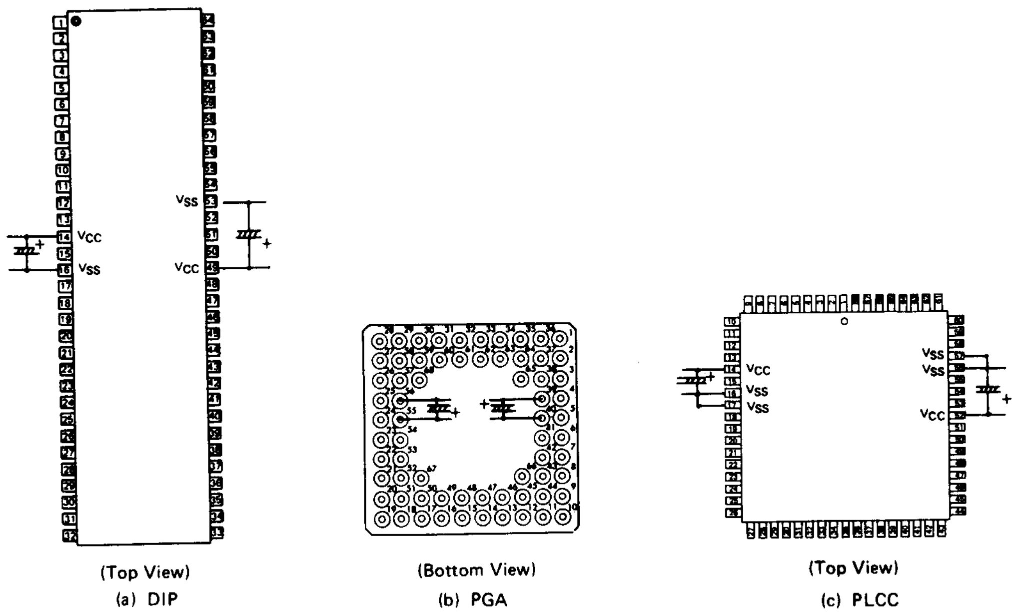


Figure 59 Address Error Timing

■ **NOTE FOR USE**

● **Power Supply Circuit**

When designing V_{CC} and V_{SS} pattern of the circuit board, the capacitors need to be located nearest to V_{CC} and V_{SS} as shown in the Figure 60.



(Top View)
(a) DIP

(Bottom View)
(b) PGA

(Top View)
(c) PLCC

1μF/35V Tantalum Capacitor (2 pairs)

Figure 60 Power Supply Circuit