Introduction to M68000 Microprocessor

Physics116B, 2/28/05

D. Pellett

References: Motorola literature, Wilkinson, Horowitz and Hill

Microprocessor Internal Structure

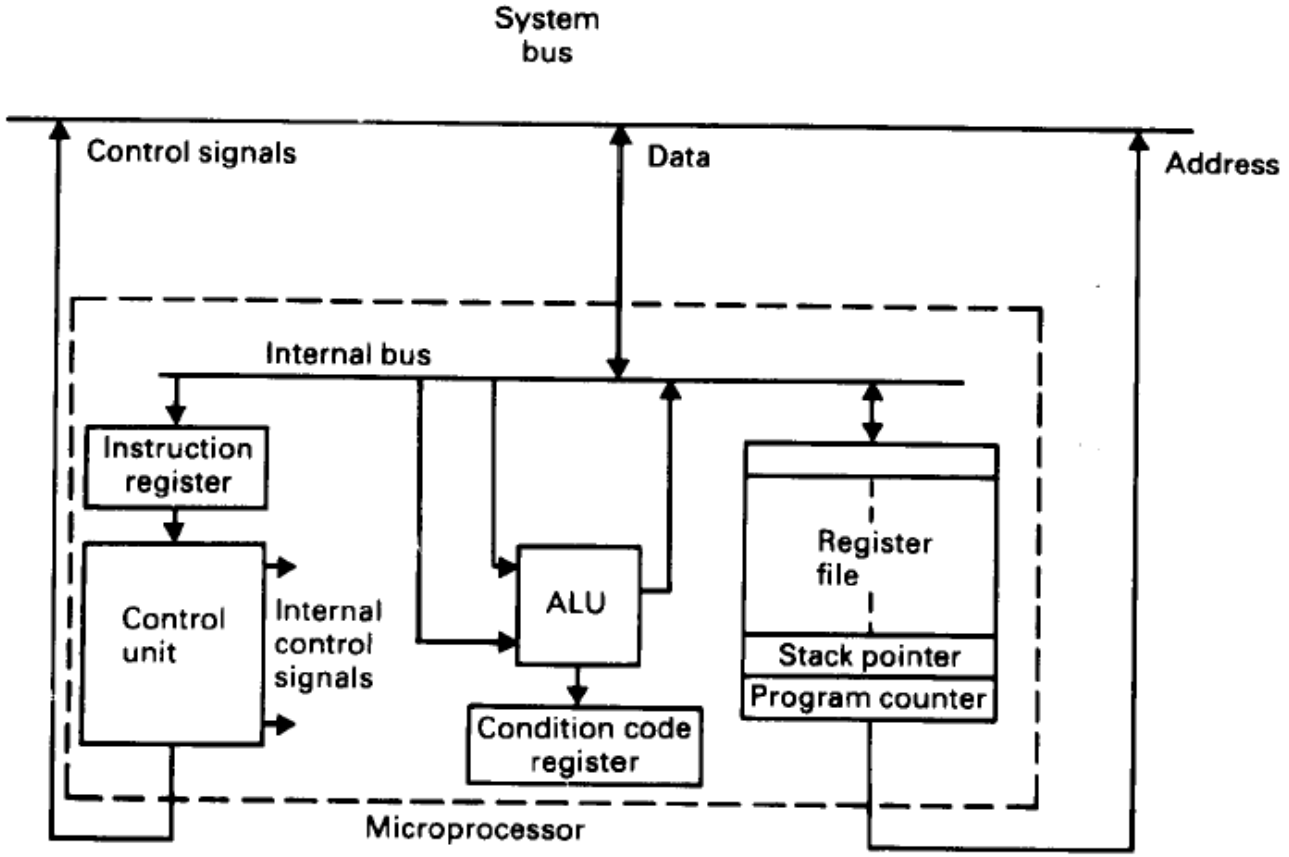


Figure 6.10 Internal architecture of simple microprocessor

- Typical of simple microprocessor such as M68000
- This part can be considered an elaborate finite state machine
- Particular instruction determines sequence or steps

Original M68000 Processor Family

MC68000, MC68HC000, MC68HC001, MC68008, MC68010, and MC68EC000 have

- 16 32-Bit Data and Address Registers
- 16-Mbyte Direct Addressing Range
- Program Counter
- 6 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory-Mapped Input/Output (I/O)
- 14 Addressing Modes

The following processors contain additional features:

- MC68010
  - —Virtual Memory/Machine Support
  - —High-Performance Looping Instructions
- MC68HC001/MC68EC000
  - —Statically Selectable 8- or 16-Bit Data Bus
- MC68HC000/MC68EC000/MC68HC001
  - —Low-Power

- MC68008 has an eight bit data bus, smaller address range.
- The MC68010 has a few additional instructions and instructions that operate differently than the corresponding instructions of the other devices.

M68000 as Hardware Device

Block Diagram

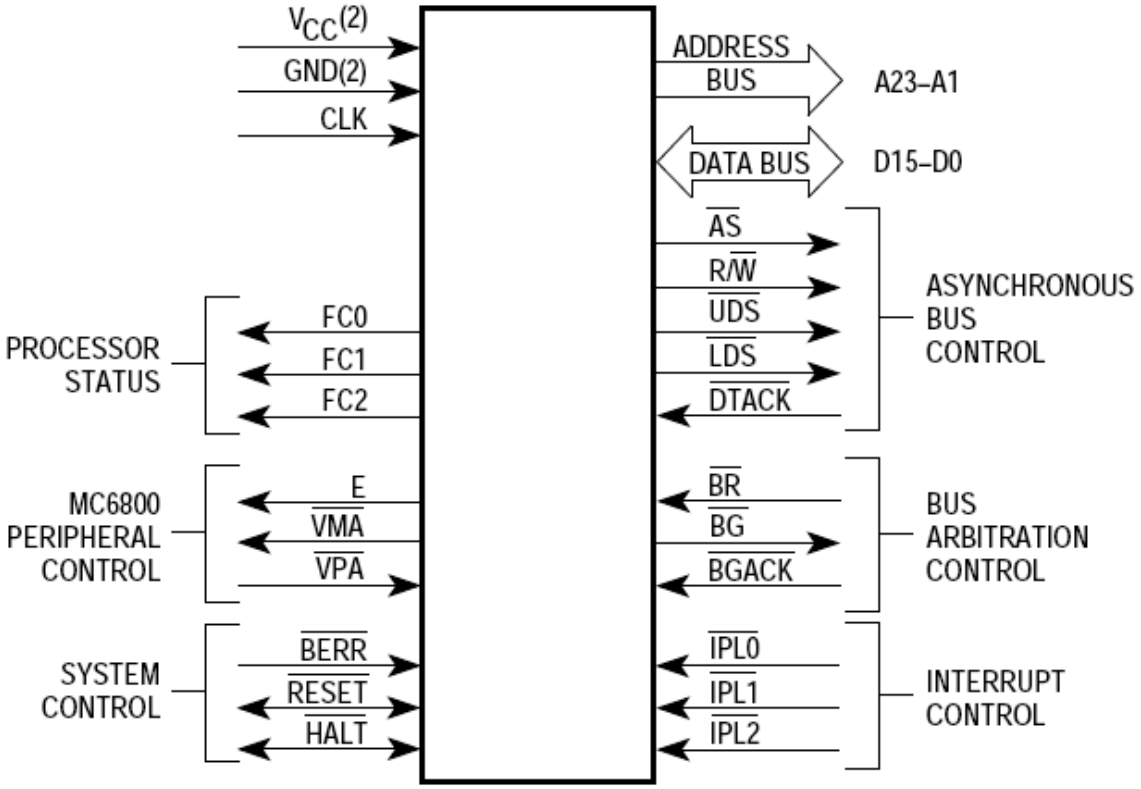


Figure 3-1. Input and Output Signals (MC68000, MC68HC000 and MC68010)

64-Pin DIP Pinouts

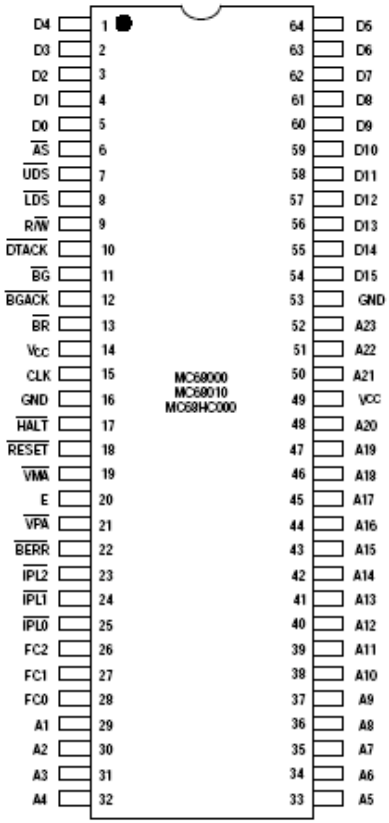


Figure 11-1. 64-Pin Dual In Line

Figures from M68000 Microprocessors User's Manual 9th Ed., © 1993 Motorola Inc.

Table 3-4. Signal Summary

Signal Name	Mnemonic	Input/Output	Active State	HI-Z	
				On $\overline{\text{HALT}}$	On Bus Relinquish
Address Bus	A0–A23	Output	High	Yes	Yes
Data Bus	D0–D15	Input/Output	High	Yes	Yes
Address Strobe	$\overline{\text{AS}}$	Output	Low	No	Yes
Read/Write	R/$\overline{\text{W}}$	Output	Read-High Write-Low	No	Yes
Data Strobe	$\overline{\text{DS}}$	Output	Low	No	Yes
Upper and Lower Data Strobes	$\overline{\text{UDS}}$, $\overline{\text{LDS}}$	Output	Low	No	Yes
Data Transfer Acknowledge	$\overline{\text{DTACK}}$	Input	Low	No	No
Bus Request	$\overline{\text{BR}}$	Input	Low	No	No
Bus Grant	$\overline{\text{BG}}$	Output	Low	No	No
Bus Grant Acknowledge	$\overline{\text{BGACK}}$	Input	Low	No	No
Interrupt Priority Level	$\overline{\text{IPL}}$0, $\overline{\text{IPL}}$1, $\overline{\text{IPL}}$2	Input	Low	No	No
Bus Error	$\overline{\text{BERR}}$	Input	Low	No	No
Mode	MODE	Input	High	—	—
Reset	$\overline{\text{RESET}}$	Input/Output	Low	No*	No*
Halt	$\overline{\text{HALT}}$	Input/Output	Low	No*	No*
Enable	E	Output	High	No	No
Valid Memory Address	$\overline{\text{VMA}}$	Output	Low	No	Yes
Valid Peripheral Address	$\overline{\text{VPA}}$	Input	Low	No	No
Function Code Output	FC0, FC1, FC2	Output	High	No	Yes
Clock	CLK	Input	High	No	No
Power Input	V_{CC}	Input	—	—	—
Ground	GND	Input	—	—	—

*Open drain.

M68000 Internal Registers

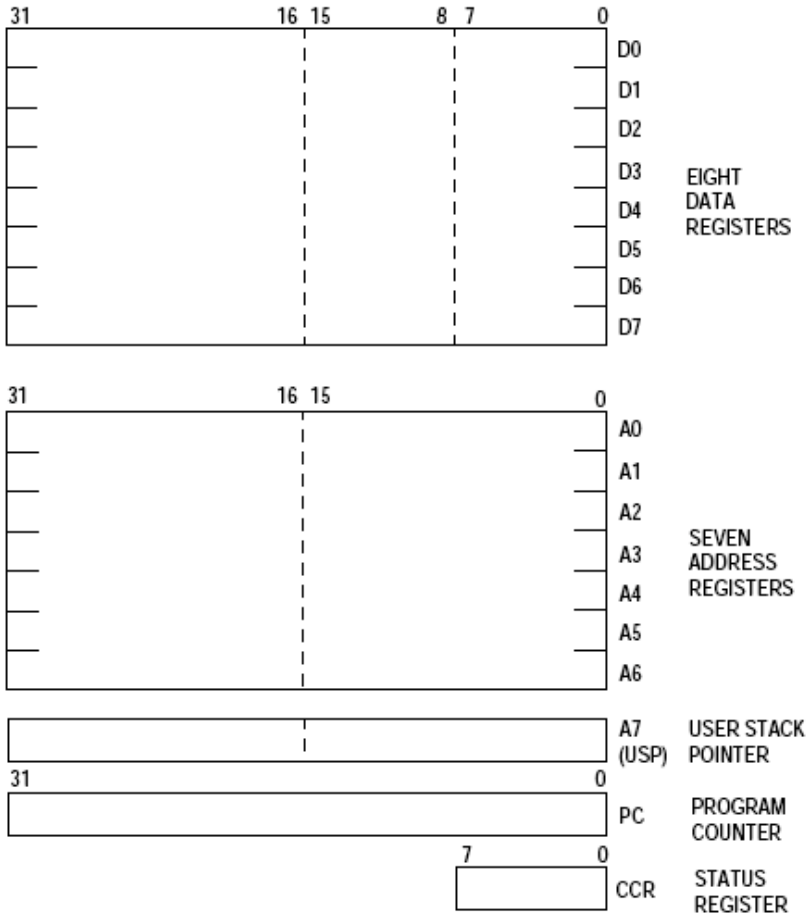
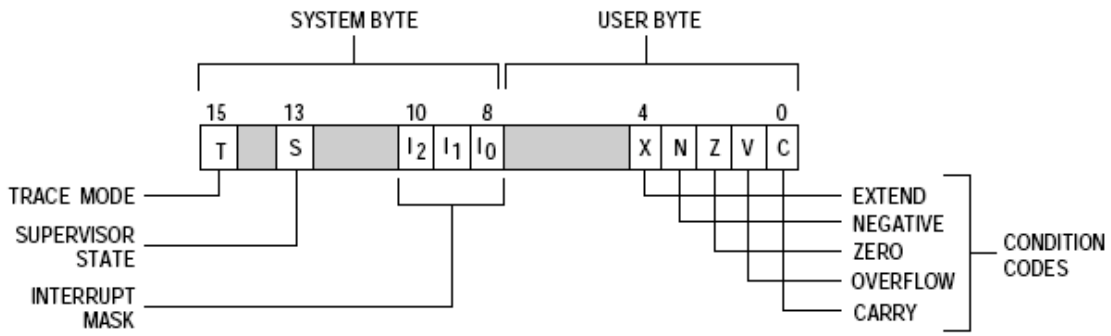


Figure 2-1. User Programmer's Model (MC68000/MC68HC000/MC68008/MC68010)

From M68000 Microprocessors User's Manual 9th Ed., © 1993 Motorola Inc.

Status Register



- Condition code bits are used for conditional branch instructions
- Set or cleared by certain instructions
- Used to make if-then-else programming constructs
- Used to extend registers for arithmetic operations

From M68000 Microprocessors User's Manual 9th Ed., © 1993 Motorola Inc.

Data Organization in Memory

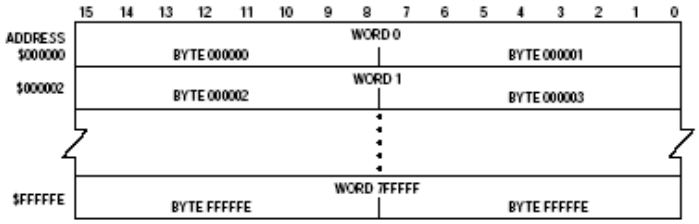
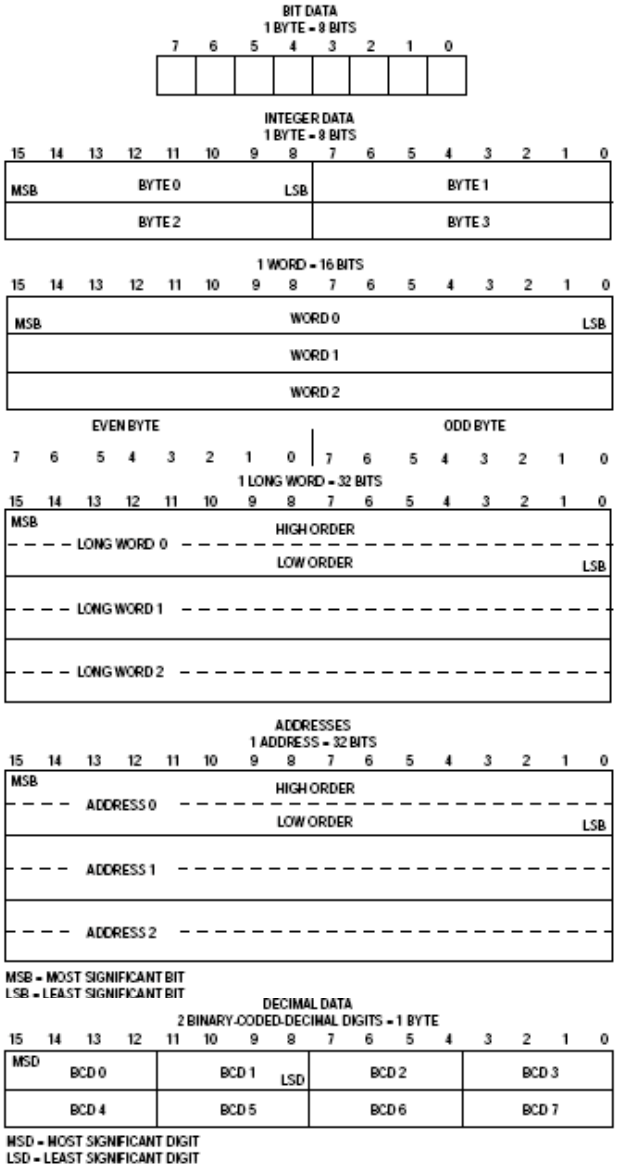


Figure 2-5. Word Organization in Memory

- Bytes are individually addressable
- High-order byte has same address a word
- Low-order byte has odd address, one count higher.
- Instructions and multibyte data accessed only on word (even byte) boundaries.
  If a long-word operand is located at address n (n even), then the second word of that operand is located at address n+2.

Data types supported by M68000:
bit data; integer data of 8, 16, and
32 bits; 32-bit addresses; binary-coded-decimal data



From M68000 Microprocessors User's Manual 9th Ed., © 1993 Motorola Inc.

Data Addressing Modes for Instructions

Table 2-1. Data Addressing Modes

Mode	Generation	Syntax
Register Direct Addressing		
Data Register Direct	EA=Dn	Dn
Address Register Direct	EA=An	An
Absolute Data Addressing		
Absolute Short	EA = (Next Word)	(xxx).W
Absolute Long	EA = (Next Two Words)	(xxx).L
Program Counter Relative Addressing		
Relative with Offset	EA = (PC)+d_{16}	(d_{16},PC)
Relative with Index and Offset	EA = (PC)+d_8	(d_8,PC,Xn)
Register Indirect Addressing		
Register Indirect	EA = (An)	(An)
Postincrement Register Indirect	EA = (An), An ← An+N	(An)+
Predecrement Register Indirect	An ♦ An–N, EA=(An)	-(An)
Register Indirect with Offset	EA = (An)+d_{16}	(d_{16},An)
Indexed Register Indirect with Offset	EA = (An)+(Xn)+d_8	(d_8,An,Xn)
Immediate Data Addressing		
Immediate	DATA = Next Word(s)	#<data>
Quick Immediate	Inherent Data	
Implied Addressing[1]		
Implied Register	EA = SR, USP, SSP, PC, VBR, SFC, DFC	SR,USP,SSP,PC, VBR, SFC,DFC

From M68000 Microprocessors User's Manual 9th Ed., © 1993 Motorola Inc.

Instruction Set Summary (next 4 Pages)

Opcode	Operation	Syntax
ABCD	$Source_{10} + Destination_{10} + X \rightarrow Destination$	ABCD Dy,Dx ABCD −(Ay), −(Ax)
ADD	Source + Destination → Destination	ADD <ea>,Dn ADD Dn,<ea>
ADDA	Source + Destination → Destination	ADDA <ea>,An
ADDI	Immediate Data + Destination → Destination	ADDI # <data>,<ea>
ADDQ	Immediate Data + Destination → Destination	ADDQ # <data>,<ea>
ADDX	Source + Destination + X → Destination	ADDX Dy, Dx ADDX −(Ay), −(Ax)
AND	Source Λ Destination → Destination	AND <ea>,Dn AND Dn,<ea>
ANDI	Immediate Data Λ Destination → Destination	ANDI # <data>, <ea>
ANDI to CCR	Source Λ CCR → CCR	ANDI # <data>, CCR
ANDI to SR	If supervisor state then Source Λ SR → SR else TRAP	ANDI # <data>, SR
ASL, ASR	Destination Shifted by <count> → Destination	ASd Dx,Dy ASd # <data>,Dy ASd <ea>
Bcc	If (condition true) then PC + d → PC	Bcc <label>
BCHG	~ (<number> of Destination) → Z; ~ (<number> of Destination) → <bit number> of Destination	BCHG Dn,<ea> BCHG # <data>,<ea>
BCLR	~ (<bit number> of Destination) → Z; 0 → <bit number> of Destination	BCLR Dn,<ea> BCLR # <data>,<ea>
BKPT	Run breakpoint acknowledge cycle; TRAP as illegal instruction	BKPT # <data>
BRA	PC + d → PC	BRA <label>
BSET	~ (<bit number> of Destination) → Z; 1 → <bit number> of Destination	BSET Dn,<ea> BSET # <data>,<ea>
BSR	SP − 4 → SP; PC → (SP); PC + d → PC	BSR <label>
BTST	− (<bit number> of Destination) → Z;	BTST Dn,<ea> BTST # <data>,<ea>
CHK	If Dn < 0 or Dn > Source then TRAP	CHK <ea>,Dn
CLR	0 → Destination	CLR <ea>
CMP	Destination − Source → cc	CMP <ea>,Dn
CMPA	Destination − Source	CMPA <ea>,An
CMPI	Destination − Immediate Data	CMPI # <data>,<ea>
CMPM	Destination − Source → cc	CMPM (Ay)+, (Ax)+
DBcc	If condition false then (Dn − 1 → Dn; If Dn ≠ −1 then PC + d → PC)	DBcc Dn,<label>

Opcode	Operation	Syntax
DIVS	Destination/Source → Destination	DIVS.W <ea>,Dn 32/16 → 16r:16q
DIVU	Destination/Source → Destination	DIVU.W <ea>,Dn 32/16 → 16r:16q
EOR	Source ⊕ Destination → Destination	EOR Dn,<ea>
EORI	Immediate Data ⊕ Destination → Destination	EORI # <data>,<ea>
EORI to CCR	Source ⊕ CCR → CCR	EORI # <data>,CCR
EORI to SR	If supervisor state then Source ⊕SR → SR else TRAP	EORI # <data>,SR
EXG	Rx ↔ Ry	EXG Dx,Dy EXG Ax,Ay EXG Dx,Ay EXG Ay,Dx
EXT	Destination Sign-Extended → Destination	EXT.W Dn extend byte to word EXT.L Dn extend word to long word
ILLEGAL	SSP – 2 → SSP; Vector Offset → (SSP); SSP – 4 → SSP; PC → (SSP); SSP – 2 → SSP; SR → (SSP); Illegal Instruction Vector Address → PC	ILLEGAL
JMP	Destination Address → PC	JMP <ea>
JSR	SP – 4 → SP; PC → (SP) Destination Address → PC	JSR <ea>
LEA	<ea> → An	LEA <ea>,An
LINK	SP – 4 → SP; An → (SP) SP → An, SP + d → SP	LINK An, # <displacement>
LSL,LSR	Destination Shifted by <count> → Destination	LSd[1] Dx,Dy LSd[1] # <data>,Dy LSd[1] <ea>
MOVE	Source → Destination	MOVE <ea>,<ea>
MOVEA	Source → Destination	MOVEA <ea>,An
MOVE from CCR	CCR → Destination	MOVE CCR,<ea>
MOVE to CCR	Source → CCR	MOVE <ea>,CCR
MOVE from SR	SR → Destination If supervisor state then SR → Destination else TRAP (MC68010 only)	MOVE SR,<ea>
MOVE to SR	If supervisor state then Source → SR else TRAP	MOVE <ea>,SR

Opcode	Operation	Syntax
MOVE USP	If supervisor state then USP → An or An → USP else TRAP	MOVE USP,An MOVE An,USP
MOVEC	If supervisor state then Rc → Rn or Rn → Rc else TRAP	MOVEC Rc,Rn MOVEC Rn,Rc
MOVEM	Registers → Destination Source → Registers	MOVEM register list,<ea> MOVEM <ea>,register list
MOVEP	Source → Destination	MOVEP Dx,(d,Ay) MOVEP (d,Ay),Dx
MOVEQ	Immediate Data → Destination	MOVEQ # <data>,Dn
MOVES	If supervisor state then Rn → Destination [DFC] or Source [SFC] → Rn else TRAP	MOVES Rn,<ea> MOVES <ea>,Rn
MULS	Source × Destination → Destination	MULS.W <ea>,Dn 16 x 16 → 32
MULU	Source × Destination → Destination	MULU.W <ea>,Dn 16 x 16 → 32
NBCD	0 – (Destination$_{10}$) – X → Destination	NBCD <ea>
NEG	0 – (Destination) → Destination	NEG <ea>
NEGX	0 – (Destination) – X → Destination	NEGX <ea>
NOP	None	NOP
NOT	~Destination → Destination	NOT <ea>
OR	Source V Destination → Destination	OR <ea>,Dn OR Dn,<ea>
ORI	Immediate Data V Destination → Destination	ORI # <data>,<ea>
ORI to CCR	Source V CCR → CCR	ORI # <data>,CCR
ORI to SR	If supervisor state then Source V SR → SR else TRAP	ORI # <data>,SR
PEA	Sp – 4 → SP; <ea> → (SP)	PEA <ea>
RESET	If supervisor state then Assert $\overline{\text{RESET}}$ Line else TRAP	RESET
ROL, ROR	Destination Rotated by <count> → Destination	ROd[1] Rx,Dy ROd[1] # <data>,Dy ROd[1] <ea>
ROXL, ROXR	Destination Rotated with X by <count> → Destination	ROXd[1] Dx,Dy ROXd[1] # <data>,Dy ROXd[1] <ea>
RTD	(SP) → PC; SP + 4 + d → SP	RTD #<displacement>

Opcode	Operation	Syntax
RTE	If supervisor state then (SP) → SR; SP + 2 → SP; (SP) → PC; SP + 4 → SP; restore state and deallocate stack according to (SP) else TRAP	RTE
RTR	(SP) → CCR; SP + 2 → SP; (SP) → PC; SP + 4 → SP	RTR
RTS	(SP) → PC; SP + 4 → SP	RTS
SBCD	$Destination_{10} - Source_{10} - X \rightarrow Destination$	SBCD Dx,Dy SBCD −(Ax),−(Ay)
Scc	If condition true then 1s → Destination else 0s → Destination	Scc <ea>
STOP	If supervisor state then Immediate Data → SR; STOP else TRAP	STOP # <data>
SUB	Destination − Source → Destination	SUB <ea>,Dn SUB Dn,<ea>
SUBA	Destination − Source → Destination	SUBA <ea>,An
SUBI	Destination − Immediate Data → Destination	SUBI # <data>,<ea>
SUBQ	Destination − Immediate Data → Destination	SUBQ # <data>,<ea>
SUBX	Destination − Source − X → Destination	SUBX Dx,Dy SUBX −(Ax),−(Ay)
SWAP	Register [31:16] ↔ Register [15:0]	SWAP Dn
TAS	Destination Tested → Condition Codes; 1 → bit 7 of Destination	TAS <ea>
TRAP	SSP − 2 → SSP; Format/Offset → (SSP); SSP − 4 → SSP; PC → (SSP); SSP−2 → SSP; SR → (SSP); Vector Address → PC	TRAP # <vector>
TRAPV	If V then TRAP	TRAPV
TST	Destination Tested → Condition Codes	TST <ea>
UNLK	An → SP; (SP) → An; SP + 4 → SP	UNLK An

NOTE: d is direction, L or R.

Assembly Language,MAS system and Macs

- See MAS Manual for lab computers
- Assembly language includes
    - Mnemonics for machine instructions
    - Directives for assembler itself
- Macintosh system uses certain conventions to allow "relocatable" instructions
    - Data in separate area
    - Code addresses relative to PC
    - Data addresses relative to A5
- MAS system has subroutines to access Macintosh keyboard and screen
- MAS system allows assembly, running and debugging of code