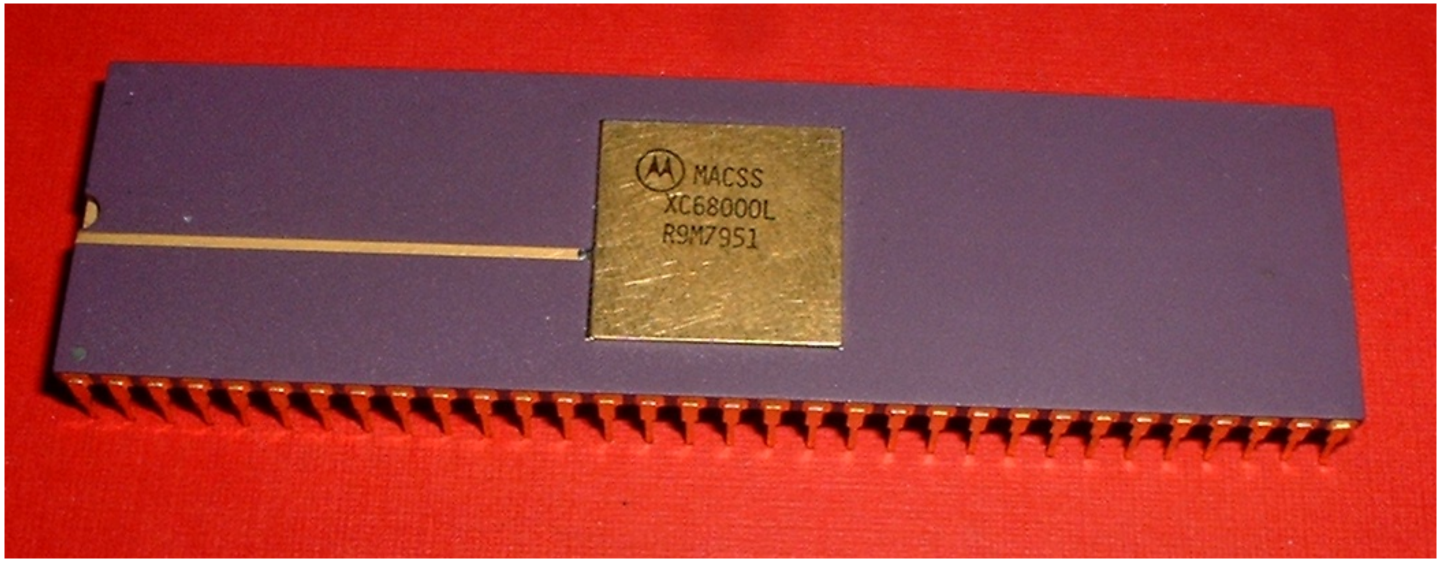
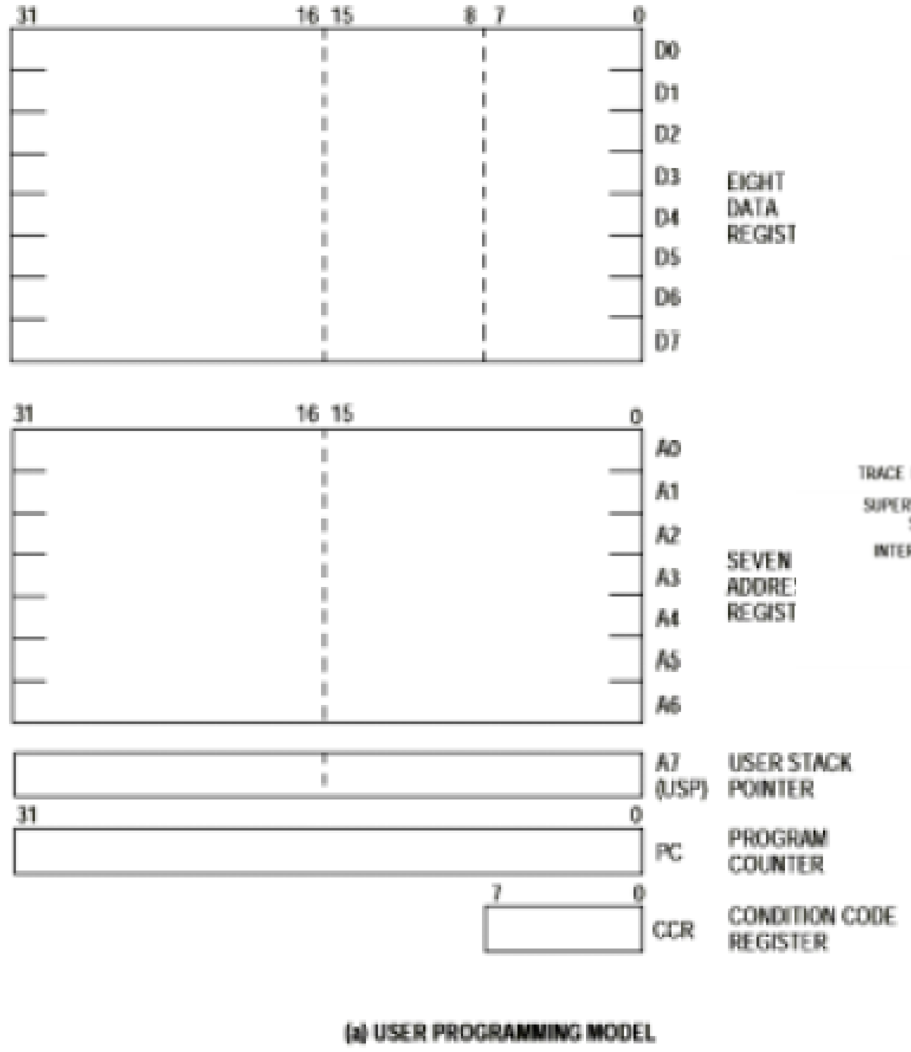


MC680000 Microprocessor Hardware

Register set



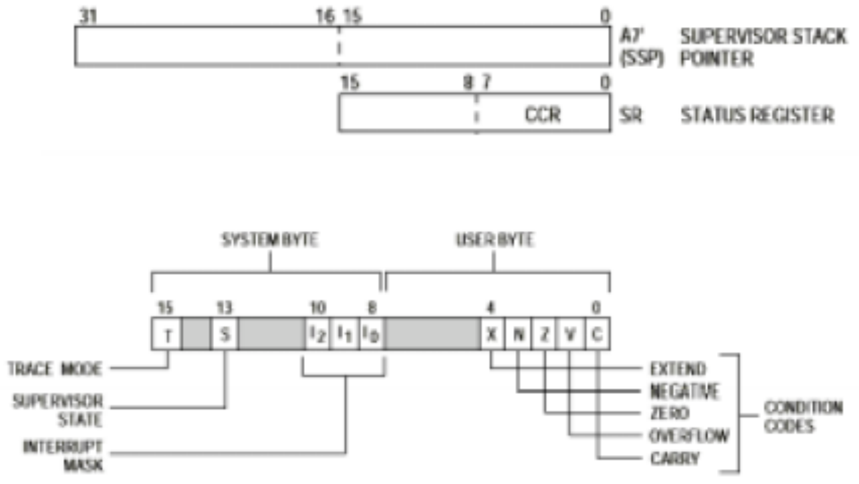


Figure 1-4. Status Register

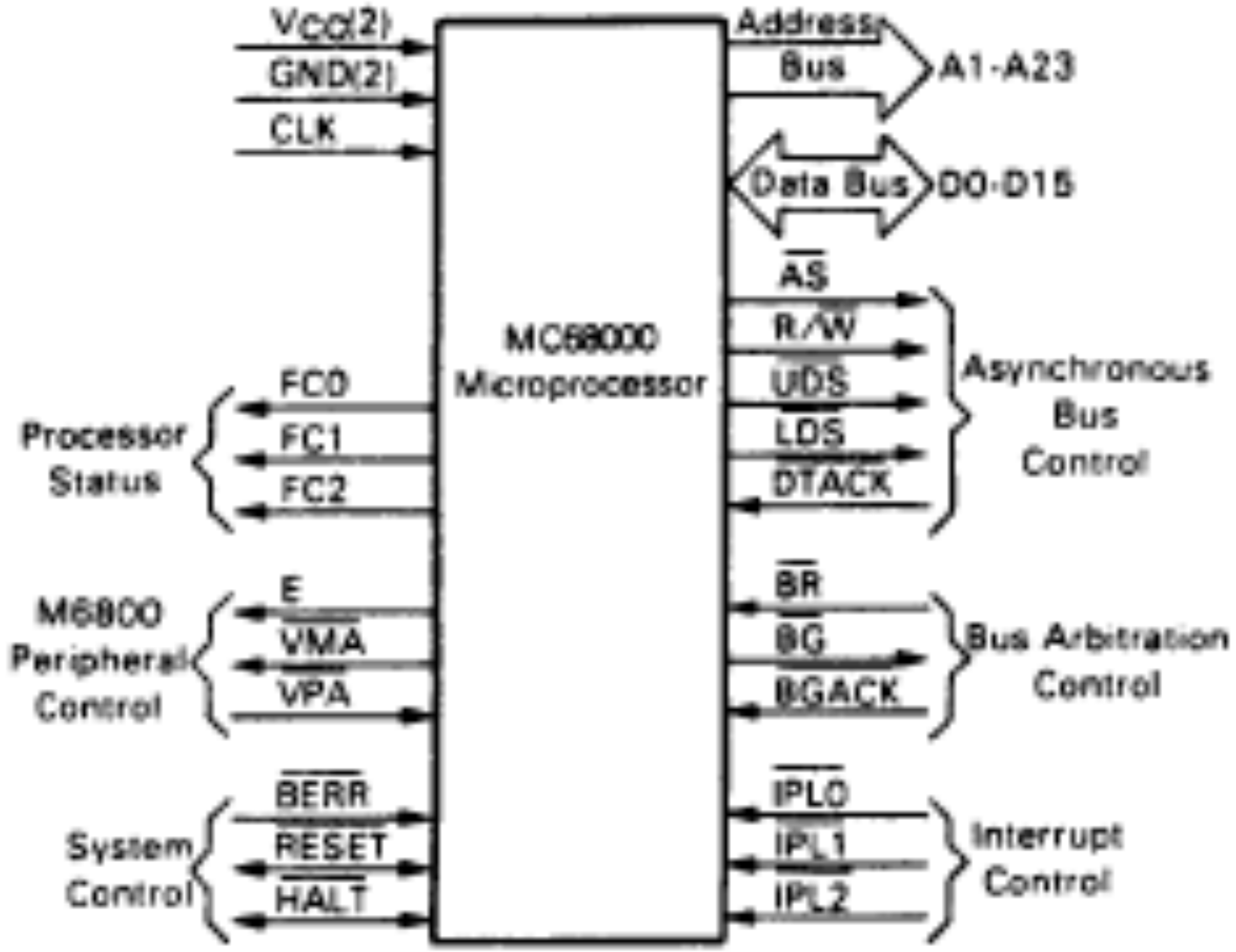
Privilege Levels

Privilege levels

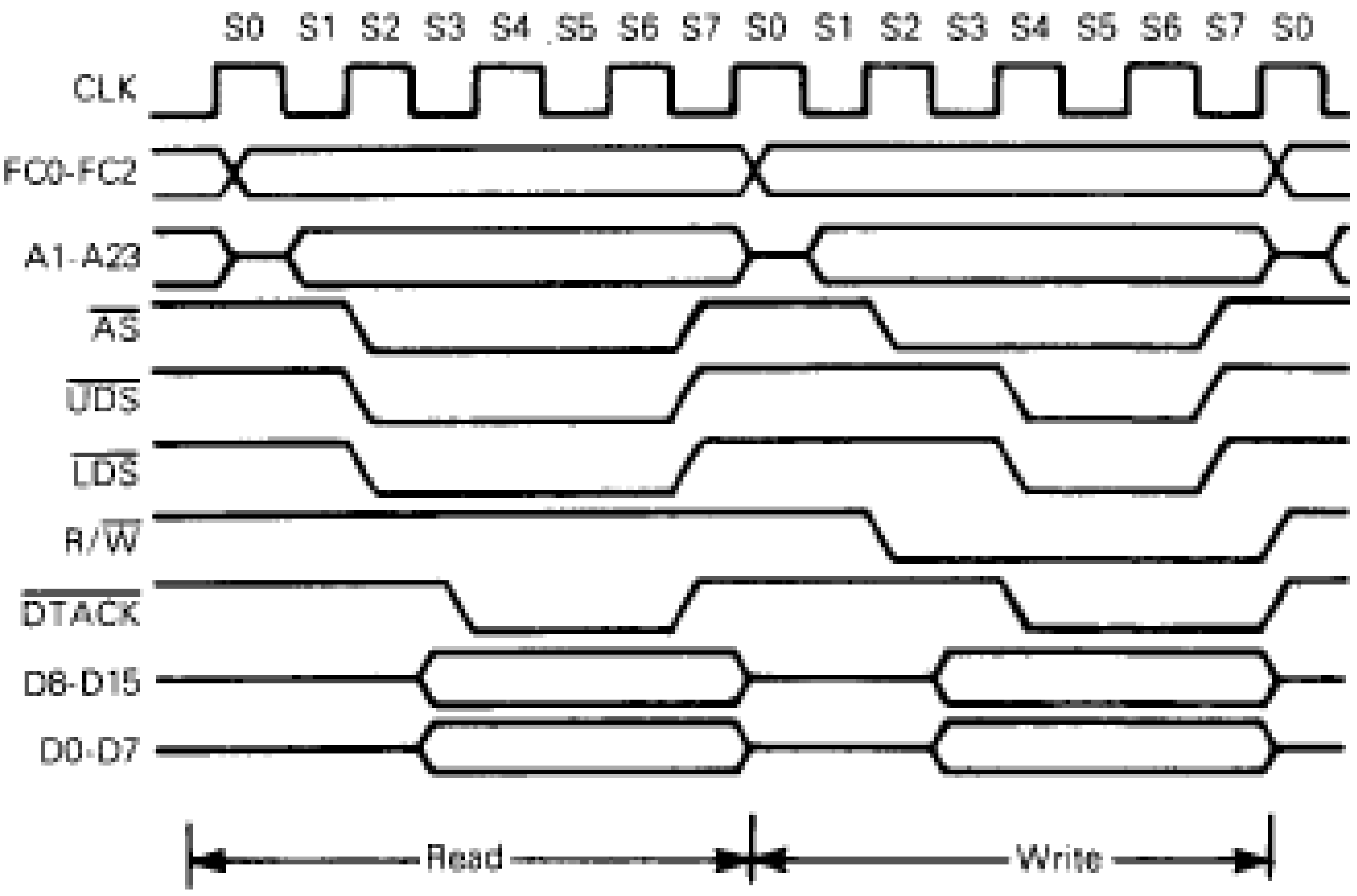
The CPU, and later the whole family, implemented exactly two levels of privilege. User mode gave access to everything except the interrupt level control. Supervisor privilege gave access to everything. An interrupt always became supervisory. The supervisor bit was stored in the status register, and visible to user programs.

A real advantage of this system was that the supervisor level had a separate stack pointer. This permitted a multitasking system to use very small stacks for tasks, because the designers did not have to allocate the memory required to hold the stack frames of a maximum stack-up of interrupts.

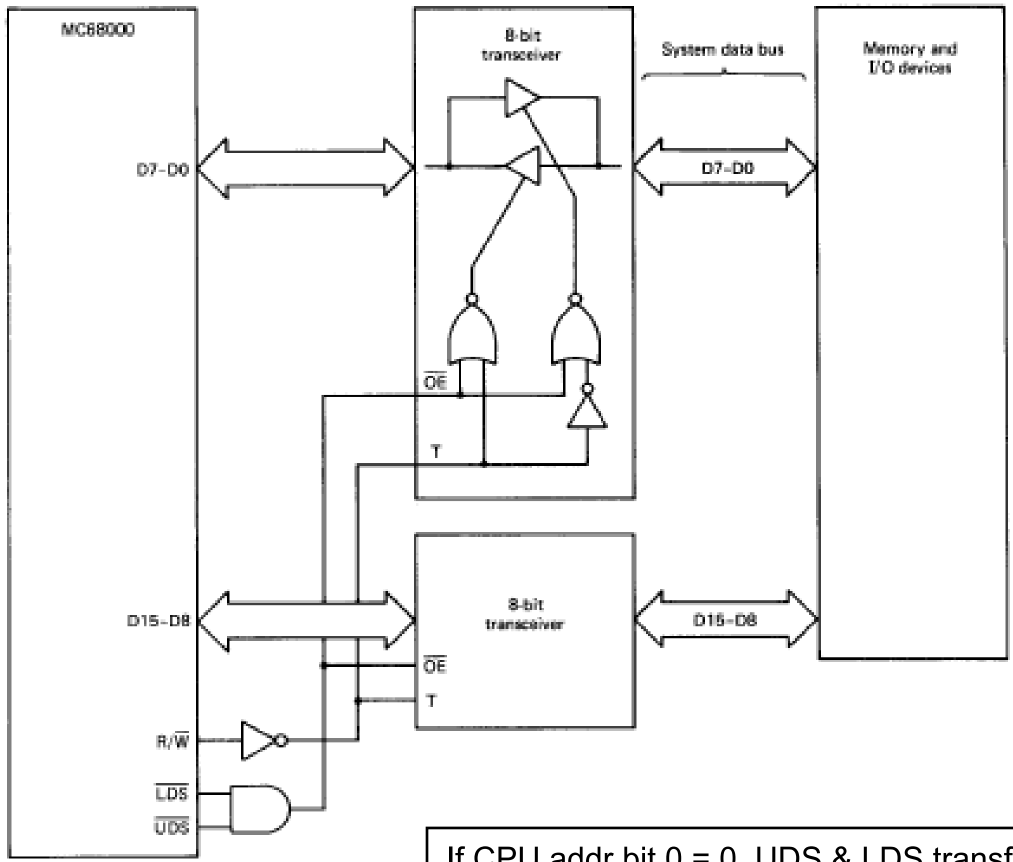
Functional pin layouts



Rd/Wr cycle timing

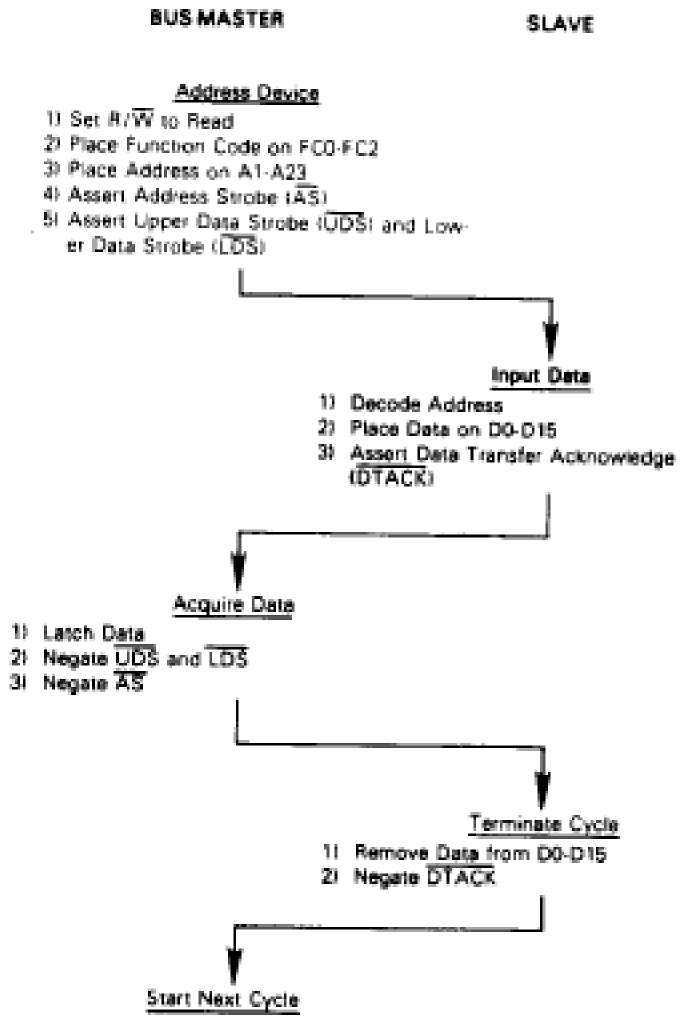


Data bus buffering

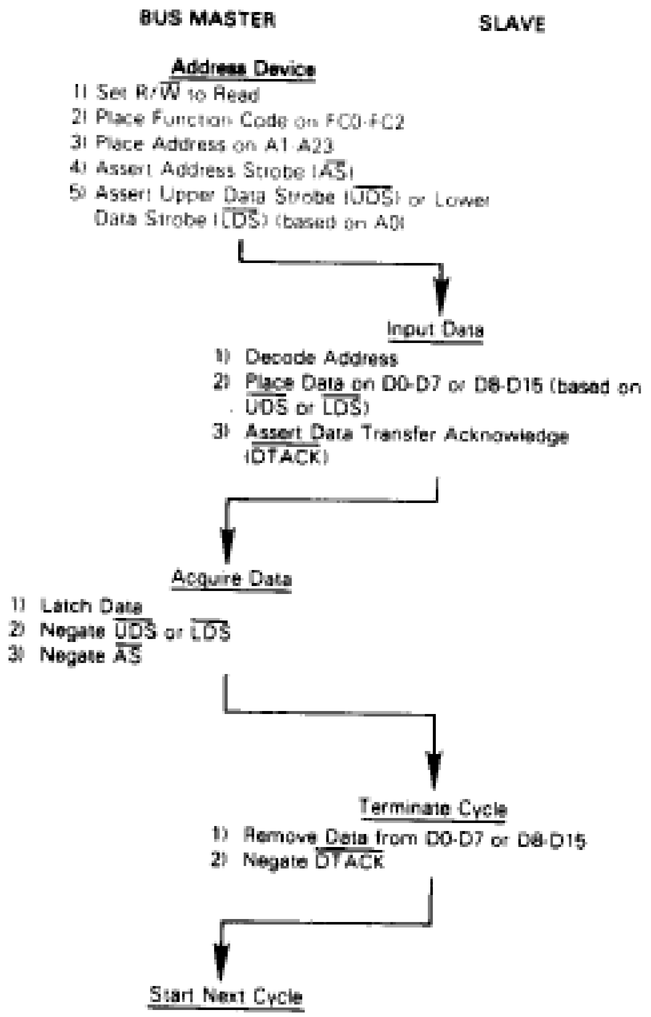


If CPU addr bit 0 = 0, UDS & LDS transfer a 16-bit word on D15 – D0
If CPU addr bit 0 = 1, UDS & LDS transfer a byte on D7 – D0

Bus cycle control flow

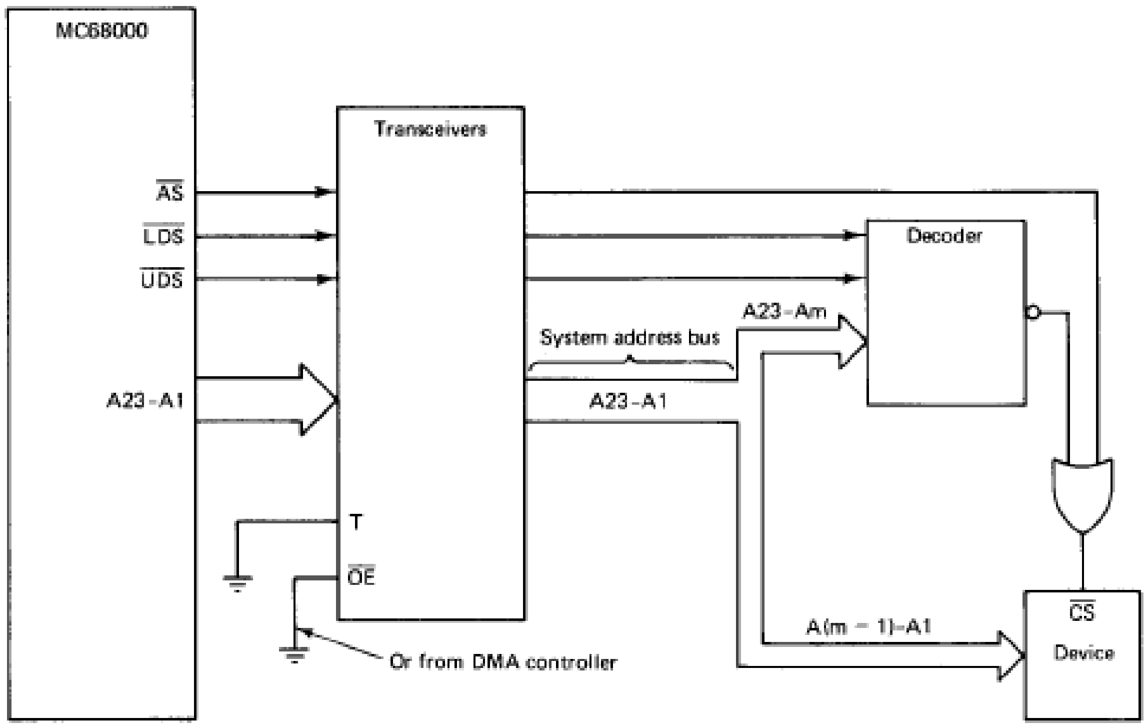

BUS MASTER
SLAVE
Address Device
1) Set R/$\overline{W}$ to Read
2) Place Function Code on FC0-FC2
3) Place Address on A1-A23
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
Input Data
1) Decode Address
2) Place Data on D0-D15
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Acquire Data
1) Latch Data
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$
Terminate Cycle
1) Remove Data from D0-D15
2) Negate $\overline{DTACK}$
Start Next Cycle

(a) Word read cycle flow chart

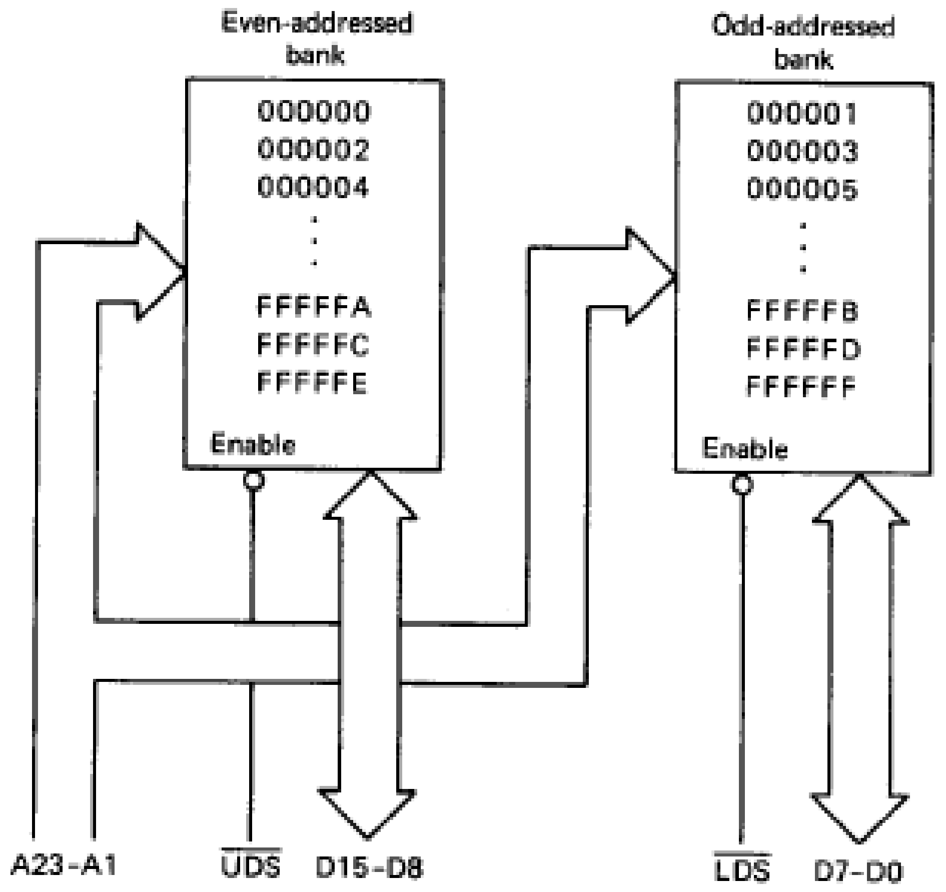

BUS MASTER
SLAVE
Address Device
1) Set R/$\overline{W}$ to Read
2) Place Function Code on FC0-FC2
3) Place Address on A1-A23
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) or Lower Data Strobe ($\overline{LDS}$) (based on A0)
Input Data
1) Decode Address
2) Place Data on D0-D7 or D8-D15 (based on $\overline{UDS}$ or $\overline{LDS}$)
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Acquire Data
1) Latch Data
2) Negate $\overline{UDS}$ or $\overline{LDS}$
3) Negate $\overline{AS}$
Terminate Cycle
1) Remove Data from D0-D7 or D8-D15
2) Negate $\overline{DTACK}$
Start Next Cycle

(b) Byte read cycle flow chart

Address bus interface



Physical implementation



Read bus with wait states

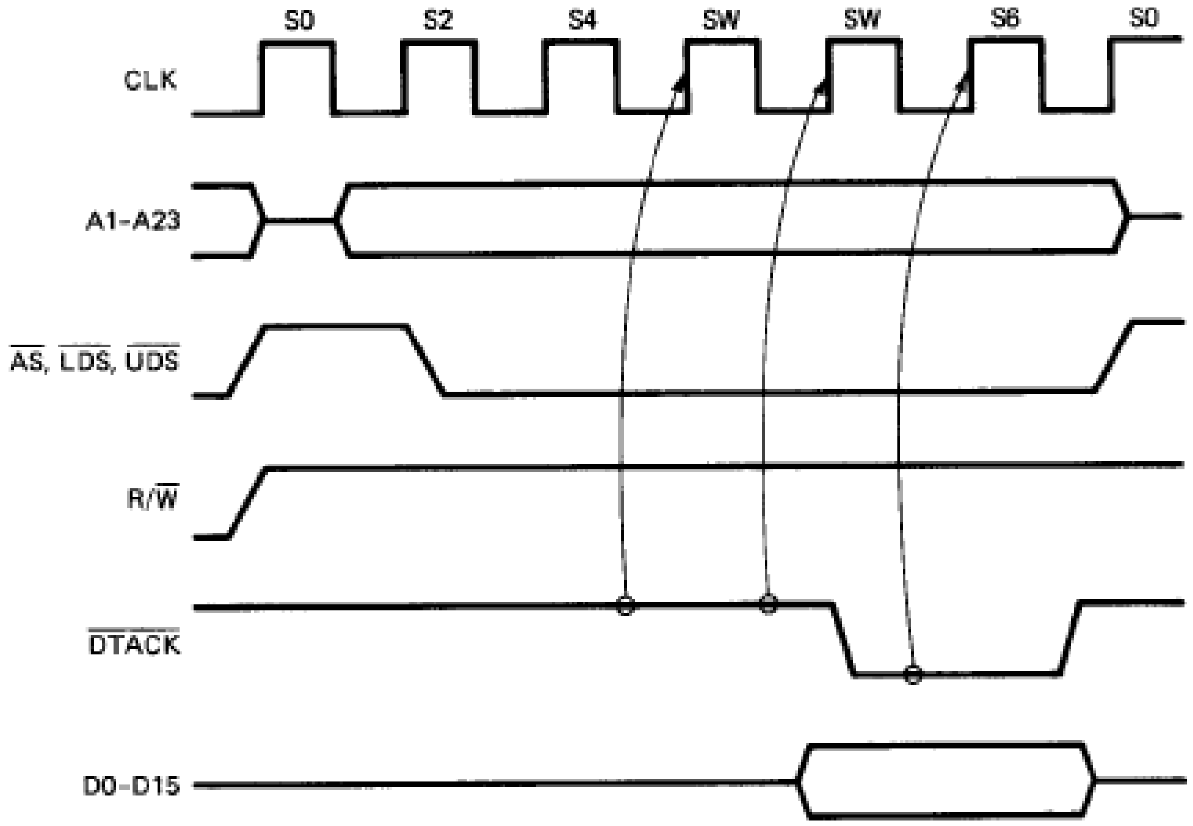
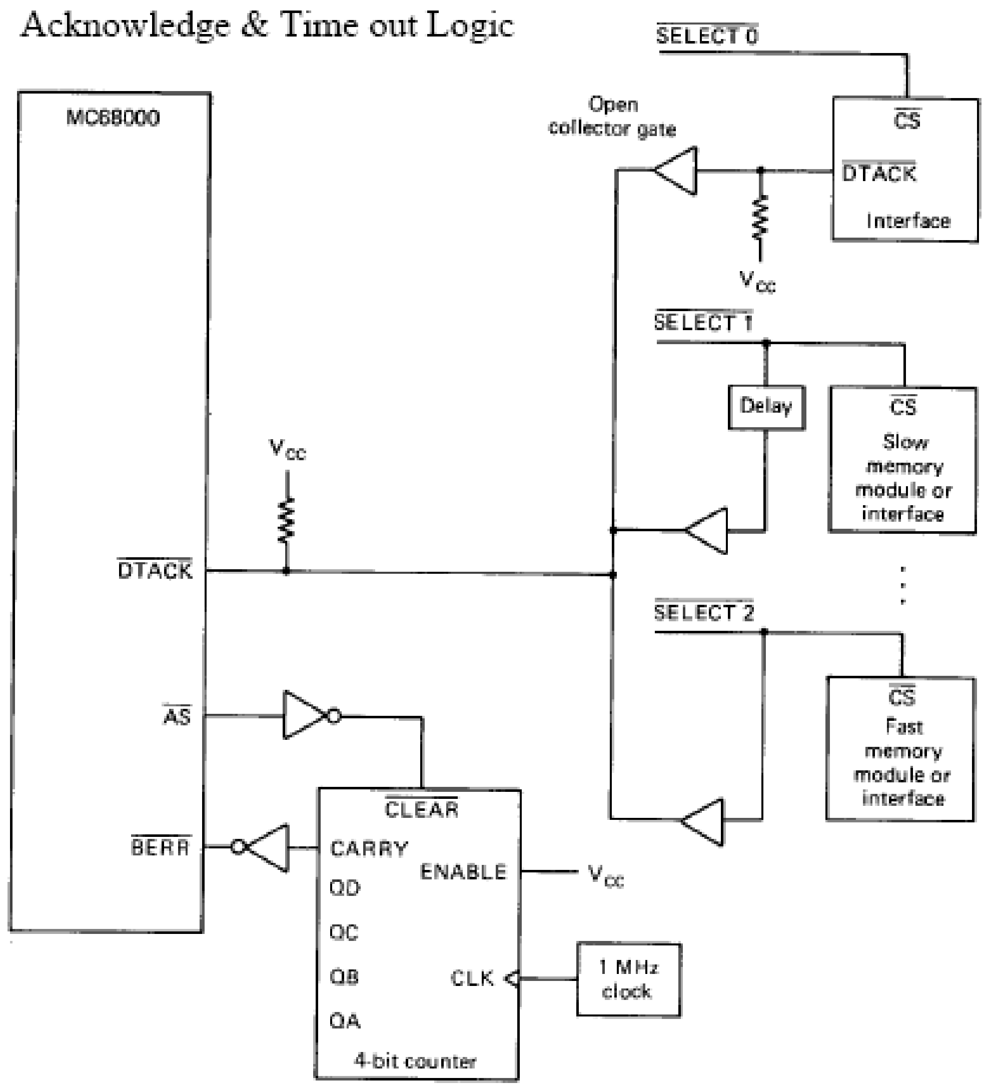
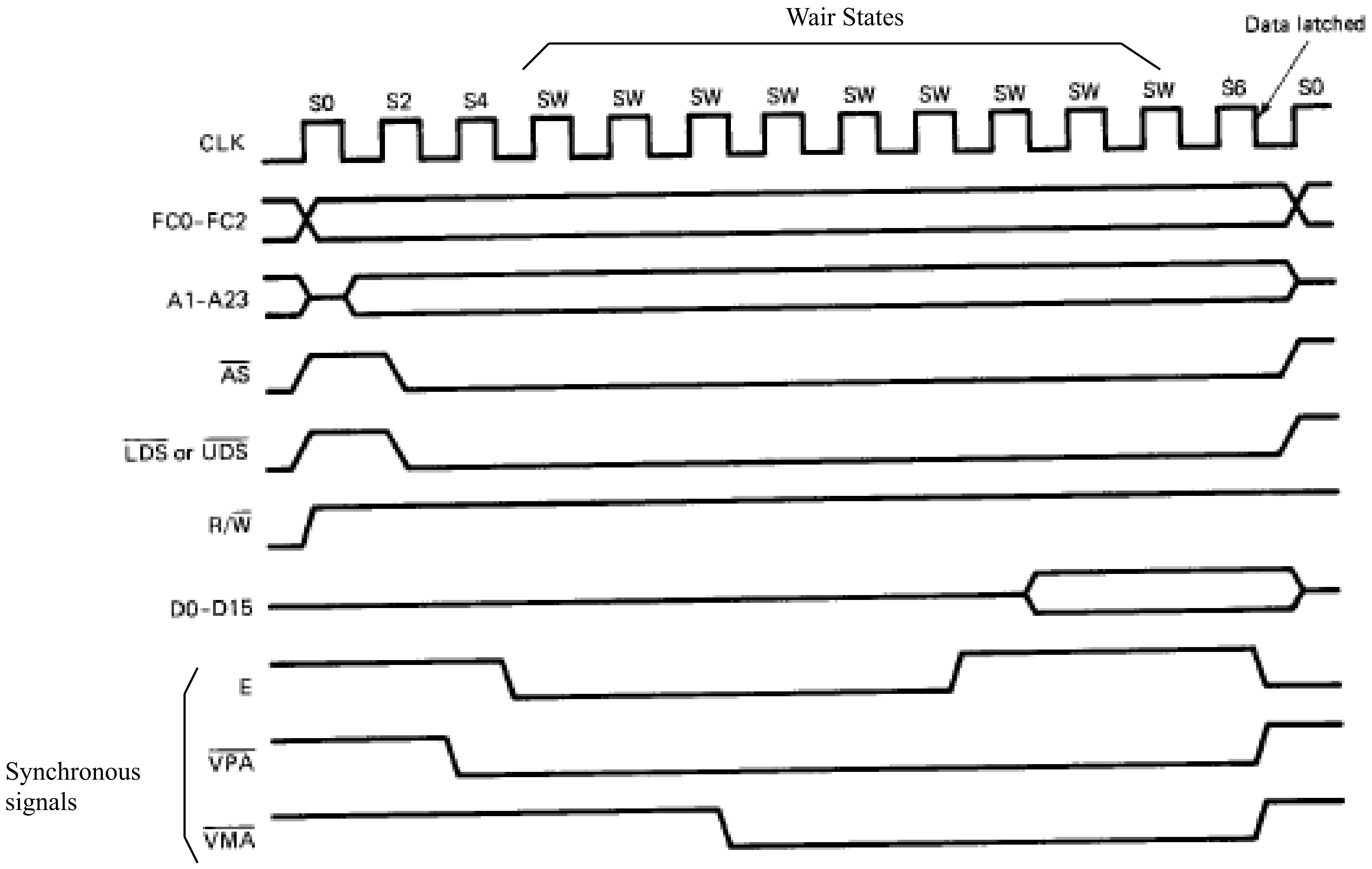


Figure 7-10 Timing of a read bus cycle with wait states.

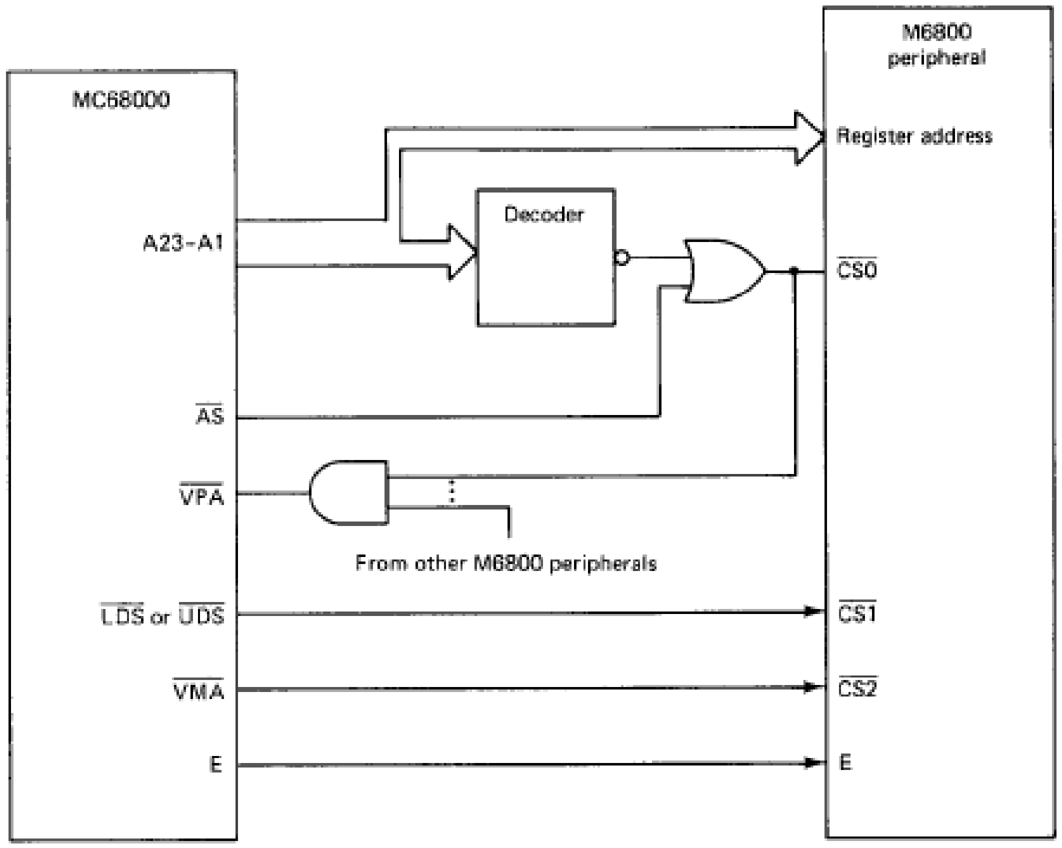
Ack & watchdog timer



Synchronous bus cycle timing

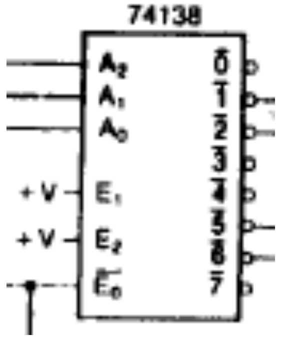


Interfacing peripherals



RAM/ROM interface example

Using 6836 ROM (2^{14} x 8) and NEC 43256 SRAM (2^{15} x 8), design a memory interface for a 68000 processor system.



<u>CPU addresses</u>

$000000 - $007FFF ➔ 2^{15} = 32Kbytes ROM
$010000 - $01FFFF ➔ 2^{16} = 64Kbytes RAM

<u>Memory Map</u>

<u>68K address leads</u>

23	22	21	20	19	18	17	16	15	14	13	12	10	09	08	07	06	05	04	03	02	01	00	CHIP
					0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	ROM1
					0	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	ROM2
					0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0	RAM1
					0	0	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	RAM2

UNUSED

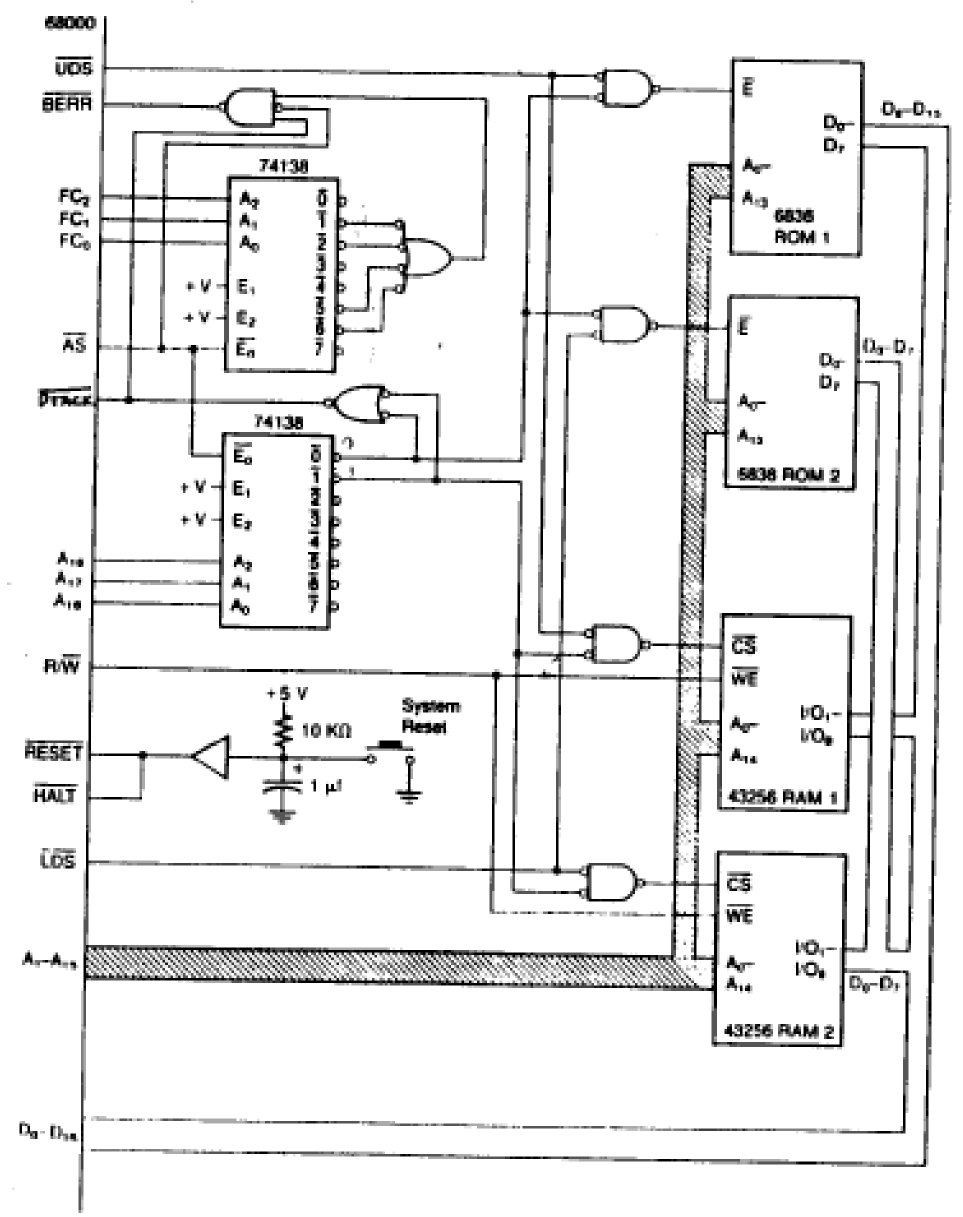
Inputs to LS138 Decoder

Replaced by UDS, LDS

<u>M6836E16 ROM</u>
A0 – A13 addr. Lines
DQ0 – DQ7 data lines
E! = Chip enable
G! = output enable

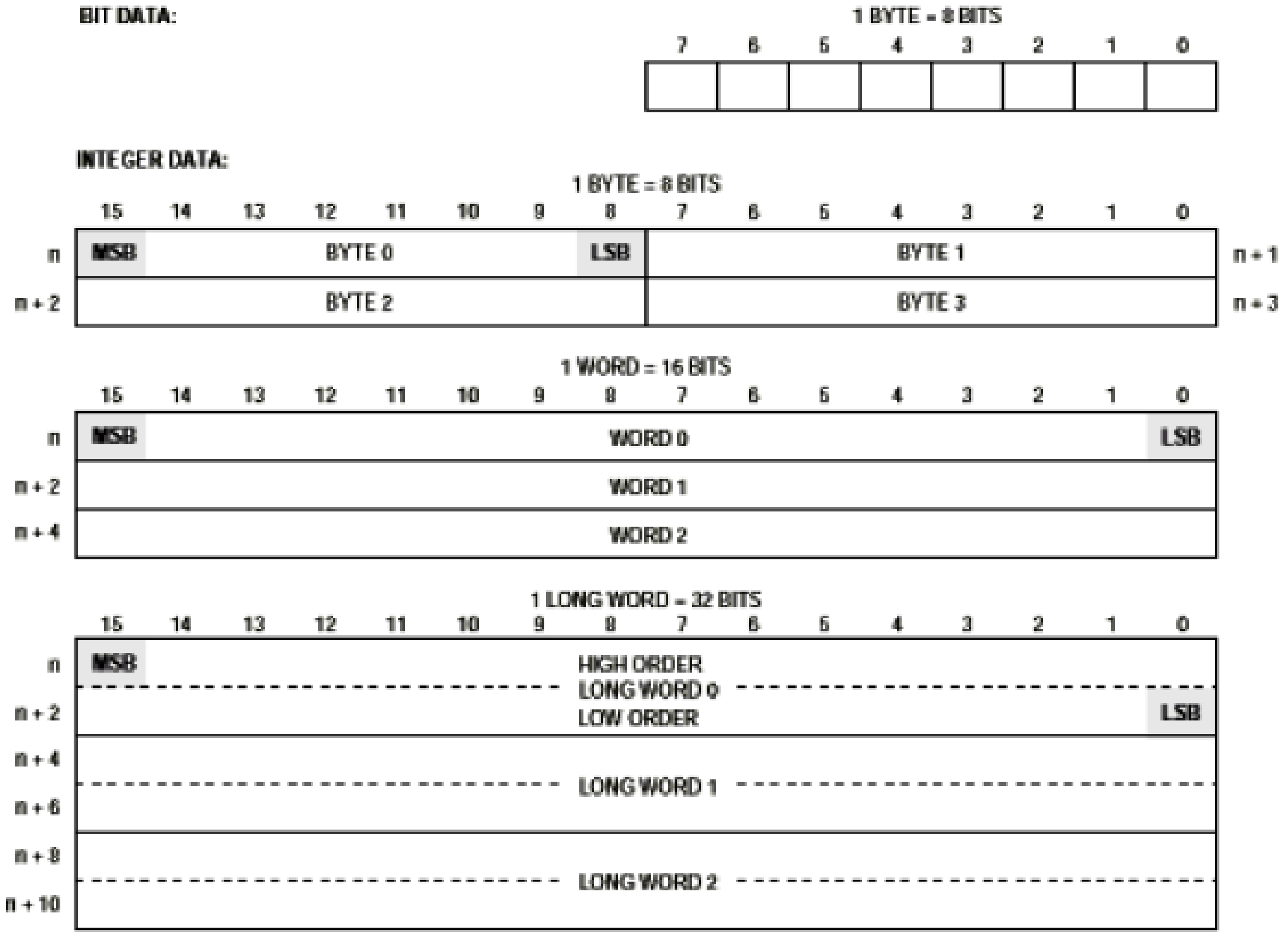
<u>NEC43256 RAM</u>
A0 – A14 addr. Lines
I/O1 – I/O8 data lines
We! = write enable
CS! = chip select

Decoded RAM/ROM interface

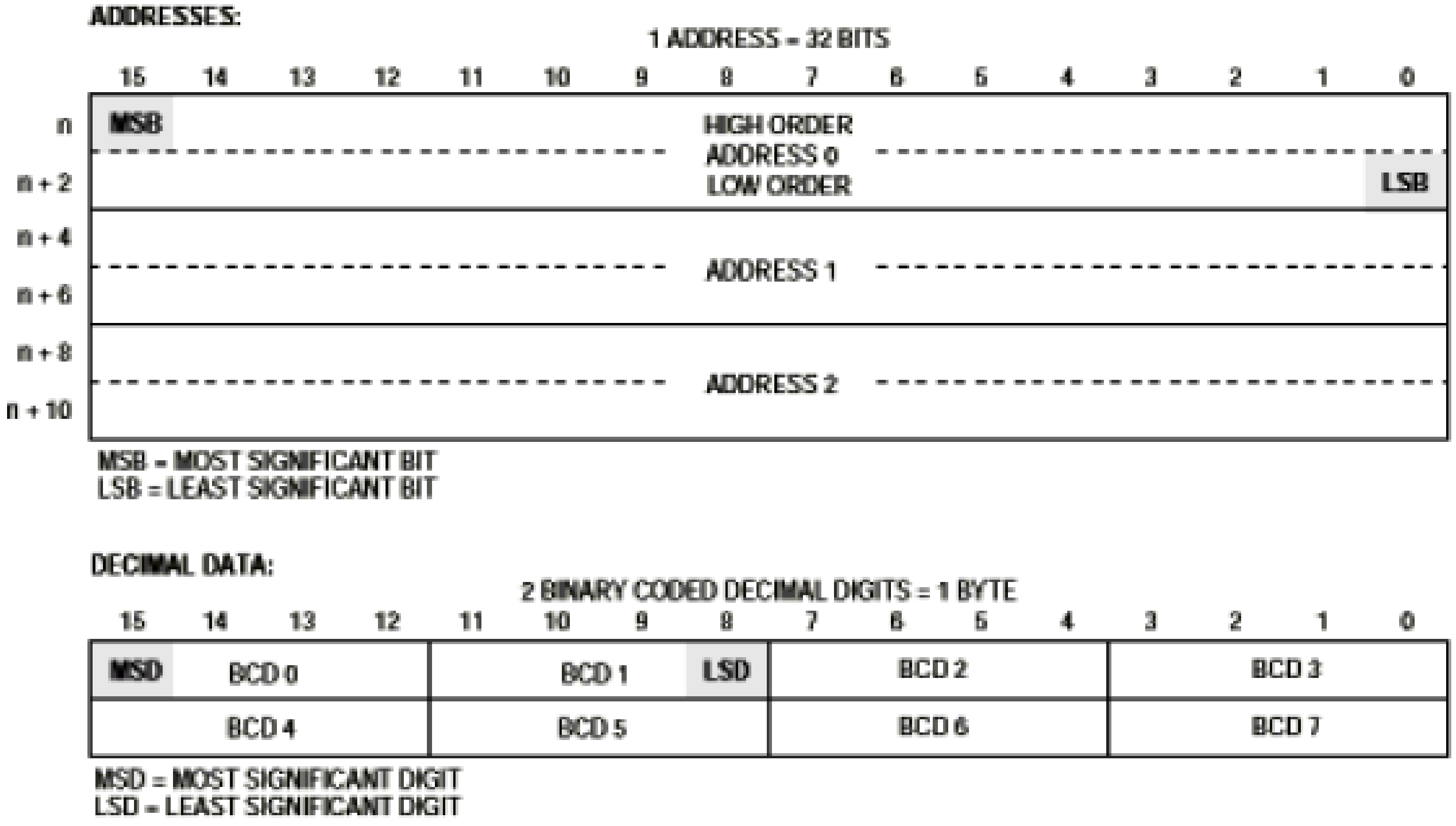


MC680000 Microprocessor Software

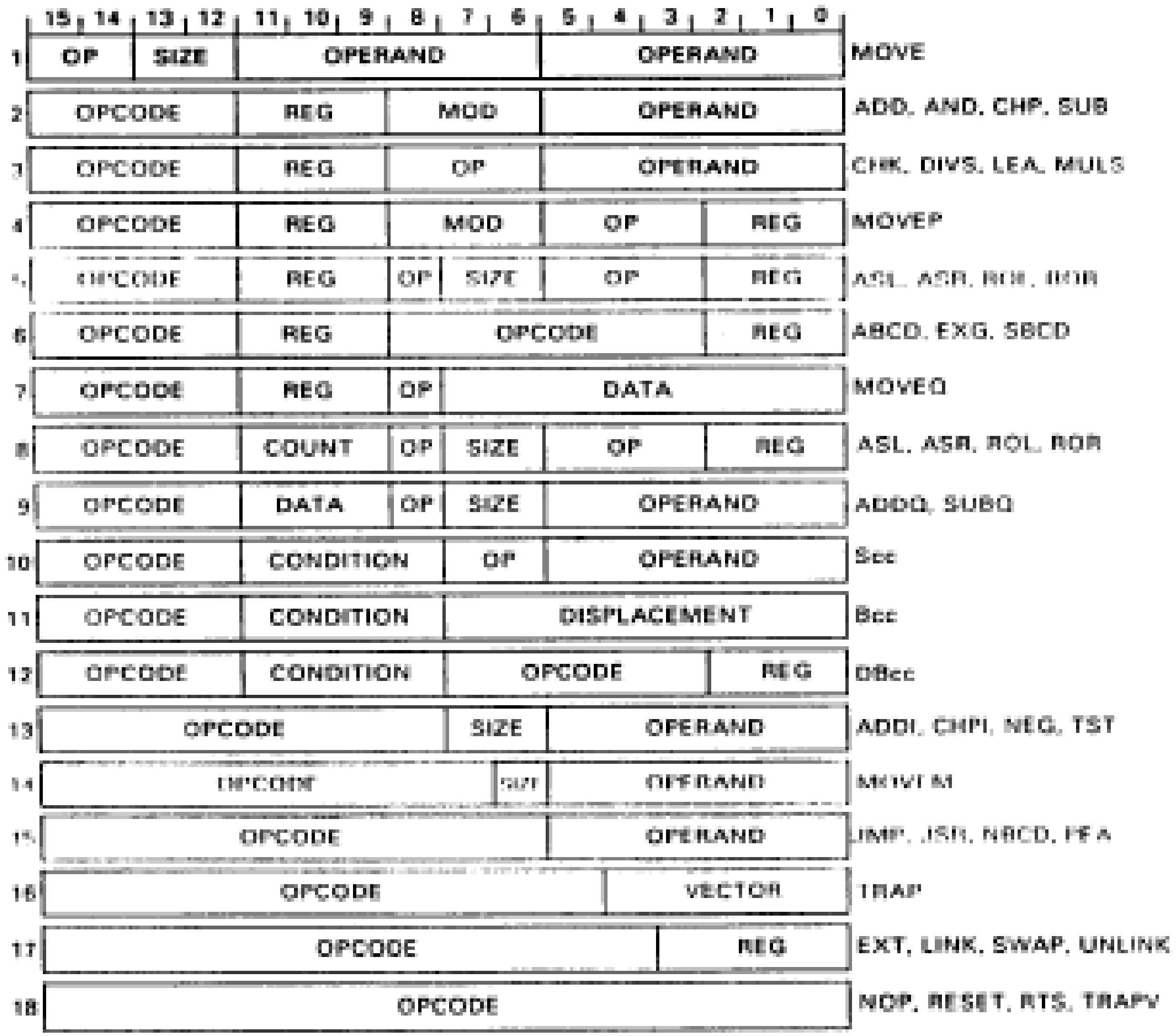
Register notation in memory



Register notation in memory



General instruction formats



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	OP		SIZE		OPERAND						OPERAND						MOVE
2	OPCODE				REG			MOD			OPERAND						ADD, AND, CMP, SUB
3	OPCODE				REG			OP			OPERAND						CHK, DIVS, LEA, MULS
4	OPCODE				REG			MOD			OP			REG			MOVEP
5	OPCODE				REG			OP	SIZE		OP			REG			ASL, ASR, ROL, ROR
6	OPCODE				REG			OPCODE						REG			ABCD, EXG, SBCD
7	OPCODE				REG			OP	DATA								MOVEQ
8	OPCODE				COUNT			OP	SIZE		OP			REG			ASL, ASR, ROL, ROR
9	OPCODE				DATA			OP	SIZE		OPERAND						ADDQ, SUBQ
10	OPCODE				CONDITION				OP		OPERAND						Scc
11	OPCODE				CONDITION				DISPLACEMENT								Bcc
12	OPCODE				CONDITION				OPCODE					REG			DBcc
13	OPCODE								SIZE		OPERAND						ADDI, CMPI, NEG, TST
14	OPCODE									SIZ	OPERAND						MOVEM
15	OPCODE										OPERAND						JMP, JSR, NBCD, PEA
16	OPCODE												VECTOR				TRAP
17	OPCODE													REG			EXT, LINK, SWAP, UNLINK
18	OPCODE																NOP, RESET, RTS, TRAPV

Addressing modes

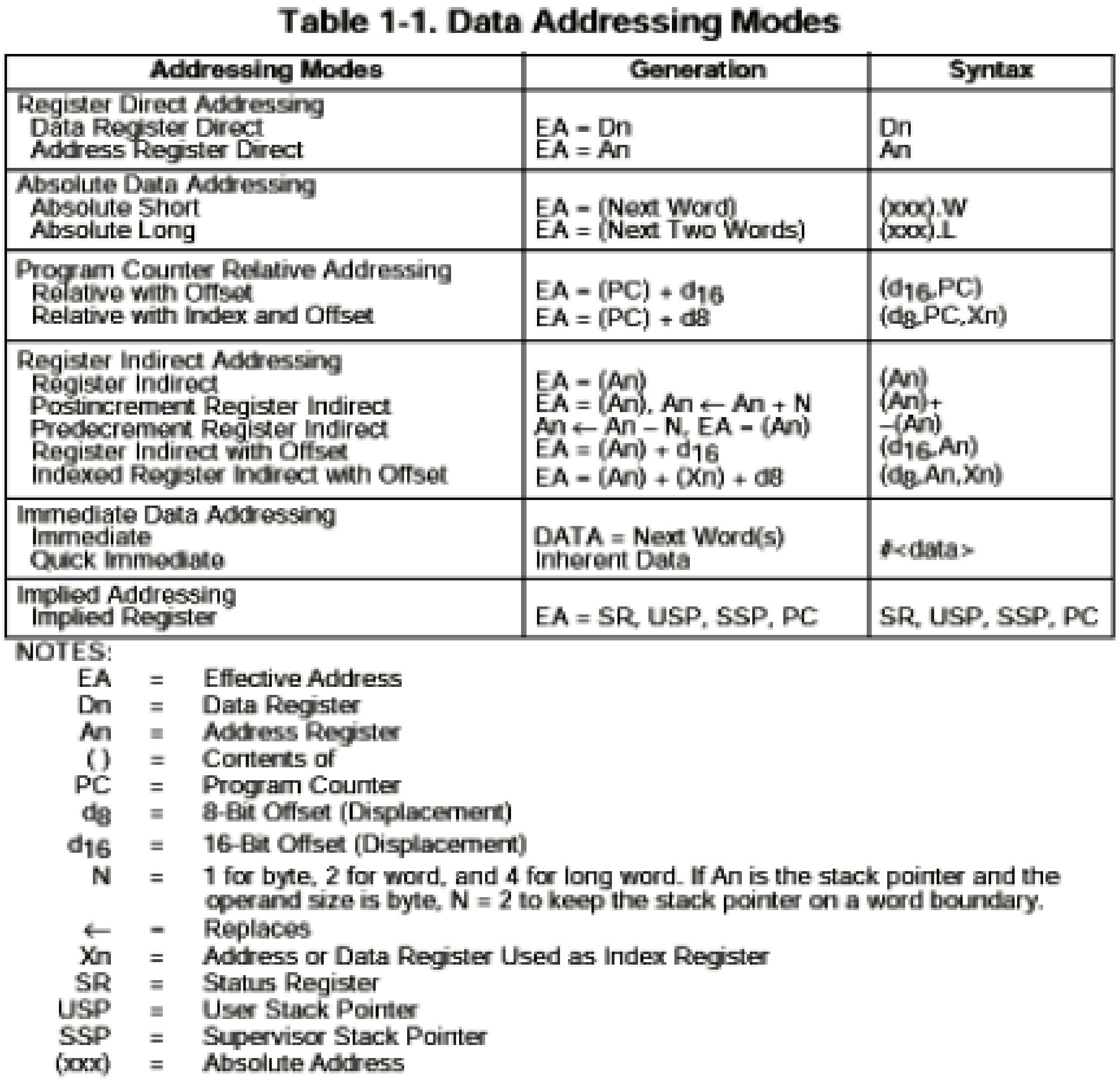


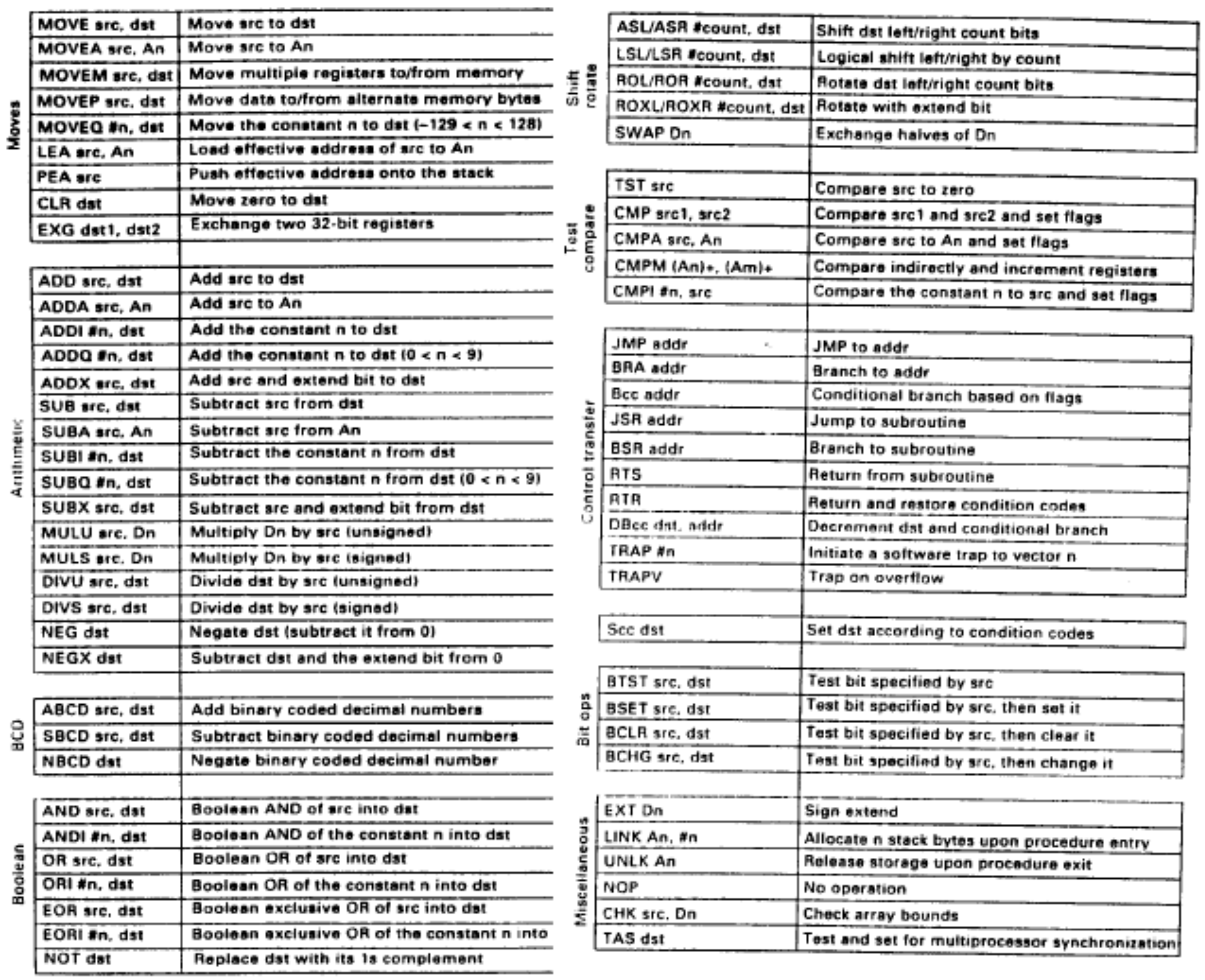
Table 1-1. Data Addressing Modes

Addressing Modes	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) + d8	(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) + d8	(d_8,An,Xn)
Immediate Data Addressing		
Immediate	DATA = Next Word(s)	#<data>
Quick Immediate	Inherent Data	
Implied Addressing		
Implied Register	EA = SR, USP, SSP, PC	SR, USP, SSP, PC

NOTES:

EA	=	Effective Address
Dn	=	Data Register
An	=	Address Register
( )	=	Contents of
PC	=	Program Counter
d_8	=	8-Bit Offset (Displacement)
d_{16}	=	16-Bit Offset (Displacement)
N	=	1 for byte, 2 for word, and 4 for long word. If An is the stack pointer and the operand size is byte, N = 2 to keep the stack pointer on a word boundary.
←	=	Replaces
Xn	=	Address or Data Register Used as Index Register
SR	=	Status Register
USP	=	User Stack Pointer
SSP	=	Supervisor Stack Pointer
(xxx)	=	Absolute Address

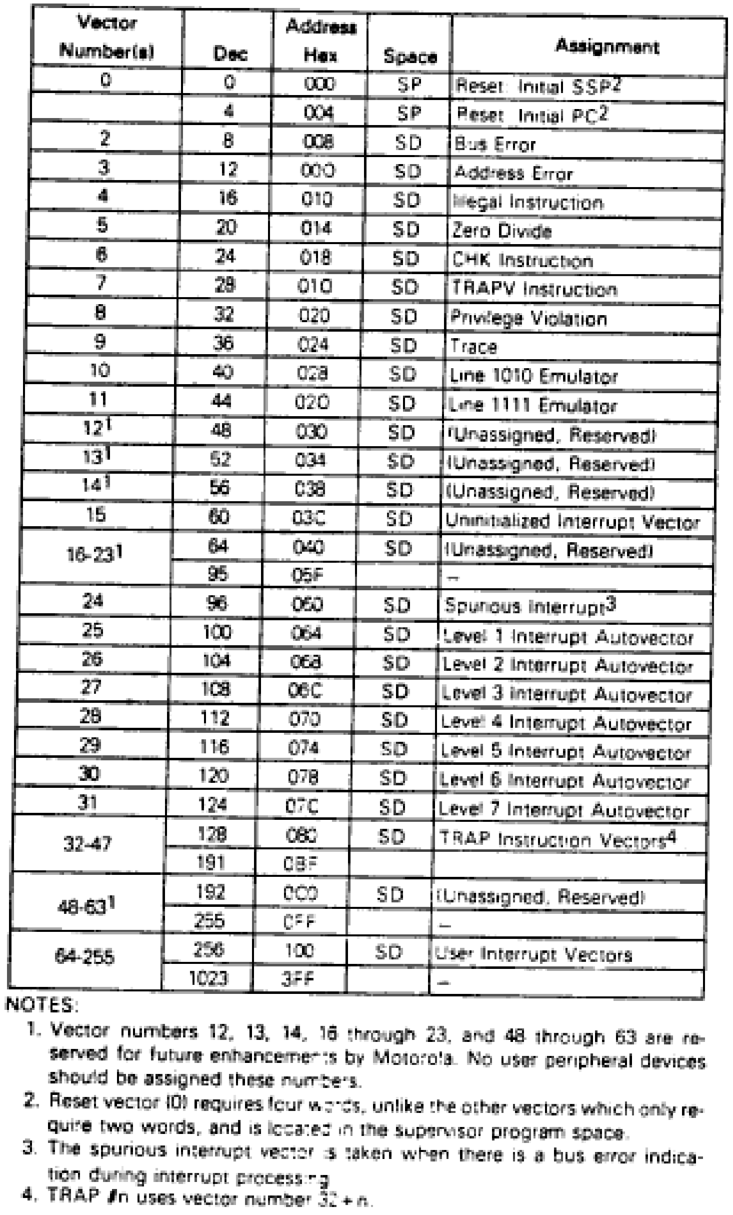
Instruction set functional groups



Moves	MOVE src, dst	Move src to dst
	MOVEA src, An	Move src to An
	MOVEM src, dst	Move multiple registers to/from memory
	MOVEP src, dst	Move data to/from alternate memory bytes
	MOVEQ #n, dst	Move the constant n to dst (–129 < n < 128)
	LEA src, An	Load effective address of src to An
	PEA src	Push effective address onto the stack
	CLR dst	Move zero to dst
	EXG dst1, dst2	Exchange two 32-bit registers
Arithmetic	ADD src, dst	Add src to dst
	ADDA src, An	Add src to An
	ADDI #n, dst	Add the constant n to dst
	ADDQ #n, dst	Add the constant n to dst (0 < n < 9)
	ADDX src, dst	Add src and extend bit to dst
	SUB src, dst	Subtract src from dst
	SUBA src, An	Subtract src from An
	SUBI #n, dst	Subtract the constant n from dst
	SUBQ #n, dst	Subtract the constant n from dst (0 < n < 9)
	SUBX src, dst	Subtract src and extend bit from dst
	MULU src, Dn	Multiply Dn by src (unsigned)
	MULS src, Dn	Multiply Dn by src (signed)
	DIVU src, dst	Divide dst by src (unsigned)
	DIVS src, dst	Divide dst by src (signed)
	NEG dst	Negate dst (subtract it from 0)
	NEGX dst	Subtract dst and the extend bit from 0
BCD	ABCD src, dst	Add binary coded decimal numbers
	SBCD src, dst	Subtract binary coded decimal numbers
	NBCD dst	Negate binary coded decimal number
Boolean	AND src, dst	Boolean AND of src into dst
	ANDI #n, dst	Boolean AND of the constant n into dst
	OR src, dst	Boolean OR of src into dst
	ORI #n, dst	Boolean OR of the constant n into dst
	EOR src, dst	Boolean exclusive OR of src into dst
	EORI #n, dst	Boolean exclusive OR of the constant n into
	NOT dst	Replace dst with its 1s complement

Shift rotate	ASL/ASR #count, dst	Shift dst left/right count bits
	LSL/LSR #count, dst	Logical shift left/right by count
	ROL/ROR #count, dst	Rotate dst left/right count bits
	ROXL/ROXR #count, dst	Rotate with extend bit
	SWAP Dn	Exchange halves of Dn
Test compare	TST src	Compare src to zero
	CMP src1, src2	Compare src1 and src2 and set flags
	CMPA src, An	Compare src to An and set flags
	CMPM (An)+, (Am)+	Compare indirectly and increment registers
	CMPI #n, src	Compare the constant n to src and set flags
Control transfer	JMP addr	JMP to addr
	BRA addr	Branch to addr
	Bcc addr	Conditional branch based on flags
	JSR addr	Jump to subroutine
	BSR addr	Branch to subroutine
	RTS	Return from subroutine
	RTR	Return and restore condition codes
	DBcc dst, addr	Decrement dst and conditional branch
	TRAP #n	Initiate a software trap to vector n
	TRAPV	Trap on overflow
	Scc dst	Set dst according to condition codes
Bit ops	BTST src, dst	Test bit specified by src
	BSET src, dst	Test bit specified by src, then set it
	BCLR src, dst	Test bit specified by src, then clear it
	BCHG src, dst	Test bit specified by src, then change it
Miscellaneous	EXT Dn	Sign extend
	LINK An, #n	Allocate n stack bytes upon procedure entry
	UNLK An	Release storage upon procedure exit
	NOP	No operation
	CHK src, Dn	Check array bounds
	TAS dst	Test and set for multiprocessor synchronization

Exception vectors



Vector Number(s)	Dec	Address Hex	Space	Assignment
0	0	000	SP	Reset: Initial SSP[2]
	4	004	SP	Reset: Initial PC[2]
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[1]	48	030	SD	(Unassigned, Reserved)
13[1]	52	034	SD	(Unassigned, Reserved)
14[1]	56	038	SD	(Unassigned, Reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23[1]	64	040	SD	(Unassigned, Reserved)
	95	05F		–
24	96	060	SD	Spurious Interrupt[3]
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[4]
	191	0BF		
48-63[1]	192	0C0	SD	(Unassigned, Reserved)
	255	0FF		–
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		–

NOTES:

1. Vector numbers 12, 13, 14, 16 through 23, and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.
2. Reset vector (0) requires four words, unlike the other vectors which only require two words, and is located in the supervisor program space.
3. The spurious interrupt vector is taken when there is a bus error indication during interrupt processing.
4. TRAP #n uses vector number 32 + n.

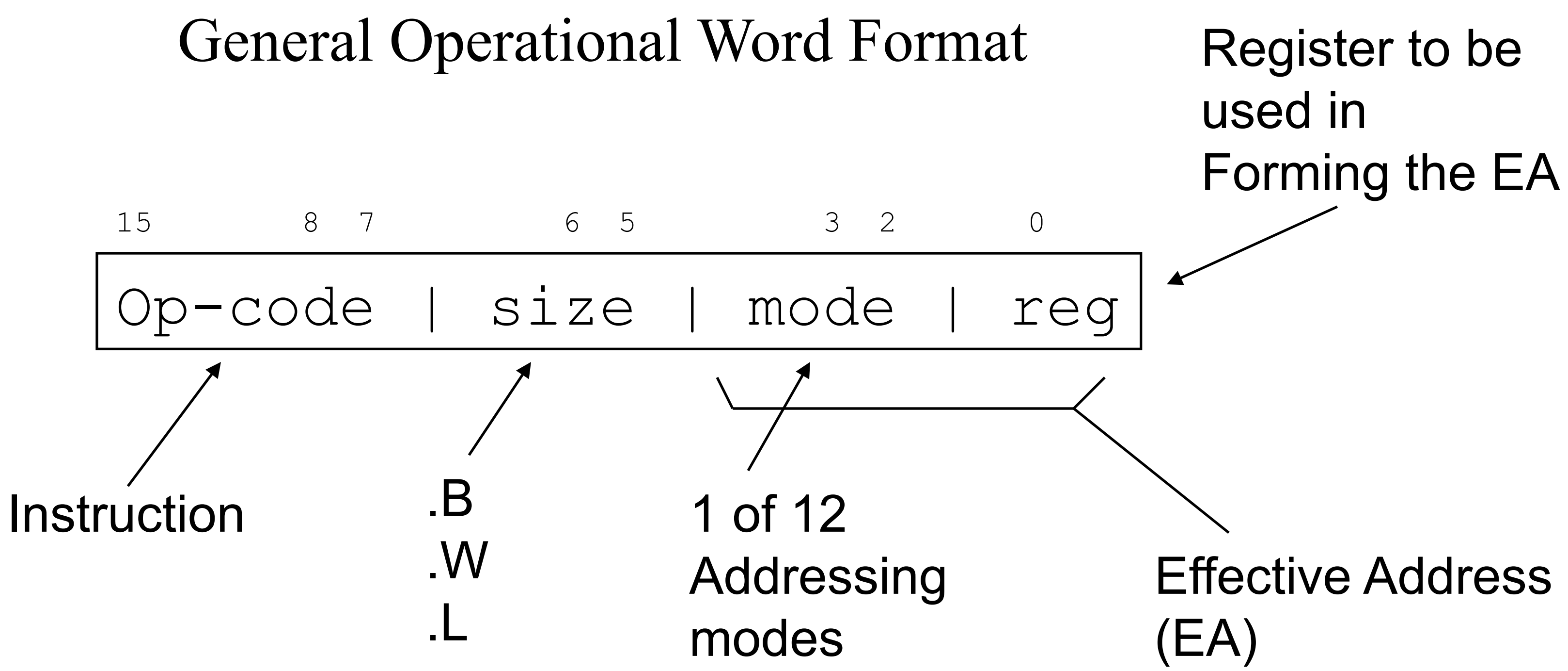
From Wikipedia:

Addressing modes, a concept from computer science, are an aspect of the instruction set architecture in most central processing unit (CPU) designs.

The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction.

An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.

Addressing modes


General Operational Word Format
Register to be used in Forming the EA
15
8
7
6
5
3
2
0
Op-code | size | mode | reg
Instruction
.B
.W
.L
1 of 12 Addressing modes
Effective Address (EA)

Addressing modes

Mode	Register	Addr. Mode	Assembler syntax
000	Data reg. #	Data reg. Direct	Dn
001	Addr reg. #	Addr reg. Direct	An
010	Addr reg. #	Addr reg. Indirect (ARI)	(An)
011	Addr reg. #	ARI w/post increment	(An)+
100	Addr reg. #	ARI w/pre-decrement	-(An)
101	Addr reg. #	ARI w/displacement	d(An)
110	Addr reg. #	ARI w/index	d(An,Ri.x)
111	000	Absolute short	$xxxx
111	001	Absolute long	$xxxx xxxx
111	010	PC w/displacement	d(PC)
111	011	PC w/index	d(PC,Ri.x)
111	100	immediate	#xxxx

Data register direct

Syntax: Dn ← contains the operand

Ex:
Clr.W D1

Before	After
FF FF FF FF	FF FF 00 00

Address register direct

Restrictions: cannot be used with .B operation
cannot be used for dest. Except for special instructions

Syntax: An ← contains the operand

Ex:
Move.W A3, D5

	Before	After
A3	12 34 56 78	12 34 56 78
D5	FF FF FF FF	FF FF 56 78

Immediate

Syntax: #<constant> 🡐 constant = operand

Ex:
Move.W #$123A, D1

	Before	After
D1	FF FF FF FF	FF FF 12 3A

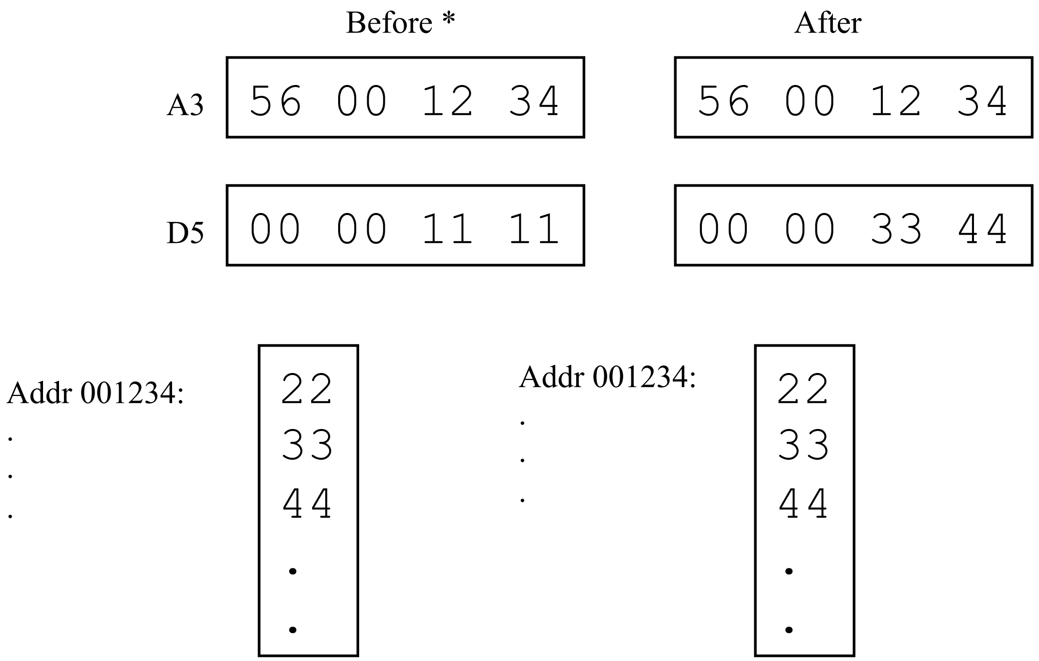
Immediate data can be : decimal, hex, binary, or octal

Address register indirect (ARI)

Syntax: (An)🡐 operand address = contents of An

Ex:
Add.W (A3), D5

D5 = D5 + word at address A3



*NOTE: only the lower 24 bits
Are used for operand address

Address register indirect (ARI)

Ex. 2
Find the sum of 3 consecutive words in memory.
A1 holds the addr. Of the first word.

<u>Assembly code:</u>

```
CLR.L D1      ;SAVE D1 FOR SUM
ADD.W (A1),D1   ;STORE PARTIAL SUM
ADDQ.L #2, A1   ;ADUST
ADD.W (A1),D1   ;PARTIAL SUM
ADD1.L #2, A1   ;ADJUST
ADD.W (A1),D1   ;FINAL SUM
```

```
(A1)      | Word 1 |
(A1) + 2  | Word 2 |
(A1) + 4  | Word 3 |
          |   .    |
          |   .    |
```

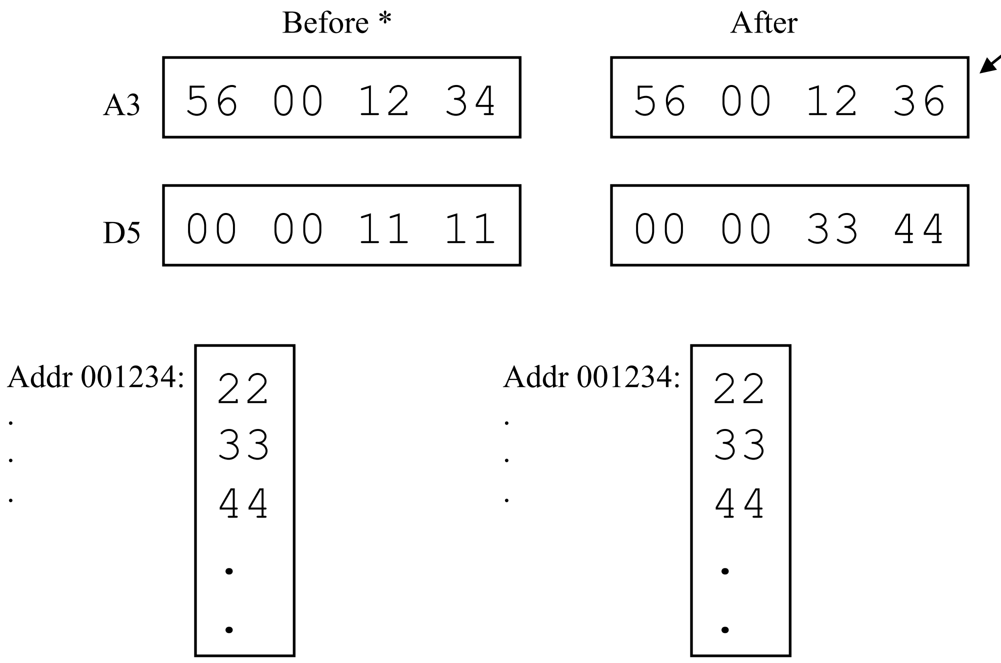
Add quick – short form of add.
#n = 1 to 8 bytes of precision

ARI with post increment

Syntax: (An)+🡐 operand address = contents of An, after accessing,
An is incremented by 1(.B), 2(.W) or 4(.L)

Ex:

Add.W (A3)+, D5

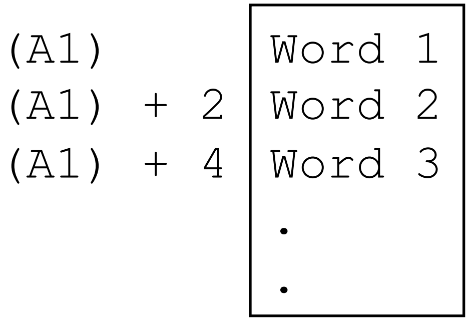


ARI with post increment

Ex. 2
Find the sum of 3 consecutive words in memory.
A1 holds the addr. Of the first word.

<u>Assembly code:</u>

```
CLR.L D1       ;SAVE D1 FOR SUM
ADD.W (A1)+,D1 ;STORE PARTIAL SUM
ADD.W (A1)+,D1 ;PARTIAL SUM
ADD.W (A1)+,D1 ;FINAL SUM
```

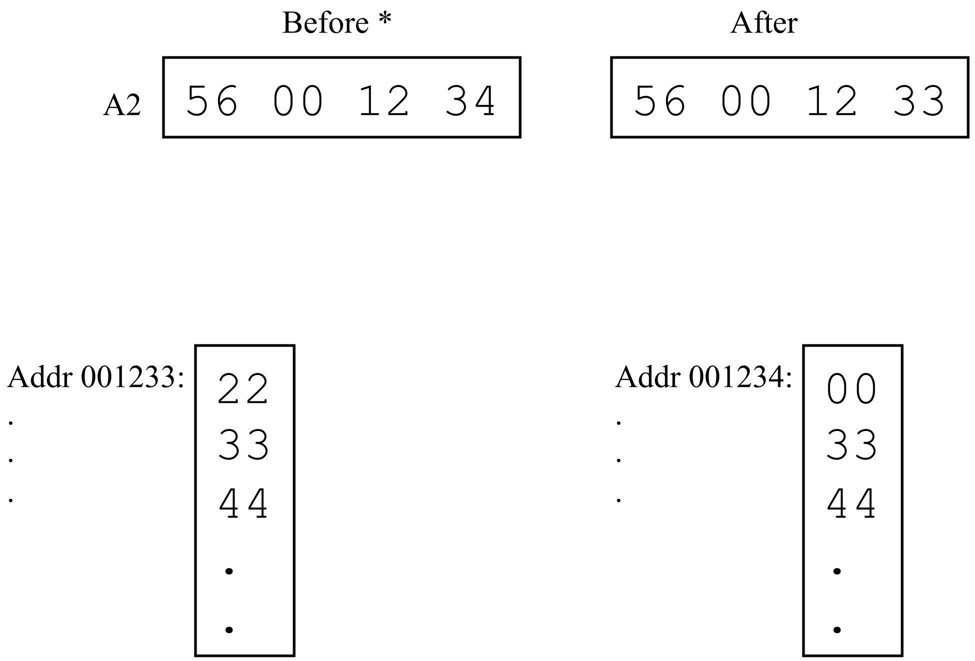


ARI with pre-decrement

Syntax: -(An)🡐 An is decremented 1st by 1, 2, or 4.

Ex:

CLR.B –(A2)



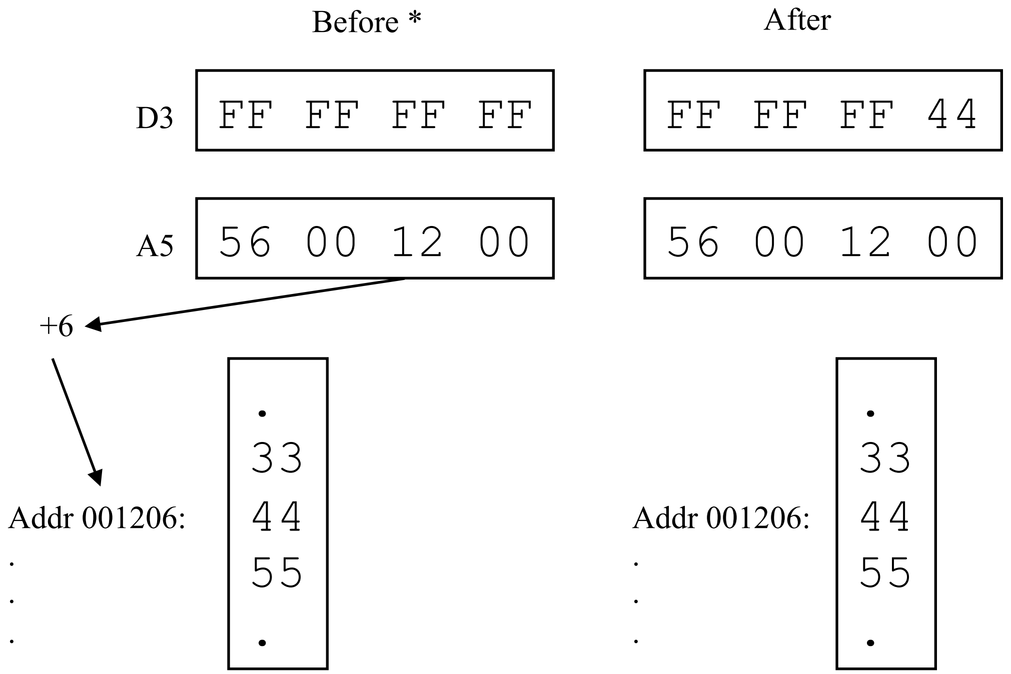
ARI with displacement

Syntax: d(An)🡐 d = constant (16 bits)

Operand address = contents of An + d

Ex:

Move.B 6(A5), D3



ARI with displacement

Useful for accessing records:

```
A1 contains address of beginning of record

Make salary = 50,000 :
    Move.L #50000, 36(A1)

Add 1 to age:
    ADD.W #1, 34(A1)

Add space as start of name:
    Move.B $20, 9(A1)
```

	Record	Size(bytes)	Starting offfset Into memory
(A1)	SS-NO	9	0
	Name	25	9
	AGE	2	34
	Salary	4	36

ARI with Index

Syntax: d(An,Ri.x) 🡐 d = displacement (8-bits); An = Address reg.
Ri = addr or data reg. (index); x = W – 16 bit index; L – 32 bit index

Ex:
Clr.B 2(A3, D5.W)

	Before *	After
A3	56 00 12 34	56 00 12 34
D5	11 11 24 06	11 11 24 06

	Before *	After
	22	22
00363C	33	00
	44	44
	.	.
	.	.

```
Addr = 56 001234
       00 002406
       00 000002
       ---------
       56 00363C
```

Sign extended

Absolute short, long and immediate

Absolute Short Address
The effective address is specified absolutely.
The address can no be larger than 16 bits.

Absolute Long Address
The effective address is specified absolutely.
The address is larger than 16 bits.

Immediate Data
The data is specified absolutely.
The maximum size depends on the opcode.
Hex data can be specified using a ‘$’.

```
MOVE #6,D0
MOVE #-6,D0
MOVE #$6,D0
MOVE #$-6,D0
```

Program counter w/displacement, index

Program Counter Indirect with Displacement
The effective address is the sum of the content of the program counter and the 16 bit two's complement integer.

```
CLR 4(PC)
```

Program Counter Indirect with Displacement and Index
The effective address is the sum of the content of the program counter, the 16 bit two's complement integer, and the index register.
The index register can be a data or an address register and
it can be a word or a long word in size.

```
LEA 4(PC,D1.L),A1
```

Links to Motorola processor data sheets:

MC68000
http://www.freescale.com/files/32bit/doc/ref_manual/MC68000UM.pdf

MC68020
http://cache.freescale.com/files/32bit/doc/ref_manual/MC68020UM.pdf?pspll=1&Parent_nodeId=H966655295119&Parent_pageType=product

MC68030
http://cache.freescale.com/files/32bit/doc/data_sheet/MC68EC030TS.pdf?fasp=1&Parent_nodeId=Y966655342274&Parent_pageType=product

Coldfire
http://www.freescale.com/webapp/sps/site/homepage.jsp?code=PC68KCF