EECE416 :Microcomputer and Microprocessor

source: www.mwftr.com

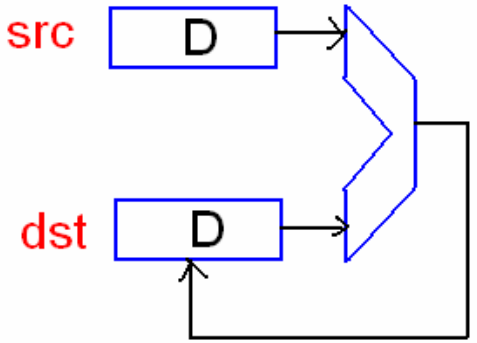
68000 Instruction and Programming Environment

Instruction and Addressing

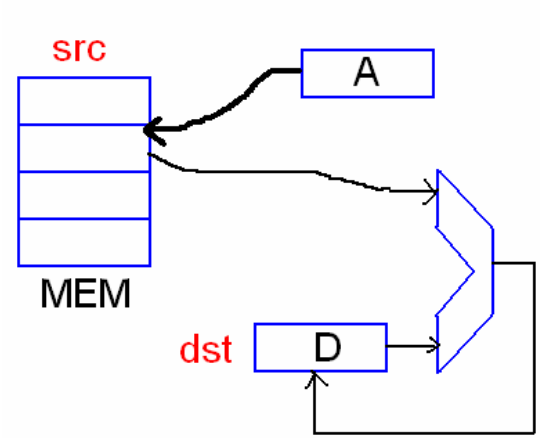
- Instruction
  - Type of function to be performed
  - Location of the operand on which to perform the function
- Addressing
  - Method to locate the operand(s)
  - 3 categories
    - Register Specification: the number of the register
    - Effective Address(EA): several modes available
    - Implicit (*special*) Reference: instruction itself implies the use of specific registers

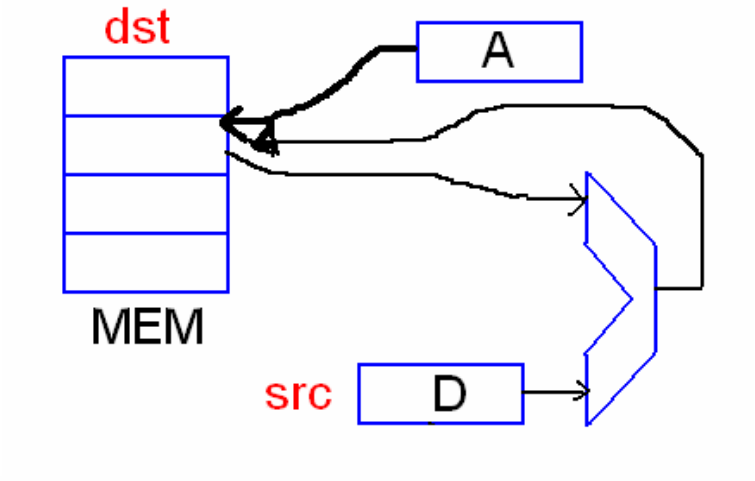
Illustration of 3 categories

- Register Specific Addressing

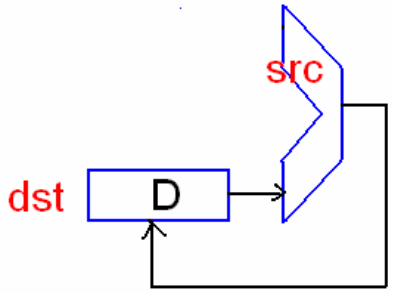


- Memory Addressing





- Special Addressing



Instruction Format & Program EXAMPLE

Instruction format is

```
<label> opcode<.field> <operands> <;comments>
```

- `<label>` pointer to the instruction's memory location
- `opcode` operation code (i.e., MOVE, ADD)
- `<.field>` defines width of operands (B,W,L)
- `<operands>` data used in the operation
- `<;comments>` for program documentation

LABEL	OPCODE	OPERAND(S)	COMMENTS	
	ORG	$1000	;start of PROGRAM area	PROGRAM AREA
	MOVE.L	#$12,d0		
	CLR.L	d1		
	MOVE.B	data,d1		
	ADD.B	d0,d1		
	MOVE.B	d1,result		
	RTS		;return	
	ORG	$2000	;start of DATA area	DATA AREA
data	DC.B	$24		
result	DS.B	1	;reserve a byte for result	
	END	$1000	;end of program and entry point	

RTL (Register Transfer Language)

1. Notation for Operands

PC	Program Counter
SR	Status Register
Source	Source contents
Destination	Destination Contents
< >	Operand data format: B, W, L
Dn	Data Register n
An	Address Register n
Rn	Any Data or Address Register
CCR	Condition Code Register (Lower Byte of SR)
SP	Stack Pointer (=A7)
d	displacement (or "offset"): d8- eight-bit offset, d16-16-bit offset

2. Notation for sub-field and qualifier

<ea>	Effective address
(<operand>)	Contents of the referenced location
#xxx	Immediate Data

3. Notation for operations

-->	Source operand is moved to the destination operand
<-->	Two operands are exchanged
^	Logical AND
v	Logical OR
⊕	Logical Exclusive OR
~	Operand is logically complemented
<>sign-ext	Operand is sign-extended (i.e., all bits of the upper portion [Upper Byte] are made equal to the sign-bit [msb] of the lower portion [Lower Byte]

4. Examples

Opcode	Operation	Syntax
ADD	Source + destination -->destination	ADD <ea>, Dn
ADDI	Immediate Data + Destination -->Destination	ADDI #<data>, <ea>
MOVE	Source --> Destination	MOVE <ea>, <ea>
NOT	~Destination --> Destination	NOT <ea>
SUB	Destination – Source --> Destination	SUB <ea>, Dn

RTL example

Examples

	Instruction		RTL
	MOVE.W	#100,D0	[D0]←100
	MOVE.W	100,D0	[D0]←[M(100)]
	ADD.W	D0,D1	[D1]←[D1]+[D0]
	MOVE.W	D1,100	[M(100)]←[D1]
data	DC.B	20	[data] ←20
	BRA	label	[PC] ←label

2. Notation for sub-field and qualifier

<ea>	Effective address
(<operand>)	Contents of the referenced location
#xxx	Immediate Data

3. Notation for operations

-->	Source operand is moved to the destination operand
<-->	Two operands are exchanged
^	Logical AND
v	Logical OR
⊕	Logical Exclusive OR
~	Operand is logically complemented
<>sign-ext	Operand is sign-extended (i.e., all bits of the upper portion [Upper Byte] are made equal to the sign-bit [msb] of the lower portion [Lower Byte]

ADDRESSING MODES

- addressing mode specifies the value of an operand, a register that contains the operand, or how to derive the effective address of an operand in memory.
  - **Data Reg. Direct Mode**
  - **Address Reg. Direct Mode**
  - **Address Reg. Indirect Mode**
  - **Address Reg. Indirect with Post-increment Mode**
  - **Address Reg. Indirect with Pre-decrement Mode**
  - **Address Reg. Indirect with Displacement Mode**
  - **Address Reg. Indirect with Index Mode**
  - **PC Indirect with Displacement Mode**
  - **PC Indirect with Index Mode**
  - **Absolute Short Addressing Mode**
  - **Absolute Long Addressing Mode**
  - **Immediate Data Mode**

Addressing Mode Summary

Addressing Modes	Syntax	Mode Field	Reg. Field	Data	Memory	Control	Alterable
Register Direct							
Data	Dn	000	reg. no.	X	—	—	X
Address	An	001	reg. no.	—	—	—	X
Register Indirect							
Address	(An)	010	reg. no.	X	X	X	X
Address with Postincrement	(An)+	011	reg. no.	X	X	—	X
Address with Predecrement	–(An)	100	reg. no.	X	X	—	X
Address with Displacement	(d_{16},An)	101	reg. no.	X	X	X	X
Address Register Indirect with Index							
8-Bit Displacement	(d_8,An,Xn)	110	reg. no.	X	X	X	X
Base Displacement	(bd,An,Xn)	110	reg. no.	X	X	X	X
Program Counter Indirect							
with Displacement	(d_{16},PC)	111	010	X	X	X	—
Program Counter Indirect with Index							
8-Bit Displacement	(d_8,PC,Xn)	111	011	X	X	X	—
Base Displacement	(bd,PC,Xn)	111	011	X	X	X	—
Program Counter Memory Indirect							
Postindexed	([bd,PC],Xn,od)	111	011	X	X	X	X
Preindexed	([bd,PC,Xn],od)	111	011	X	X	X	X
Absolute Data Addressing							
Short	(xxx).W	111	000	X	X	X	—
Long	(xxx).L	111	000	X	X	X	—
Immediate	#<xxx>	111	100	X	X	—	—

Instruction Word (machine Code) Format

1 Word Length: Simple Instruction

4 Word Length: Complex Instruction with EA extension

SINGLE EFFECTIVE ADDRESS OPERATION WORD FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	EFFECTIVE ADDRESS MODE			REGISTER		

BRIEF EXTENSION WORD FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		0	DISPLACEMENT							

FULL EXTENSION WORD FORMAT

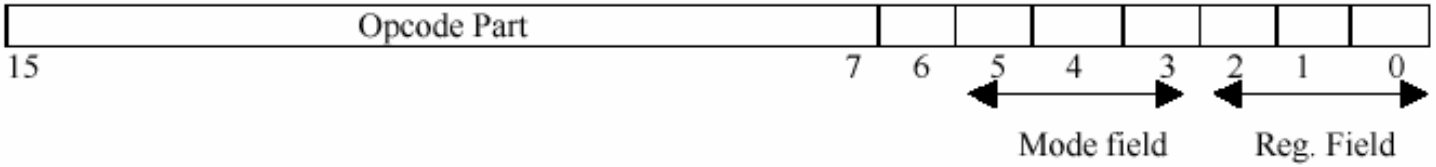
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		1	BS	IS	BD SIZE		0	I/IS		
BASE DISPLACEMENT (0, 1, OR 2 WORDS)															
OUTER DISPLACEMENT (0, 1, OR 2 WORDS)															

15 0

SINGLE EFFECTIVE ADDRESS OPERATION WORD (ONE WORD, SPECIFIES OPERATION AND MODES)
SPECIAL OPERAND SPECIFIERS (IF ANY, ONE OR TWO WORDS)
IMMEDIATE OPERAND OR SOURCE EFFECTIVE ADDRESS EXTENSION (IF ANY, ONE TO SIX WORDS)
DESTINATION EFFECTIVE ADDRESS EXTENSION (IF ANY, ONE TO SIX WORDS)

Instruction Word Field (Opcode Bit Pattern): single word

1. Opcode Bit Pattern



2. Addressing Modes

Selected Mnemonics

Mnemonic	Size	Address Mode	Opcode Bit Pattern 1111 11 5432 1098 7654 3210	Boolean
ADD	B/W L	s=Dn d=Dn S=Dn d=Dn	1101 DDD1 SSEE EEEE 1101 DDD0 SSee eeee 1101 DDD1 10EE EEEE 1101 DDD0 10ee eeee	d+Dn→d Dn+s→Dn d+Dn→d Dn+s→Dn
MOVE	B/W L		00XX RRRM MMee eeee 0010 RRRM MMee eeee	s→d s→d

Opcode Bit Pattern Codes:

Selected items

Code	Description		Code	Description
A	Address Register Number		**s**	Source
D	Data Register Number		**d**	Destination
E	Destination Effective Address		**R**	Destination Register
e	Source Effective Address		**r**	Source Register
M	Destination EA Mode		**P**	Displacement
S	Size: 00 byte, 01 Word, 10 Long		**XX**	Move Size 01 byte 11 Word

Machine Code Example

Mnemonic	Size	Address Mode	Opcode Bit Pattern 1111 11 5432 1098 7654 3210	Boolean
ADD	B/W L	s=Dn d=Dn S=Dn d=Dn	1101 DDD1 SSEE EEEE 1101 DDD0 SSee eeee 1101 DDD1 10EE EEEE 1101 DDD0 10ee eeee	d+Dn→d Dn+s→Dn d+Dn→d Dn+s→Dn
MOVE	B/W L		00XX RRRM MMee eeee 0010 RRRM MMee eeee	s→d s→d

Table given for 68000 processor by Motorola

3. Mode categories

Type	Mode	Register	Generation	Assembler Syntax
Data Register Direct	000	Reg. No.	EA=Dn	Dn
Address Register Direct	001	Reg. No.	EA=An	An
Register Indirect	010	Reg. No.	EA=(An)	(An)
Post-increment Reg. Ind.	011	Reg. No.	EA=(An), An ←An+N	(An)+
Pre-decrement Reg. Ind.	100	Reg. No.	An ←An-N, EA=(An)	-(An)
Reg. Indirect with Disp.	101	Reg. No.	EA=(An)+d_{16}	d_{16}(An)
Indexed Reg. Ind. w/ Disp	110	Reg. No.	EA=(An)+(Xn)+d_8	d_8 (An, Xn)
Absolute Short	111	000	EA=(Next Word)	XXX
Absolute Long	111	001	EA=(Next Two Words)	XXXXXX
PC relative with Disp.	111	010	EA=(PC)+d_{16}	d_{16}(PC)
PC rel. w/ Ind. and Disp.	111	011	EA=(PC)+(Xn)+d_8	d_8(PC+Xn)
Immediate	111	100	Data=Next Word(s)	#XXX

Code	Description		Code	Description
A	Address Register Number		s	Source
D	Data Register Number		d	Destination
E	Destination Effective Address		R	Destination Register
e	Source Effective Address		r	Source Register
M	Destination EA Mode		P	Displacement
S	Size: 00 byte, 01 Word, 10 Long		XX	Move Size 01 byte 11 Word

ADD.B D2, D3 **1101 DDD0 SS eeeeee**

D2 is source, D3 is destination
Therefore DDD =011 (register number 3)
SS=00, byte size
eee=000 source register mode
eee=010 source register number 2
Finally, the code is: **1101 0110 0000 0010 --->D602**

How about **ADD.W D0, D1**? ---->DDD=001, SS=01, ee=000, eee=000 ----->**D240**

Opcode Patterns of Selected Instructions

Mnemonic	Size	Mode	Opcode Bit Pattern[1] 15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Boolean	Condition Code[2] X	N	Z	V	C
ADD	B/W	s=Dn	1	1	0	1	D	D	D	1	S	S	E	E	E	E	E	E	d +Dn --> d	*	*	*	*	*
		d=Dn	1	1	0	0	D	D	D	0	S	S	e	e	e	e	e	e	Dn + s -->Dn					
	L	s=Dn	1	1	0	1	D	D	D	1	1	0	E	E	E	E	E	E	d + Dn -->d					
		d=Dn	1	1	0	0	D	D	D	0	1	0	e	e	e	e	e	e	Dn + s --> Dn					
ADDA	W	d=An	1	1	0	1	A	A	A	0	1	1	e	e	e	e	e	e	An + s --> An	-	-	-	-	-
	L	d=An	1	1	0	1	A	A	A	1	1	1	e	e	e	e	e	e						
ADDI	B/W	s=Imm	0	0	0	0	0	1	1	0	S	S	E	E	E	E	E	E	d + # --> d	*	*	*	*	*
	L	s=imm																						
AND	B/W	s=Dn	1	1	0	0	D	D	D	1	S	S	E	E	E	E	E	E	d<and>DN -->d	-	*	*	0	0
		d=Dn	1	1	0	0	D	D	D	0	S	S	e	e	e	e	e	e	Dn<and>s -->Dn					
	L	s=Dn	1	1	0	0	D	D	D	1	1	0	E	E	E	E	E	E	d(and>Dn -->d					
		d=Dn	1	1	0	0	D	D	D	0	1	0	e	e	e	e	e	e	Dn<and>s -->Dn					
Bcc[3]	B		0	1	1	0	C	C	C	C	P	P	P	P	P	P	P	P	If CC true, then	-	-	-	-	-
	W																		PC+disp --> PC					
BRA	B		0	1	1	0	0	0	0	0	P	P	P	P	P	P	P	P	PC+disp --> PC	-	-	-	-	-
	W																							
BSR	B		0	1	1	0	0	0	0	1	P	P	P	P	P	P	P	P	PC -->-(SP),	-	-	-	-	-
	W																		PC+disp -->PC					
CLR	B/W		0	1	0	0	0	0	1	0	S	S	E	E	E	E	E	E	0 --> d	-	0	1	0	0
	L																							
CMP	B/W	d=Dn	1	0	1	1	D	D	D	0	D	D	e	e	e	e	e	e	Dn - s	-	*	*	*	*
	L	d=Dn																						
CMPA	B/W	d=An	1	0	1	1	A	A	A	0	1	1	e	e	e	e	e	e	An - s	-	*	*	*	*
	L	d=An	1	0	1	1	A	A	A	1	1	1	e	e	e	e	e	e						

[1] **Opcode Bit Pattern Codes:**
A: Address Register Number C: Test Condition D: Data Register Number E: Destination Effective Address e: Source Effective Address
M: Destination EA Mode P: Displacement Q: Quick Immediate Data R: Destination Register r: Source Register
S: Size (00: B, 01: W, 10: L) XX: Move size (01:B, 11:W)

[2] **Condition Code Notation**
*: Set according to result of operation - : Not affected by operation 0: Cleared 1: Set U: Undefined

[3] See Page 3, "Condition Tests" table

																			d --> PC					
MOVE	B/W		0	0	X	X	R	R	R	M	M	M	e	e	e	e	e	e	s --> d	-	*	*	0	0
	L		0	0	1	0	R	R	R	M	M	M	e	e	e	e	e	e	s --> d					

Opcode Patterns of Selected Instructions

DIVS	W	d=Dn	1	0	0	0	D	D	D	1	1	1	e	e	e	e	e	e	Dn32/s16 -->Dn(r:q)	-	*	*	*	0
DIVU	W	d=Dn	1	0	0	0	D	D	D	0	1	1	e	e	e	e	e	e	DN32/s16 -->DN (r:q)	-	*	*	*	0
EOR	B/W L	s=Dn s=Dn	1	0	1	1	r	r	r	1	S	S	E	E	E	E	E	E	d⊕Dn --> d	-	*	*	0	0
JMP			0	1	0	0	1	1	1	0	1	1	E	E	E	E	E	E	d --> PC	-	-	-	-	-
JSR			0	1	0	0	1	1	1	0	1	0	E	E	E	E	E	E	PC --> -(SP), d --> PC	-	-	-	-	-
MOVE	B/W L		0 0	0 0	X 1	X 0	R R	R R	R R	M M	M M	M M	e e	e e	e e	e e	e e	e e	s --> d s --> d	-	*	*	0	0
MOVEA	W L		0 0	0 0	1 1	1 0	A A	A A	A A	0 0	0 0	1 1	e e	e e	e e	e e	e e	e e	s --> An	-	-	-	-	-
MULS	W	d=Dn	1	1	0	0	D	D	D	1	1	1	e	e	e	e	e	e	Dn x s --> Dn	-	*	*	0	0
MULU	W	d=Dn	1	1	0	0	D	D	D	0	1	1	e	e	e	e	e	e	Dn x s --> Dn	-	*	*	0	0
NEG	B/W L		0	1	0	0	0	1	0	0	S	S	E	E	E	E	E	E	0 – d --> d	*	*	*	*	*
NOT	B/W L		0	1	0	0	0	1	1	0	S	S	E	E	E	E	E	E	~d --> d	-	*	*	0	0
OR	B/W L	s=Dn d=Dn s=Dn d=Dn	1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	D D D D	D D D D	D D D D	1 0 1 0	S S 1 1	S S 0 0	E e E e	E e E e	E e E e	E e E e	E e E e	E e E e	d<or>Dn --> d Dn <or> s --> Dn d <or> Dn --> d DN <or> s --> Dn	-	*	*	0	0
RTS			0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1	(SP)+ --> PC	-	-	-	-	-
SUB	B/W L	s=Dn d=Dn s=Dn d=Dn	1 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1	D D D D	D D D D	D D D D	1 0 1 0	S S 1 1	S S 0 0	E e E e	E e E e	E e E e	E e E e	E e E e	E e E e	d – Dn -->d Dn – s --> Dn d – Dn -->< d Dn – s --> Dn	*	*	*	*	*
SUBA	W L	d=An d=An	1 1	0 0	0 0	1 1	A A	A A	A A	0 1	1 1	1 1	e e	e e	e e	e e	e e	e e	An – s --> An	-	-	-	-	-
SWAP	W		0	1	0	0	1	0	0	0	0	1	0	0	0	D	D	D	Dn(31:16) <--->DN(15:0)	-	*	*	0	0
TRAP			0	1	0	0	1	1	1	0	0	1	1	1	0	1	1	1	PC --> -(SSP), SR --> -(SSP), (trap vector) --> PC	-	-	-	-	-
TST	B/W L		0	1	0	0	1	0	1	0	S	S	E	E	E	E	E	E	test d --> cc	-	*	*	0	0

Machine Code Example 2

MOVE	B/W L		0 0	0 0	X 1	X 0	R R	R R	R R	M M	M M	M M	e e	e e	e e	e e	e e	e e	s --> d s --> d

[1] **Opcode Bit Pattern Codes:**

A: Address Register Number C: Test Condition D: Data Register Number E: Destination Effective Address e: Source Effective Address
M: Destination EA Mode P: Displacement Q: Quick Immediate Data R: Destination Register r: Source Register
S: Size (00: B, 01: W, 10: L) XX: Move size (01:B, 11:W)

3. Mode categories

Type	Mode	Register	Generation	Assembler Syntax
Data Register Direct	000	Reg. No.	EA=Dn	Dn
Address Register Direct	001	Reg. No.	EA=An	An
Register Indirect	010	Reg. No.	EA=(An)	(An)
Post-increment Reg. Ind.	011	Reg. No.	EA=(An), An ←An+N	(An)+
Pre-decrement Reg. Ind.	100	Reg. No.	An ←An-N, EA=(An)	-(An)
Reg. Indirect with Disp.	101	Reg. No.	EA=(An)+d_{16}	d_{16}(An)
Indexed Reg. Ind. w/ Disp	110	Reg. No.	EA=(An)+(Xn)+d_8	d_8 (An, Xn)
Absolute Short	111	000	EA=(Next Word)	XXX
Absolute Long	111	001	EA=(Next Two Words)	XXXXXX
PC relative with Disp.	111	010	EA=(PC)+d_{16}	d_{16}(PC)
PC rel. w/ Ind. and Disp.	111	011	EA=(PC)+(Xn)+d_8	d_8(PC+Xn)
Immediate	111	100	Data=Next Word(s)	#XXX

- **Move.w (A0), D0 →00xx RRRMMM eeeeee**
  →0011 000000 010000→3010
- **Move.B (A0), D0 →0001 000000 010000→1010**
- **Move.W (A1), D1 →0011 001000 010001→3211**
- **Move.B (A1), D1 →0001 001000 010001→3211**

Machine Code Example 3

MEMORY LOCATION	MACHINE CODE				ASSEMBLY CODE	
00001000			1		ORG	$1000
00001000	203C	00000012	2		MOVE.L	#$12,d0
00001006	4281		3		CLR.L	d1
00001008	1239	00002000	4		MOVE.B	data,d1
0000100E	D200		5		ADD.B	d0,d1
00001010	13C1	00002001	6		MOVE.B	d1,result
00001016	4E75		7		RTS	
			8			
00002000			9		ORG	$2000
00002000	24		10	data	DC.B	$24
00002001			11	result	DS.B	1
00002002			12		END	$1000

ASSEMBLY CODE	INSTRUCTION FORMAT	MACHINE CODE
MOVE.L #$12,d0	00 10 000 000 111 100	203C 00000012
MOVE.B data,d1	00 01 001 000 111 001	1239 00002000

FULL EXTENSION WORD FORMAT

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	REGISTER			W/L	SCALE		1	BS	IS	BD SIZE		0	I/IS		
BASE DISPLACEMENT (0, 1, OR 2 WORDS)															
OUTER DISPLACEMENT (0, 1, OR 2 WORDS)															

MOVE	B/W L		0 0	0 0	X 1	X 0	R R	R R	R R	M M	M M	M M	e e	e e	e e	e e	e e	e e	s --> d s --> d

[1] **Opcode Bit Pattern Codes:**

A: Address Register Number C: Test Condition D: Data Register Number E: Destination Effective Address e: Source Effective Address
M: Destination EA Mode P: Displacement Q: Quick Immediate Data R: Destination Register r: Source Register
S: Size (00: B, 01: W, 10: L) XX: Move size (01:B, 11:W)

3. Mode categories

Type	Mode	Register	Generation	Assembler Syntax
Data Register Direct	000	Reg. No.	EA=Dn	Dn
Address Register Direct	001	Reg. No.	EA=An	An
Register Indirect	010	Reg. No.	EA=(An)	(An)
Post-increment Reg. Ind.	011	Reg. No.	EA=(An), An ←An+N	(An)+
Pre-decrement Reg. Ind.	100	Reg. No.	An ←An-N, EA=(An)	-(An)
Reg. Indirect with Disp.	101	Reg. No.	EA=(An)+d_{16}	d_{16}(An)
Indexed Reg. Ind. w/ Disp	110	Reg. No.	EA=(An)+(Xn)+d_8	d_8 (An, Xn)
Absolute Short	111	000	EA=(Next Word)	XXX
Absolute Long	111	001	EA=(Next Two Words)	XXXXXX
PC relative with Disp.	111	010	EA=(PC)+d_{16}	d_{16}(PC)
PC rel. w/ Ind. and Disp.	111	011	EA=(PC)+(Xn)+d_8	d_8(PC+Xn)
Immediate	111	100	Data=Next Word(s)	#XXX

$001000	20	3C
$001002	00	00
$001004	00	12
$001006	42	81
$001008	12	39
$00100A	00	00
$00100C	20	00
$00100E	D2	00
$001010	13	C1
$001012	00	00
$001014	20	01
$001016	4E	75

	data	result
$002000	20	00

ASSEMBLY CODE	INSTRUCTION FORMAT	MACHINE CODE
MOVE.L #$12,d0	00 10 000 000 111 100	203C 00000012
MOVE.B data,d1	00 01 001 000 111 001	1239 00002000

Machine Code Test

- 1. MOVE.B D2, D3
- 2. MOVE.B #$35, D2
- 3. MOVE.B D1, (A1)+
- 4. MOVE.B (A2)+, D7

```
SS: 00(B), 01(W), 10(L)
RRR: Dst Reg
eeeeee: Src EA (Mode+ Reg.No)
EEEEEEE: Dst EA (Mode+Reg. No)
MMM: Dst EA Mode


     Mode + Reg. No
------------------------------
Dn        000nnn
An        001nnn
(An)      010nnn
(An)+     011nnn
Imm       111100


Solution
------------------------------------
1. MOVE.B D2, D3
         00XXRRRMMMeeeeee
         0001011000000010 --> 1602
              ---      ---
               D3       D2
```

Instruction Summary

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

Instruction Summary

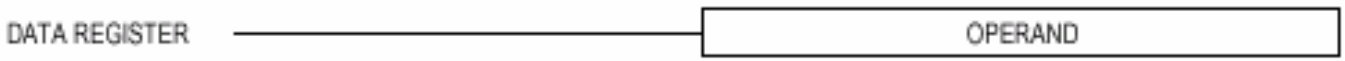
Opcode	Operation	Syntax
MOVE USP	If supervisor state then USP → An or An → USP else TRAP	MOVE USP,An MOVE An,USP
MOVEC	If supervisor state then Rc → Rn or Rn → Rc else TRAP	MOVEC Rc,Rn MOVEC Rn,Rc
MOVEM	Registers → Destination Source → Registers	MOVEM register list,<ea> MOVEM <ea>,register list
MOVEP	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 → 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

Data Reg.Direct Mode

the effective address field specifies the data register containing the operand.

GENERATION:	EA = Dn
ASSEMBLER SYNTAX:	Dn
EA MODE FIELD:	000
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	0



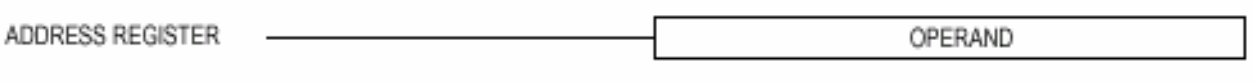

```
move.W    D3, D4      ;D3 source,   D4 destination

                      Before:  D3 =100030FE
                               D4=8E552900

                      After:   D3=100030FE
                               D4=8E5530FE
```

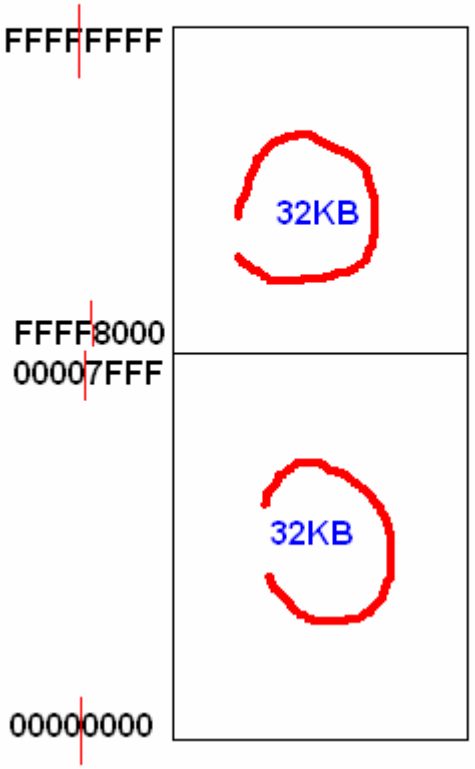
Address Reg. Direct Mode

The effective address field specifies the address register containing the operand.

GENERATION:	EA = An
ASSEMBLER SYNTAX:	An
EA MODE FIELD:	001
EA REGISTER FIELD:	REG. NO.
NUMBER OF EXTENSION WORDS:	0

"Short Addressing" --fast access inside the range
Word operation, instead of
Long Word Operation





Address Register Mode involves, usually, 32-bit (Long Word) operation.
Since Address is expressed by a Long Word.

```
movea.L   A5, A2
                    ;After the operation   (A5)=(A2)
```

When, in address register mode, size in NOT in long word:
the EA's upper word is determined by the "sign extended from lower word"

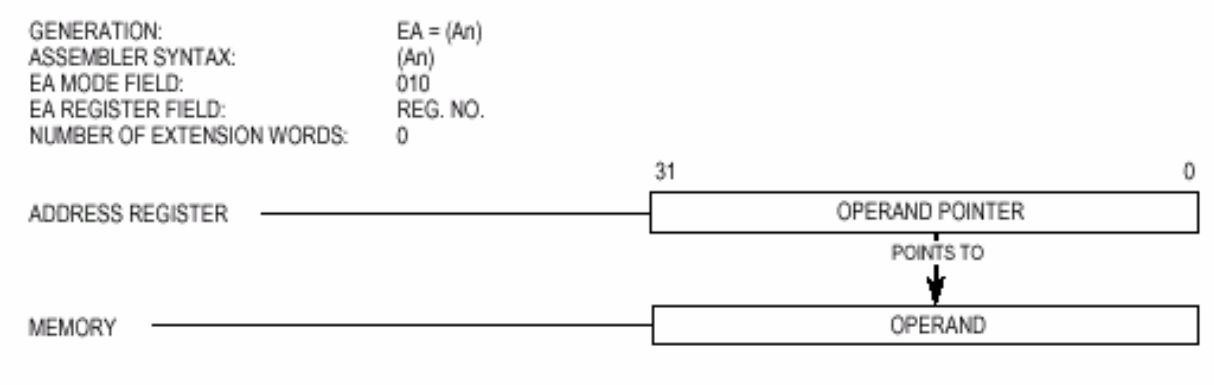
EXample:

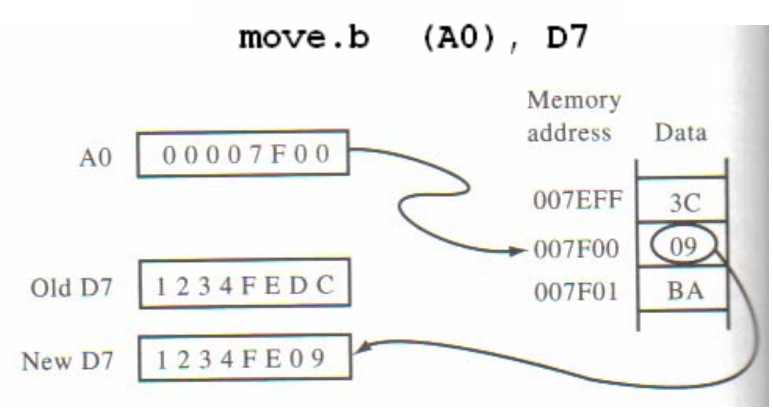
Before: A0=00006800
A1=0000C580

```
movea.W   A0, A2          After: A2=00006800

movea.W   A1, A2          After: A2=FFFFC580
```

Address Reg. Indirect Mode

- The effective address field specifies the address register containing the address of the operand in memory.



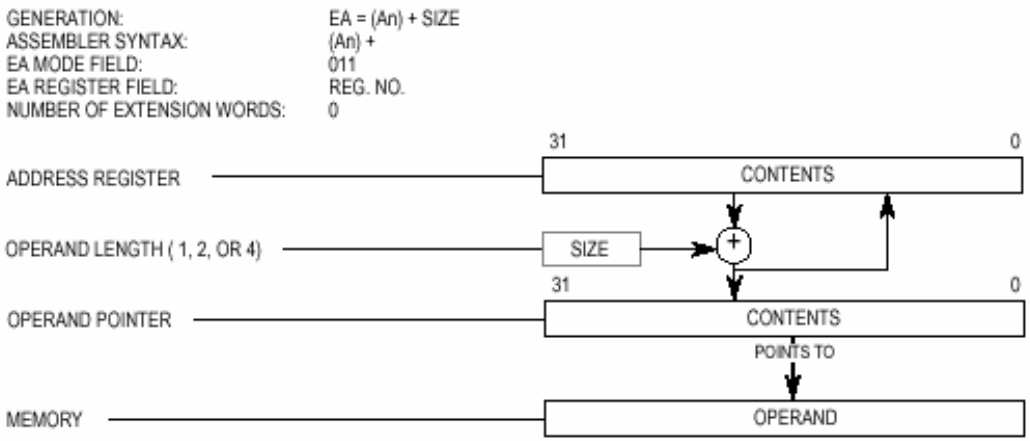


What would be the content of D7 after

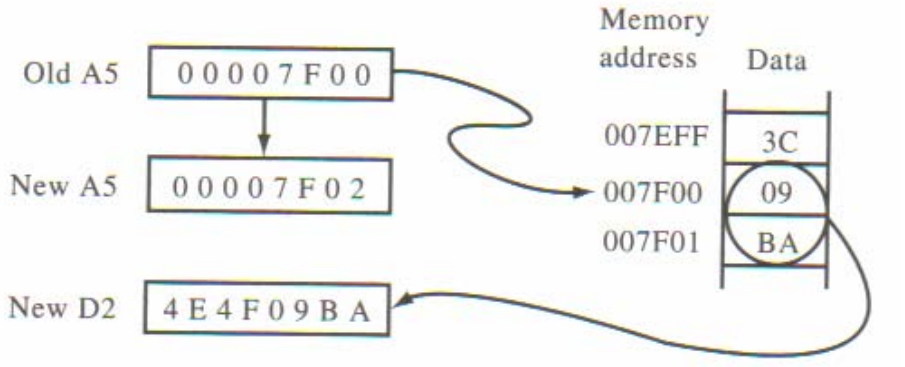
move.W (A0), D7 ?

Address Reg. Indirect with Post-increment Mode

- The effective address field specifies the address register containing the address of the operand in memory.
- After the operand address is used, it is incremented by **one, two, or four** depending on the **size of the operand: byte, word, or long word, respectively.**

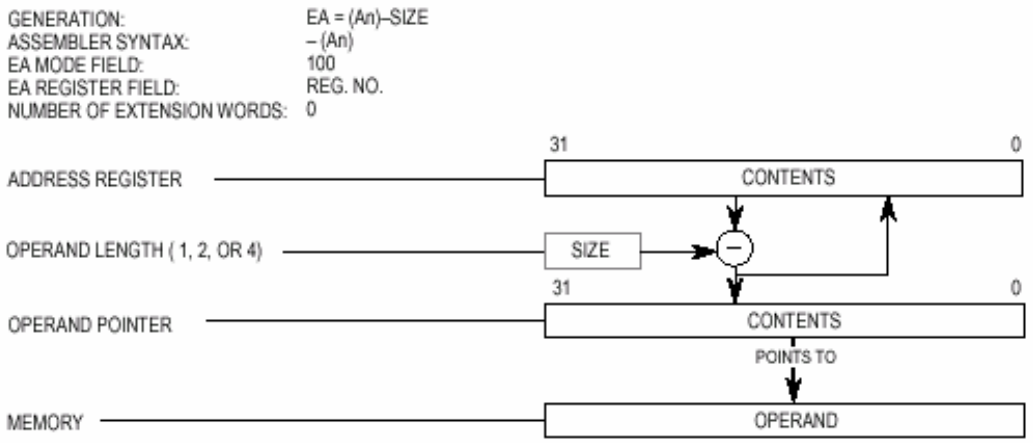



```
move.W   (A5)+, D2
```

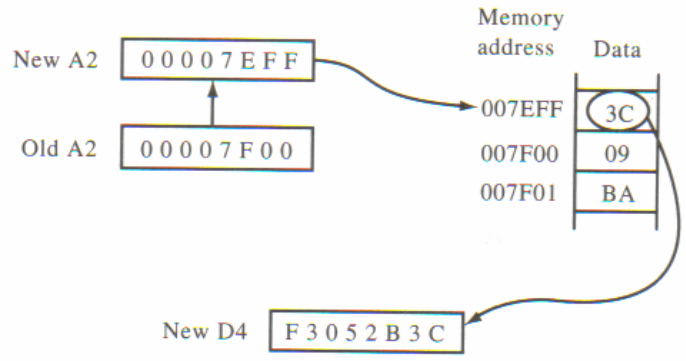


Address Register Indirect with Pre-decrement Mode

- The effective address field specifies the address register containing the address of the operand in memory.
- Before the operand address is used, it is decremented by **one, two, or four** depending on the operand size: **byte, word, or long word**, respectively.

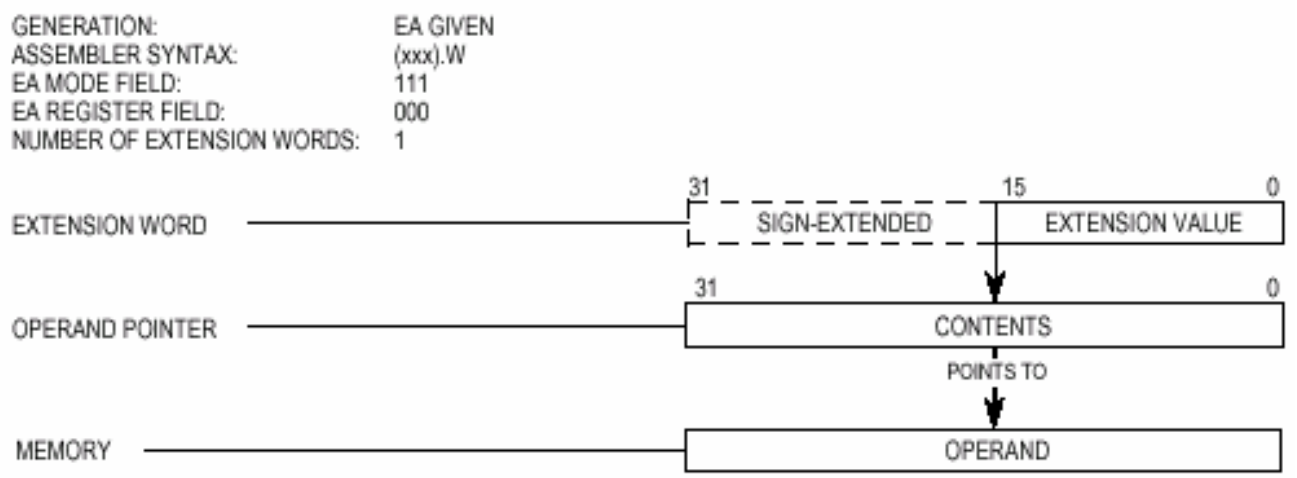



```
move.b  -(A2), D4
```



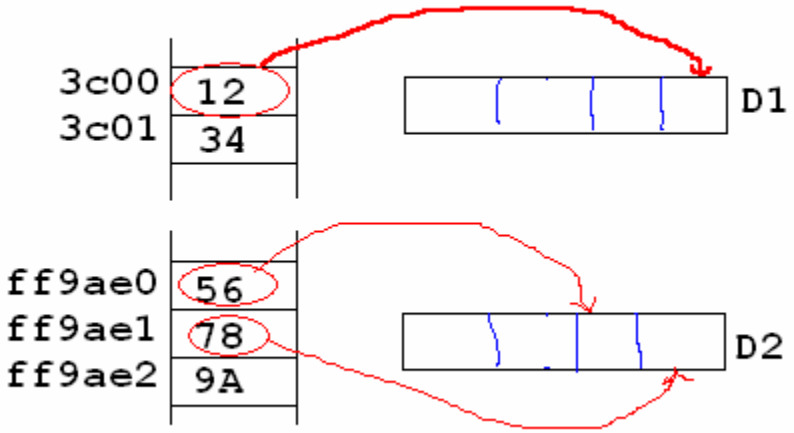
Absolute Short Addressing Mode

- the address of the operand is in the extension word.
- The 16-bit address is sign-extended to 32 bits before it is used.



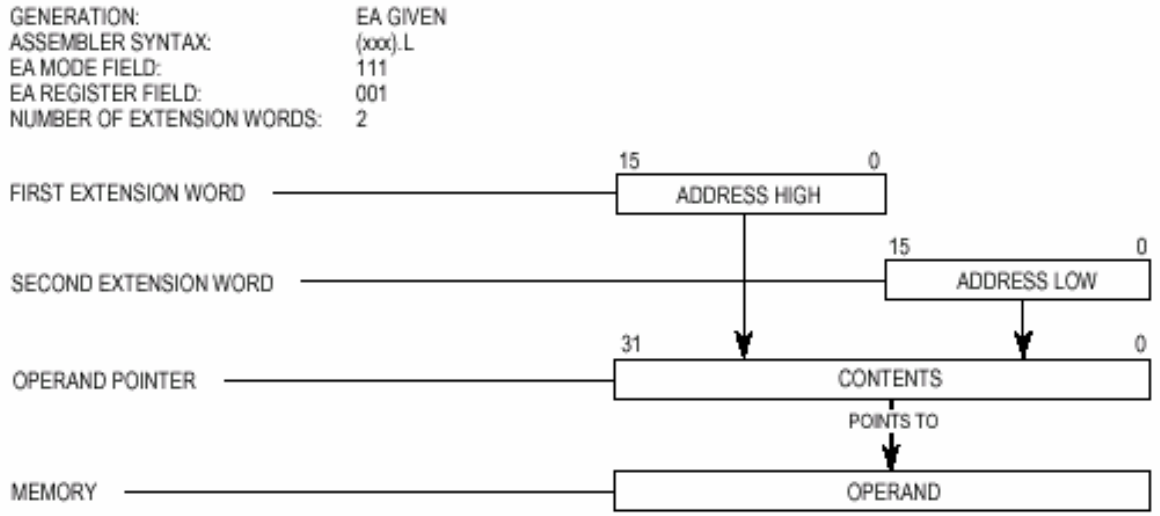
Example:

```
move.b   $3C00, d1

MOVE.W   $9AE0, D2
```



Absolute Long Addressing Mode

- the operand's address occupies the two extension words following the instruction word in memory.
- The first extension word contains the high-order part of the address; the second contains the low-order part of the address.




```
move.b $2e000, D0
```

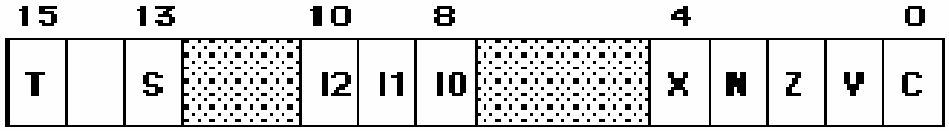
Immediate Mode

- the operand is in one or two extension words.

GENERATION:	OPERAND GIVEN
ASSEMBLER SYNTAX:	#<xxx>
EA MODE FIELD:	111
EA REGISTER FIELD:	100
NUMBER OF EXTENSION WORDS:	1,2,4, OR 6, EXCEPT FOR PACKED DECIMAL REAL OPERANDS

```
D5: 12345678

(1) move,b    #$3A,D5          New D5=?
(2) move.w    #$9E00, D5       New D5=?
(3) move.l    #1, D5           New D5=?
```

CCR and Condition Code



Bit	Meaning
T	Tracing for run-time debugging
S	Supervisor or User Mode
I	System responds to interrupts with a level higher than I
C	Set if a carry or borrow is generated. Cleared otherwise
V	Set if a signed overflow occurs. Cleared otherwise
Z	Set if the result is zero. Cleared otherwise
N	Set if the result is negative. Cleared otherwise
X	Retains information from the carry bit for multi-precision arithmetic

Condition Codes

- Most instructions affect the state of the five flags
  - N (Negative flag):
    - 1 (set): MSB of the result is 1 (set)
    - 0 (cleared): otherwise

```
move.b   #$3F, D0
addi.b   #1,D0
```

```
3F
01
40 -->0100 0000      N=0
```

```
move.b   #$7F,D0
addi.b   #1,D0
```

```
7F
01
80 --> 1000 0000    N=1
```

Condition Codes

- Z(Zero Flag)
  - Set (1): result equals zero
  - Clear(1): otherwise

```
Initial value of D0 = 00000003
subi.b   #1,D0  ; D0=0000 0002    Z=0
subi.b   #1,D0  ; D0=0000 0001    Z=0
subi.b   #1,D0  ; D0=0000 0000    Z=1
```

- V (Overflow Flag)
  - Set: a result represents a sign change
  - Clear: no sign change before and after an operation

```
                                  Word  Byte
move.b   #$77, D0 ;D0=0000 0077   MSB=0   V=x
addi.b   #3, D0   ;D0=0000 007A   MSB=0   V=0
addi.b   #9, D0   ;D0=0000 0083   MSB=1   V=1
subi.b   #1, D0   ;D0=0000 0082   MSB=1   V=0
subi.b   #4, D0   ;D0=0000 007E   MSB=0   V=1
                  Long Word
```

Condition Code

- C (Carry Flag)
  - Set :
    - Carry out of the MSB of the result (addition)
    - Borrow as a result (subtraction)
  - Clear: otherwise

```
move.b   #6, D0   ;D0=0000 0006          C=x
subi.b   #1, D0   ;D0=0000 0005          C=0

move.b   #6, D0   ;D0=00000006           C=x
subi.b   #9, D0   ;D0=FFFFFFFD (borrow)  C=1
```

```
Addition with 2's
complement

00000006
2's Complement of 9
FFFFFFF7
--------
FFFFFFFD (no carry)
 that means there
 WAS borrow
```