Address Indirect Addressing with Index and Displacement – Mode 6

Opcode - 4	dRn - 3	dmd - 3	sMS - 6

rt - 1	xm - 3	s -1	und - 3	displacement - 8

rt – type of register used – address (1) or data (0) register
xm – address of index register used (3-bits)
s – index register defined as longword (1) or word (0)
und – undefined

Example: move $E(a5, a2.w), d1

Equivalent machine instruction: 0011 001000 110 101
1 010 0 000 00001110

24 bits EA of source = a5 (32 bits) + a2 (16 low order bits – sign extended) + displacement (8 bits – sign extended)

Example of Mode 6 (with index and displacement

Given an array of *m x n*

	0	1	2	3	…..	n
0	(0,0)	(0,1)	(0,2)	(0,3)	..	..
1	(1,0)	(1,1)	(1,2)	..	…	..
….	..	..				
m	..					

a1 and a2 registers are initialized to 0000 0000

a1 is used to point at rows
a2 is used to point at columns
X is just a reference point

```
        move X(a1, a2), d1
LOOP2
LOOP1   Add #2, a2
        …
        Do LOOP1 "n" times
        Add #2, a1
        Do LOOP2  "m" times
```

The elements are stored in the memory as follows

X	(0,0)	x + a1 + a2
	(0,1)	
	(0,2)	
	…	
	(0,n)	
	(1,0)	
	(1,1)	
	….	
	(1,n)	
	(2,0)	
	…..	

Address Register Indirect Addressing with Post-increment – Mode 3

0011	010 000	011	001

Example: move.w (a1)+, d2

This means –

EA = [a1]

a1 ← a1 + const ; here const = 2

Increment (const) depends on the data size provided in opcode. It can take byte, word, or longword

Suppose initially –

a1 = 0000 1230

d2 = 87C3 187A

Mem. Loc. 001230 = 320D 0005

001234 = ???? ????

Then after move.w (a1)+, d2 → a1 = 0000 1232

d2 = 87C3 320D

But after move.b (a1)+, d2 → a1 = 0000 1231

d2 = 87C3 1832

And after move.l (a1)+, d2 → a1 = 0000 1234

d2 = 320D 0005

Address Register Indirect Addressing with Pre-decrement – Mode 4

0011	010 000	100	001

Example: move.w -(a1), d2

This means –

a1 ← a1 - const ; here const = 2

d0 ← M[a1]

Increment (const) depends on the data size provided in opcode. It can take byte, word, or longword

Suppose initially –

a1 = 0000 1230

d2 = 87C3 187A

Mem. Loc. 00122C = ABCD 5678

001230 = 320D 0005

Then after move.b -(a1), d2 → a1 = 0000 122F

d2 = 87C3 1878

But after move.w -(a1), d2 → a1 = 0000 122E

d2 = 87C3 5678

And after move.l -(a1), d2 → a1 = 0000 122C

d2 = ABCD 5678

Absolute Addressing (short or long) – Modes 70 and 71

Absolute Short – absolute address restricted to 16 bits, but is sign extended to 32 bits at run time

0011	001 000	111	000
Displacement (16 bits)			

Absolute Long – absolute address is 32 bits

0011	001 000	111	001
Displacement (16 bits)			
Displacement (16 bits)			

Example: move.b $14, d1 ➔ EA = const (sign-ext)
move.b ($61234).w, d1 ➔ Absolute short – will consider $001234 as EA
move.b ($61234).l, d1 ➔ Absolute long – will consider 24 bits $061234 as EA

PC Relative Addressing (with Displacement) – Mode 72
(with Index and Displacement) – Mode 73

Example: move.w $30(PC), d1

Displacement is a word

EA = PC + displ + 2

Another Example: here we consider the case when a Relative Expression is used to specify the displacement

move.w num(PC), d1

This means that the “word” at location “num” is moved to d1, where EA = “num”.
However the calculation of Effective Address is made via PC-relative mode.

Let, value of Symbol “num” is $000030

Suppose initially PC = $000034 where “move” instruction is.
Then,

EA = PC + 2 + displ

→ 000030 = 000034 + 2 + displ

→ displ = $FFFA

num	F286	000030
	????	000032
	Move instr	000034 = PC
	displ = FFFA	000036
	….	

This “relative expressions” is used to write <u>Position-Independent</u> code.

Position-dependent vs Position-independent code

Position-dependent code

Location	Object	Code		Src	Code
000000				org	$0000
000000	90C8			sub	a0, a0
000002	3228	0016		move	X(a0), d1
000006	3439	00000018		move	X+2, d2
00000C	C5C1			muls	d1,d2
00000E	3039	0000001A		move	X+4, d2
000014	4E40			trap	#0
000016	000B		X	dc	11
000018	0029			dc	41
00001A	0003			dc	3
00001C				end	

The program above will not execute as expected if it is loaded at any other location than $0000

Position-independent code using PC-relative mode

Location	Object	Code		Src	Code
000000				org	$0000
000000	90C8			sub	a0, a0
000002	323B	000E		move	X(pc,a0.l), d1
000006	343A	000C		move	X+2(pc), d2
00000A	C5C1			muls	d1,d2
00000C	303A	0008		move	X+4(pc), d2
000010	4E40			trap	#0
000012	000B		X	dc	11
000014	0029			dc	41
000016	0003			dc	3
000018				end	

The program above solves the problem. It can start from any location, and will execute as expected.

Immediate Addressing – Mode 74

Example: movei.w #1234,d1 → d1 = ???? 04D2

When Immediate data is 8-bits

0000 0110	00	000	001
Unused (8 bits)	IData (8 bits)		

When Immediate data is 16-bits

0000 0110	01	000	001
IData (16 bits)			

When Immediate data is 32-bits

0000 0110	10	000	001
IData (16 bits)			
IData (16 bits)			

Quick Data – no mode associated with this addressing

Opcode (4-bits)	Qdata (3-bits)	Opcode (1 bit)	om (2-bits)	dMS (6-bits)

By examining the 5-bit total opcode, it knows it is quick data.
om (2-bits) specify if the data considered is to be extended to a byte (00), word (01) or longword (10).

Example:

addq #5, d1

0101	101	0	10	000	001

moveq #$45, d1

0111	Data (3-bits)	0	Data (5-bits)	Dn (3-bits)
0111	010	0	00101	001

If we need to move an 8-bit data to a data register, "moveq" instruction will occupy one word in memory, whereas "movei" instruction will occupy two words in memory.