



Immediate Value to Data Register

- In previous examples we have referred to either a **memory location** or a **data register**.
 - > A location where the data is stored.
- Sometimes we need to specify a given value known as an **immediate value**.
 - > Use immediate addressing mode.

9th Lecture, Michael Manzik, Page: 1



Example: Move Value 10 into D2

- > Source mode = immediate mode = 111
- > Source register = immediate mode = 100

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	Size	Destination Register	Mode	Mode	Source Mode	Source Register								
0	0	11	010	000	111	100									

0 0 11 010 000 111 100 -> \$343C

9th Lecture, Michael Manzik, Page: 2



Immediate Values Symbol

- We distinguish immediate values from addresses using **#** symbol:
 - #1000** = Decimal value 1000
 - #\$1000** = Hexadecimal value 1000
 - 1000** = Address 1000 (decimal)
 - \$1000** = Address 1000 (hexadecimal)

9th Lecture, Michael Manzik, Page: 3



Mnemonics

Template for a move instruction.

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

- Difficult to remember template for each instruction.
- We'll replace this with the symbol (mnemonic):
 - move**
- Replace the size field with the **b**, **w** and **l** subscripts
 - move.b** -> move a **byte**
 - move.w** -> move a **word**
 - move.l** -> move a **long word**

9th Lecture, Michael Manzik, Page: 4



Encode Operands and the Destination/Source Modes

- The operands and the destination /source modes are encoded as arguments following the symbol(**src**, **dest**):
 - move.b d0,d1**
 - move.w \$2000,d5**
 - move.l \$2000, \$200A**
 - move.l #10, d2**
- Assembler syntax for the move command is:
 - move <ea> <ea>**
- <ea>** effective address = an expression which, when evaluated, produces a value.

9th Lecture, Michael Manzik, Page: 5



Move

MOVE

Move Data from Source to Destination

MOVE

Operation: Source → Destination

Assembler Syntax: MOVE (ea), (ea)

Attributes: Size = (Byte, Word, Long)

Description: Moves the data at the source to the destination location, and sets the condition codes according to the data.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X Not affected.

N Set if the result is negative. Cleared otherwise.

Z Set if the result is zero. Cleared otherwise.

V Always cleared.

C Always cleared.

9th Lecture, Michael Manzik, Page: 6



Defaults

- In order to simplify coding of assembly language, certain defaults are assumed:
 - move -> move.w (i.e.: word size is default)
 - move 10,d2 -> absolute addressing
 - 10 -> decimal 10

9th Lecture, Michael Manzke, Page: 7



Addressing Restrictions

Only **Bytes** may be read from **ODD** memory addresses. **Words and Long-words** must be read from **EVEN** memory addresses!

- This was a design decision to simplify memory interfacing.
- Accessing words or long-words on odd boundaries causes the 68000 to terminate the F->D->E cycle.
- -> An address error occurs

9th Lecture, Michael Manzke, Page: 8



Which of these instruction is legal?

- a) move.w \$4001, d0 ✗
- b) move.b \$4004, d1 ✓
- c) move.b \$4003, d3 ✓
- d) move.w \$4001, \$4003 ✗
- e) move.b \$4000, \$4001 ✓

9th Lecture, Michael Manzke, Page: 9