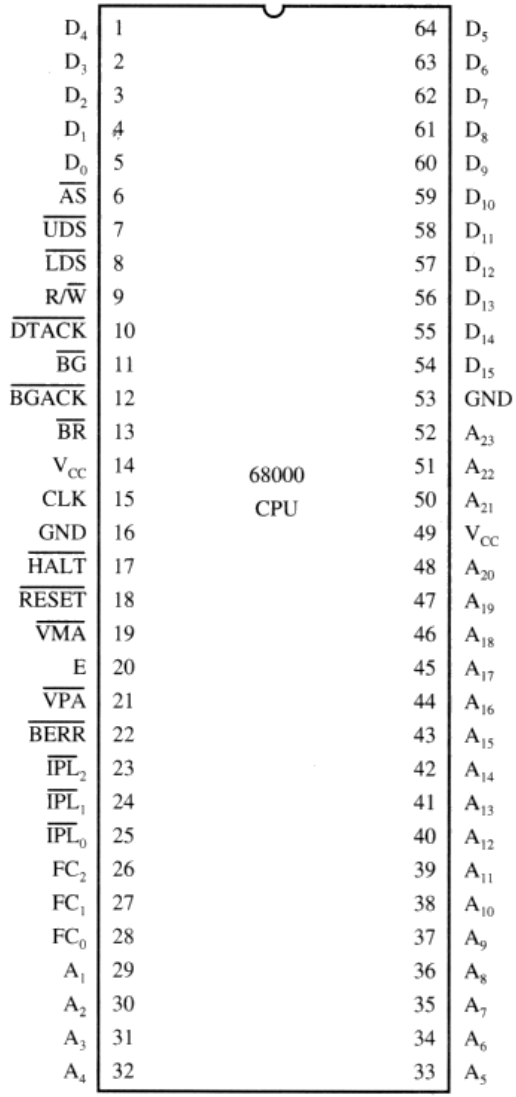
Lecture 2: MC68000 architecture

- **General information**
- **Programmer's model**
- **Memory organization**
- **Instruction format**
- **Our first assembly program**
- **The `sim68k` simulator**
- **Machine translation**

MC68000 general information

- **Specifications**
  - 32-bit Data and Address Registers
  - 16-bit Data Bus
  - 24-bit Address Bus (16MB)
  - 14 Addressing Modes
  - Memory-Mapped Input/Output
  - Program Counter
  - 56 Instructions
  - 5 Main Data Types
  - 7 interrupt levels
  - Clock speeds: 4MHz to 12.5MHz
  - Synchronous and asynchronous data transfers

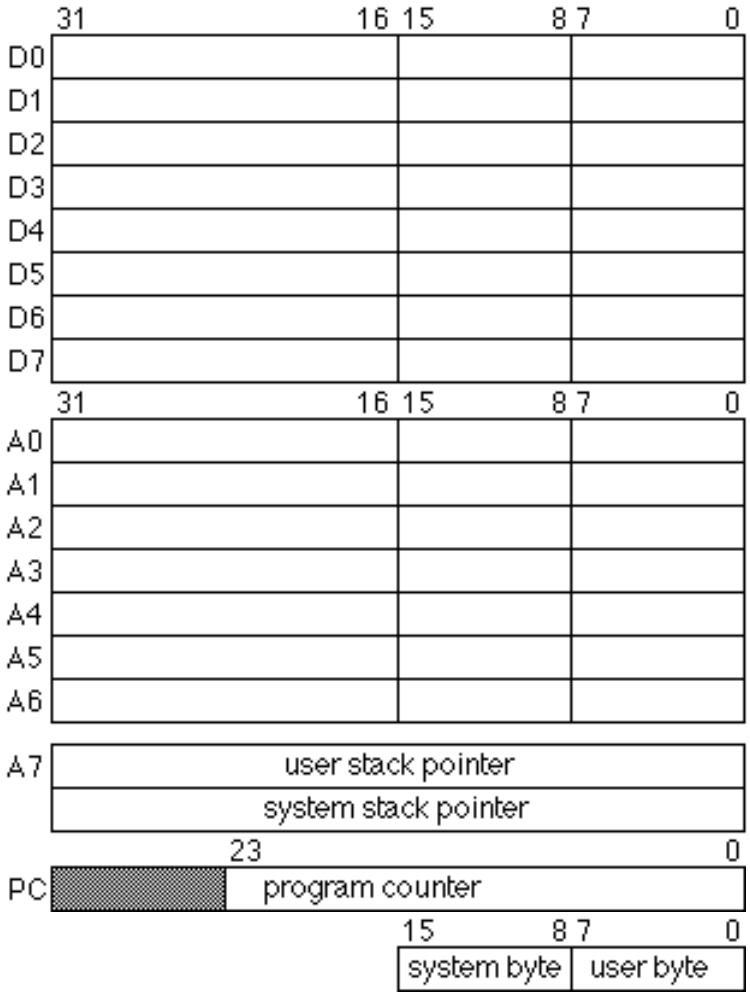


What is a "microprocessor architecture"?

- **For our purposes the architecture is the software or programmer's model of the microprocessor**
  - The **CPU registers** available to the programmer
  - The basic **instructions** the CPU can perform
  - The ways these instructions can **specify a memory location**
  - The way **data is organized** in memory
  - How the CPU accesses & controls **peripheral** devices

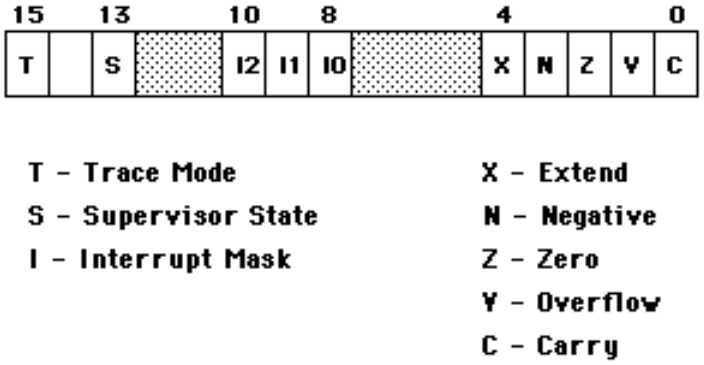
MC68000 register set

- **8 data registers (D0-D7)**
- **8 address registers (A0-A7)**
  - There are TWO A7 registers
    - User Stack Pointer (USP)
    - Supervisor Stack Pointer (SSP)
- **Program Counter (PC)**
- **Status Register / Condition Code Register (SR/CCR)**



Status/Condition Code Register

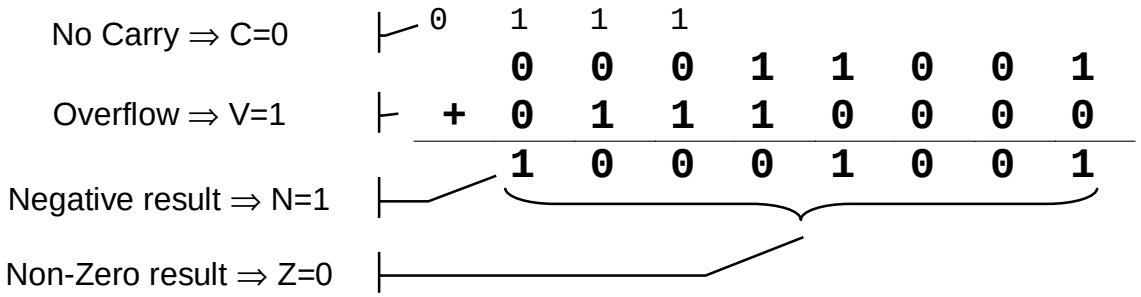
- **More significant byte: SR**
  - Only modifiable in supervisor mode
  - Details in later sections
- **Least significant byte: CCR**
  - For user-level programs
  - Behavior depends on instruction



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 Code Register example

- **Add the numbers $19 and $70 and show the effect on the CCR bits**
  - The following piece of code performs the operation
    - **`MOVE.B #$70,D0`**
    - **`ADD.B  #$19,DO`**
  - To illustrate the behavior of the CCR we will perform an addition in base 2



Data representation (review)

- **Three bases will be used thoroughly**
  - Decimal
    - $321_{10} = 3\times10^2 + 2\times10^1 + 1\times10^0$
  - Hexadecimal
    - $AB6_{16} = 10\times16^2 + 11\times16^1 + 6\times16^0$
  - Binary
    - $101_{2} = 1\times2^2 + 0\times2^1 + 1\times2^0$
- **Express the following numbers in base 10**
  - 0111_{10}
  - $FF05_{16}$
  - 1000_{2}

2's complement (review)

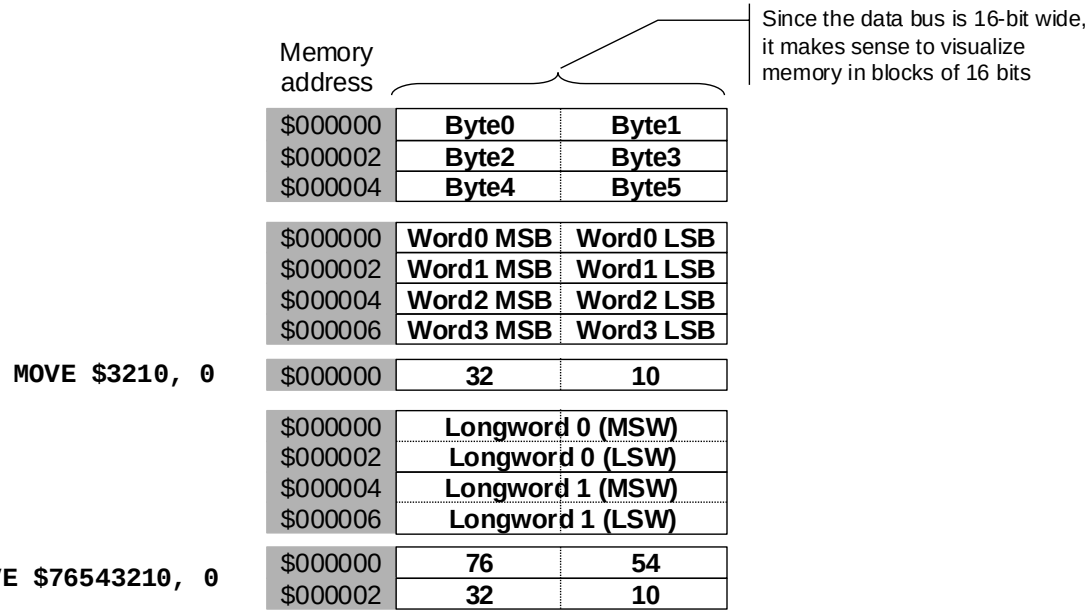
- **How to express negative numbers in binary?**
  - Sign-Magnitude:
    - Use the Most Significant Bit to encode the sign
      - $0111_2=+7_{10}$
      - $1111_2=-7_{10}$
  - 2s Complement:
    - The most commonly used form

Binary number ($+5_{10}$)		**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
		↓	↓	↓	↓	↓	↓	↓	↓
1s complement		**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
Add 1	+								**1**
2s complement (-5_{10})		**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**

    - Subtraction is made *very* easy (perform the operation 5-7)
    - The range of numbers that can be represented is from $\mathbf{-2^{n-1}}$ to $\mathbf{+2^{n-1}-1}$

Memory organization

- **24-bit addresses**
  - 16MB can be addressed
- **3 data widths**
  - BYTE: 8-bit, can be at even or odd address
  - WORD: 16-bit, must be at x2 address
  - LONGWORD:32-bit, must be at x4 address
- **Big-endian order**
  - Words are stored with the lower 8-bits in the higher of the two storage locations
    - As opposed to little-endian processors, like the Intel 80x86 family



Register Transfer Language (RTL)

- **A notation that describes the micro-processors actions clearly and unambiguously**
- **We'll use a simplified version**
  - 100 means "#100"
  - [M(4)] means "contents stored in memory location 4"
  - [MAR] means "contents stored in MAR"
  - [M(4)]=100 means "memory location 4 contains #100"
  - [MAR]=100 means "MAR contains #100"
  - [PC]←4 means "load number 4 onto PC"
  - [M(4)]←100+[M(4)] means "add #100 to contents of memory location 4 and save"

MC68000 instructions

- **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

- **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

Instruction operands

- **Operands can be**
  - registers
  - constants
  - memory addresses
- **Basic addressing modes (to be expanded)**
  - D_n: data register direct `MOVE.L D0, D1`
  - A_n: register indirect `MOVE.L (A0), D1`
  - #n: immediate `MOVE.L #10, D1`
  - n: absolute `MOVE.L $08FF00, D1`
- **Immediate operands can be specified in several formats**
  - Hexadecimal: prefixed by $
  - Octal: prefixed by @
  - Decimal: prefixed by & (or nothing)
  - Binary: prefixed by %
  - ASCII: within single quotes ‘abc’

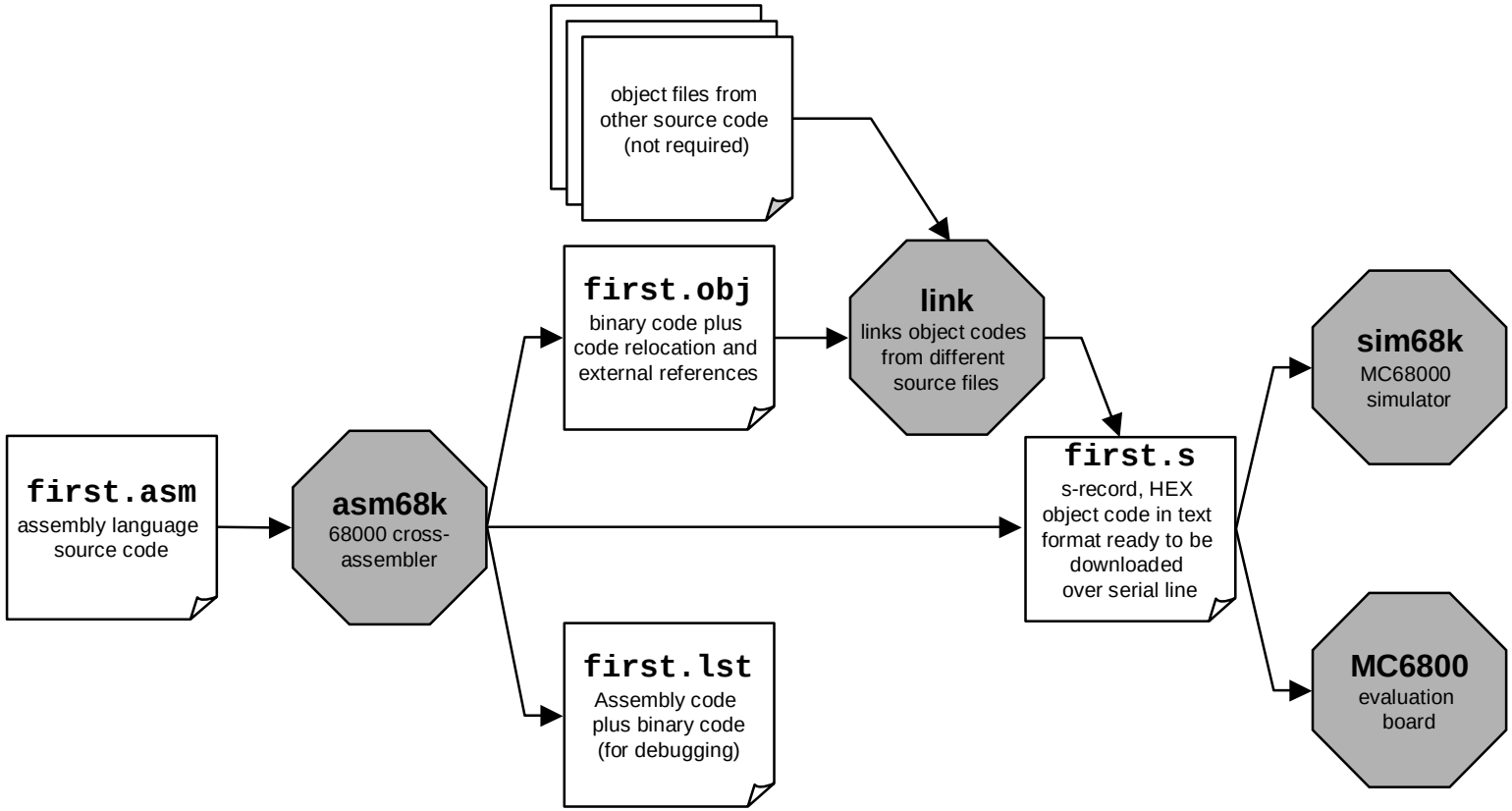
Our first program: `first.asm`

LABEL	OPCODE	OPERAND(S)	COMMENTS
	ORG	$1000	;start of 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	DC.B	$24	
result	DS.B	1	;reserve a byte for result
	END	$1000	;end of program and entry point

PROGRAM AREA

DATA AREA

Source program assembly



Compilation and execution steps

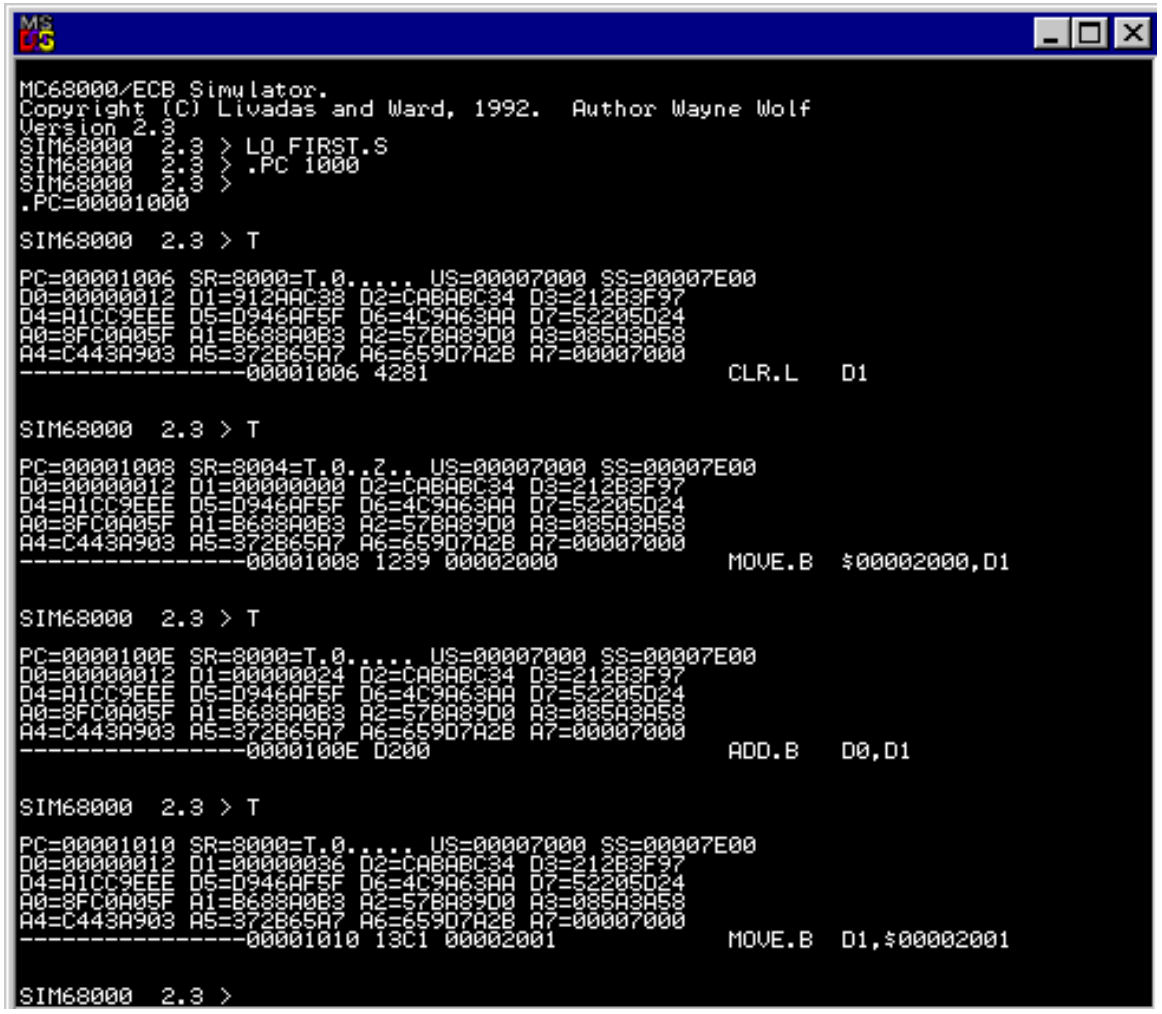
- **Compilation steps**
  - Write source in text editor and save (**`first.asm`**)
  - Create binary with assembler (**`asm68k first.asm -l`**)
    - `-l` flag creates list file (**`first.lst`**) with assembly and binary code
  - Link with linker --only if multiple source files
- **Execution steps (on simulator)**
  - Start simulator (**`sim68k`**)
  - Load program (**`LO first.s`**)
  - Execute
    - Directly (**`GO 1000`**) or
    - Step by step (**`T`**)

The sim68k simulator

- **Available on thor.cs.wright.edu and 339 Russ PCs**
- **Type sim68k on command line**
- **Basic simulator commands**

Command	Syntax	Example
Display Formatted	DF	DF
Memory Dump	MD <address> <count> [;DI]	MD 1000 20 ;DI
Memory Modify	MM <address> [;{ ,W,L}]	MM 1000
Register Modify	.<register> [<contents>]	.PC 1000
Block Fill	BF <addr1> <addr2> <data>	BF 1000 2000 FF
Go	GO [<address>]	GO
Trace	T [<address>]	T
Set Breakpoint	BR [<address>]	BR 1000
Clear Breakpoint	NOBR [<address>]	NOBR
Load	LO <fname>	LO FIRST.S
Store	ST <fname> <address> <count>	ST FIRST 100 10
Exit	EX	EX

Run the simulator



How is the program's machine code stored in physical memory?

MEMORY LOCATION	MACHINE CODE	SOURCE LINE		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

$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

Machine code translation

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

MOVE **Move Data From Source to Destination** **MOVE**

Operation: (Source)→Destination

Assembler Syntax MOVE <ea>,<ea>

Attributes: Size=(Byte,Word,Long)

Description: Move the content of the source to the destination location. The data is examined as it is moved, and the condition codes set accordingly. The size of the operation may be specified to be byte, word or long.

Condition Codes:

X	N	Z	V	C
—	*	*	0	0

N Set if the result is negative. Cleared otherwise.
Z Set if the result is zero. Cleared otherwise.
V Always cleared.
C Always cleared.
X Not affected.

Instruction Format:

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

Instruction Fields:

Size field — Specifies the size of the operand to be moved.
01 — byte operation.
11 — word operation.
10 — long operation.

Destination Effective Address field — Specifies the destination location. Only data alterable addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	register number	d(An,Xi)	110	register number
An	—	—	Abs.W	111	000
(An)	010	register number	Abs.L	111	001
(An)+	011	register number	d(PC)	—	—
-(An)	100	register number	d(PC,Xi)	—	—
d(An)	101	register number	Imm	—	—

Source Effective Address field — Specifies the source operand. All addressing modes are allowed as shown:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	register number	d(An,Xi)	110	register number
An*	001	register number	Abs.W	111	000
(An)	010	register number	Abs.L	111	001
(An)+	011	register number	d(PC)	111	010
-(An)	100	register number	d(PC,Xi)	111	011
d(An)	101	register number	Imm	111	100

* For byte size operation, address register direct is not allowed.

Memory organization example