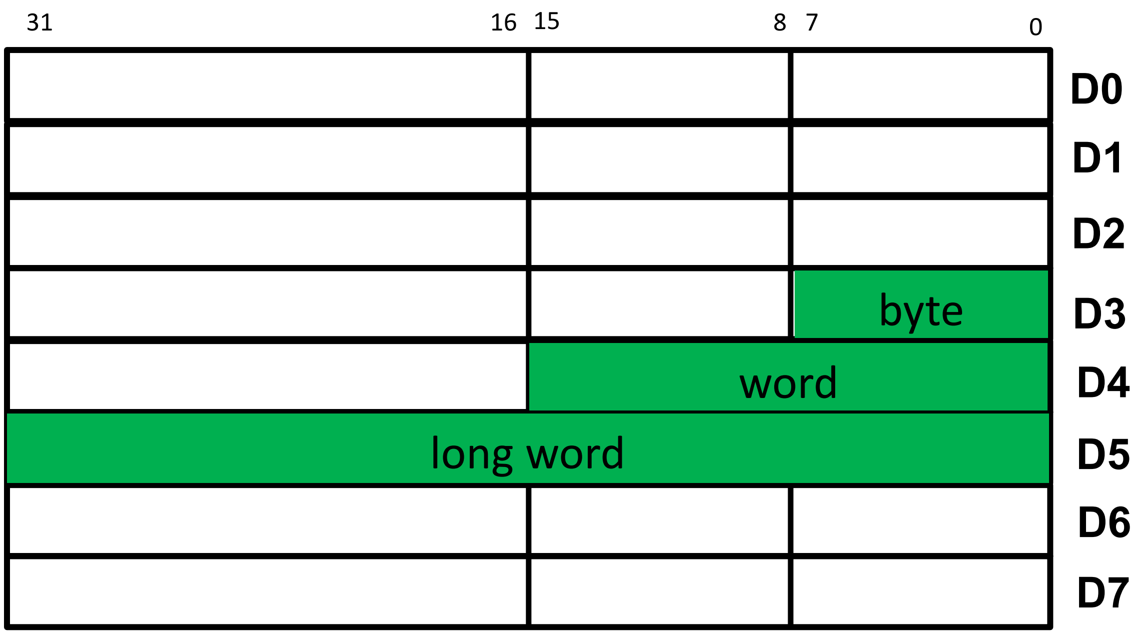
C vs M68000
Programmer's Interface

	C	M68000 ISA
Registers		8, 32b integer, D0-D7 7, 32b address, A0-A6 2, 32b stack pointers, USP & SSP SR, 16b PC, 24b
Memory	local variables global variables	2^{24} linear array of bytes
Data types	int, short, char, unsigned, float, double, aggregate data types, pointers	byte(8b), word (16b) and long-word (32b)
Arithmetic operators	+, -, *, %,++, <, etc.	add, sub, mul, div, etc.
Memory access	a, *a, a[i], a[i][j], a.x	13 address modes
Control	If-else, while, do-while, for, switch, function call, return	bcc, jmp, rts, link, unlink

Data Registers

- Data registers are used to hold data independent of memory
  - 8, 32-bit homogeneous registers
  - Allow byte, word, and long-word operations



Copying Data Between Data Registers

Hexadecimal

- Copy the byte from D0 to D1

```
move.b d0,d1
```

D0	12	34	56	78
D1	00	00	00	78

- Copy the word from D0 to D1

```
move.w d0,d1
```

D0	12	34	56	78
D1	00	00	56	78

- Copy the long-word from D0 to D1

```
move.l d0,d1
```

D0	12	34	56	78
D1	12	34	56	78

C Example Involving Addition

- Consider the C operation for addition

  `a = a + b;`

- Assume that the variables are <u>bytes</u> (i.e., chars) and **`a`** is contained in D1 and **`b`** is contained in D0

- To perform the addition, use the add instruction

```
add.b d0,d1          ;a = a + b
```

BEFORE

D0	12	34	56	78
D1	1E	3C	1A	92

AFTER

D0	12	34	56	78
D1	1E	3C	1A	0A

Complex Operations

- What about more complex statements?

  `a = a + b + c - d`

- Break into multiple instructions

- Assume `a`(D0), `b`(D1), `c`(D2), and `d`(D3) are bytes

```
add.b  d1,d0           ;a = a + b
add.b  d2,d0           ;a = (a + b) + c
sub.b  d3,d0           ;a = (a + b + c) - d
```

Constant Values

- Consider the following C code

```
a = 3;
```

- Often want to be able to specify a constant in an instruction
  - These are called immediate or literal values

- Use the **#** symbol to specify a constant

```
move.b #3,d0        ;a = 3
```

- The immediate is an 8-bit, 16-bit or 32-bit value and must not exceed the range of the size indicator

Constant Values

- Consider the following C code

```
a = 3;
```

- Often want to be able to specify a constant in an instruction
  - These are called immediate or literal values

- Use the **#** symbol to specify a constant

```
move.b #3,d0          ;a = 3
```

- The immediate is an 8-bit, 16-bit or 32-bit value and must not exceed the range of the size indicator

- The immediate value can be specified in hex($), binary(%) or decimal (nothing)

Examples with Different Number Systems

- Initialize the word in D0 with the constant 10

Before

D0 `12 34 56 78`

```
move.w #10,d0
```

after → D0 `12 34 00 0A`

```
move.w #$A,d0
```

after → D0 `12 34 00 0A`

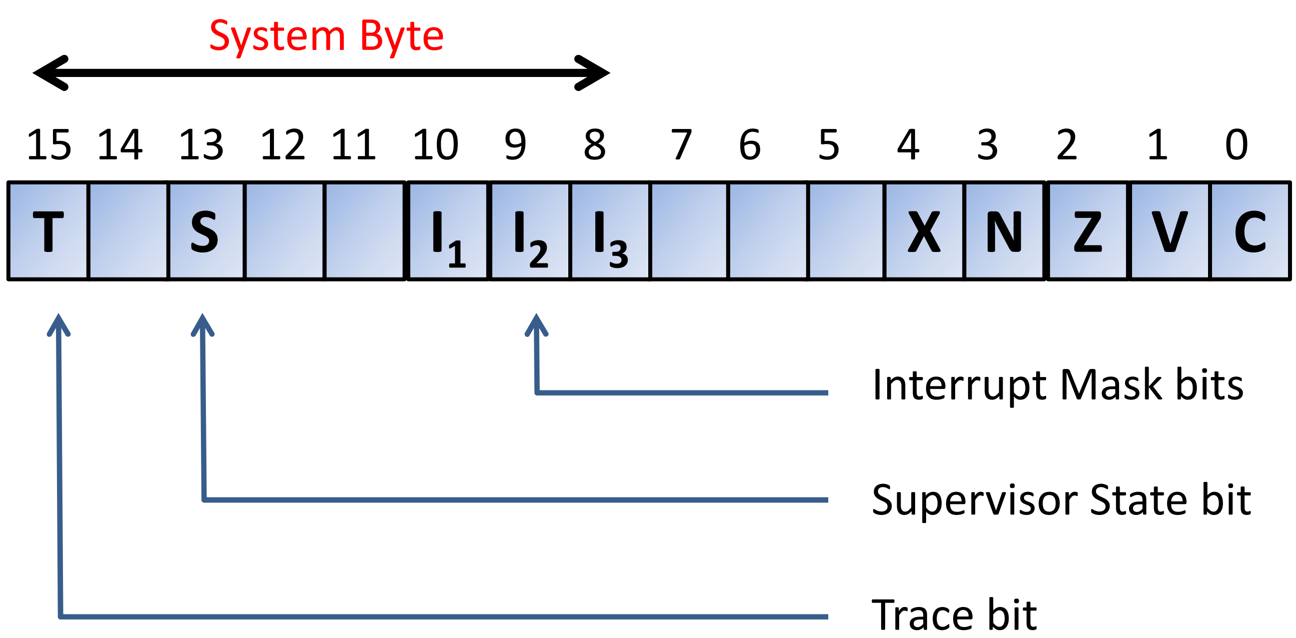
```
move.w #%1010,d0
```

after → D0 `12 34 00 0A`

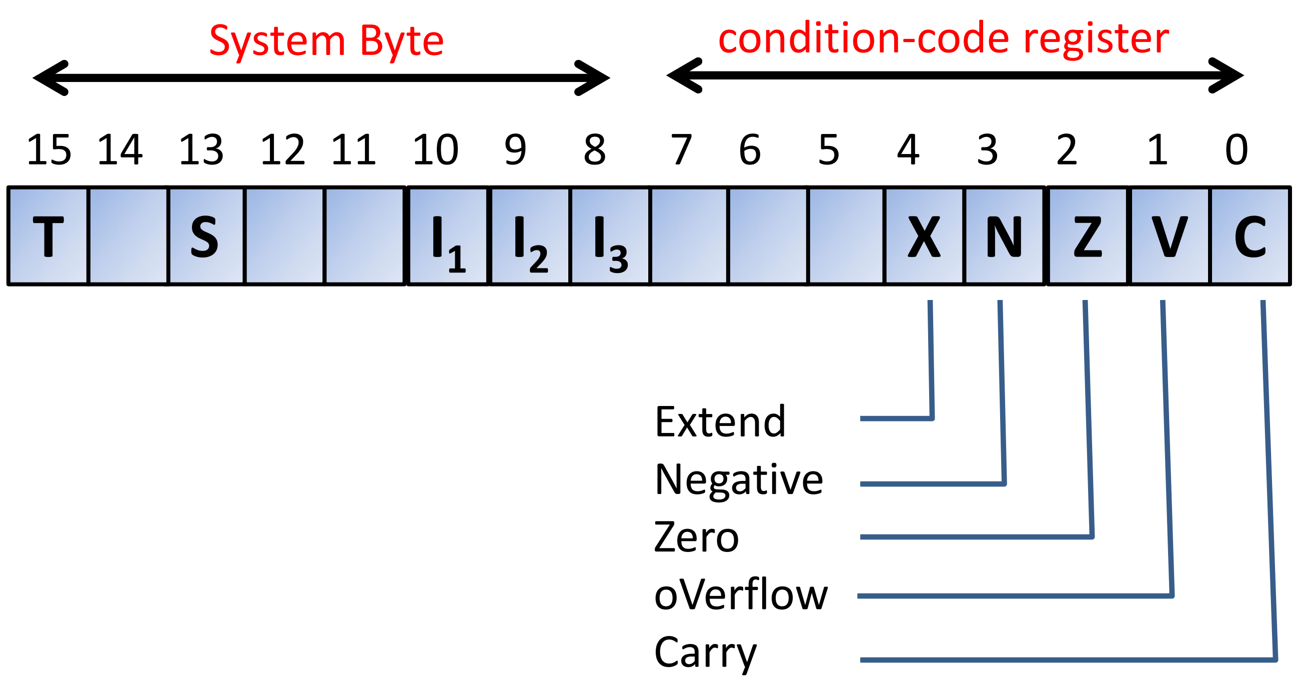
68000 Simple Arithmetic

Instruction	Size	Example	Meaning
Addition	B/W/L	add d0,d1	d1 = d1 + d0
Subtraction	B/W/L	sub d0,d1	d1 = d1 – d0
Multiplication unsigned	W	mulu d0,d1	d1 = d0 x d1; (32-bit result)
Multiplication signed	W	muls d0,d1	d1 = d0 x d1; (32-bit result)
Division unsigned	L	divu d0,d1	d1[15:0] = d1[31:0] / d0[15:0] d1[31:16] = d1[31:0] % d0[15:0]
Division signed	L	divs d0,d1	d1[15:0] = d1[31:0] / d0[15:0] d1[31:16] = d1[31:0] % d0[15:0]
Swap	L	swap d1	d1[15:0] = d1[31:16] d1[31:16] = d1[15:0]
Sign Extension	W/L	ext d1	d1[15:8] = d1[7] d1[31:16] = d1[15]

Status Register



Status Register



- The condition-code register is updated after most instructions execute to reflect the characteristics of the result

Examples

- What are the contents of the condition-code register after the following program segment executes?

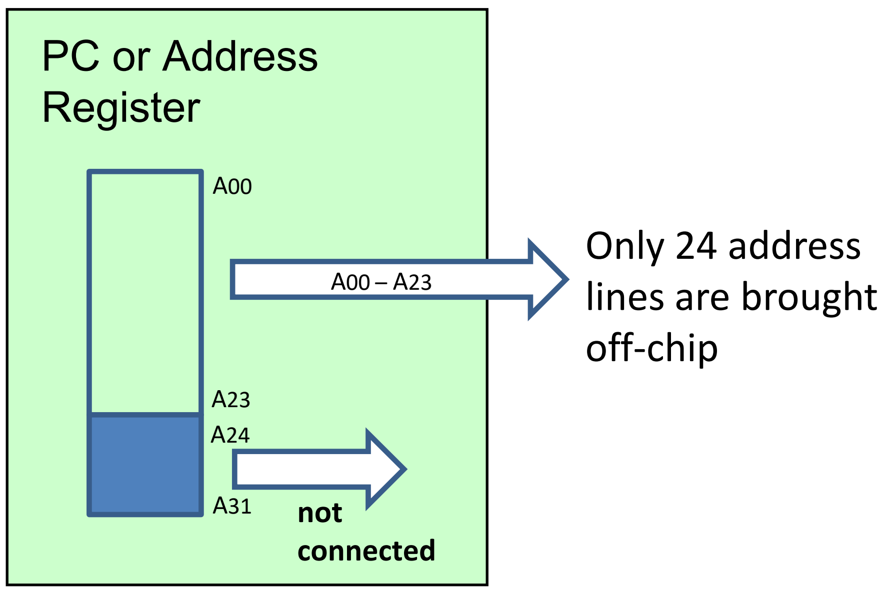
```
move.b #$ff,d0
add.b  #1,d0
```

- What are the contents of the condition-code register after the following program segment executes?

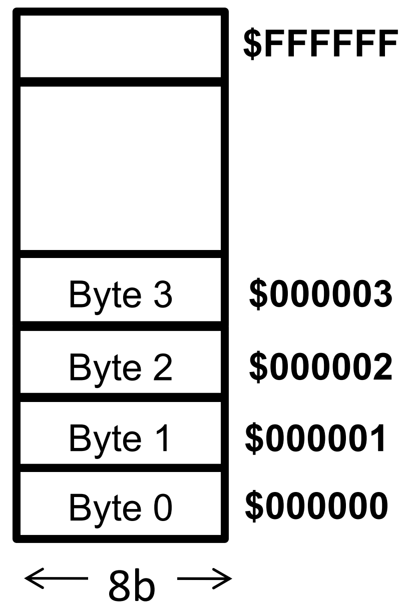
```
move.b #3,d0
sub.b  #4,d0
```

Memory

- The M68000 has a 24-bit address bus which limits the size of the addressable memory to 16 MB



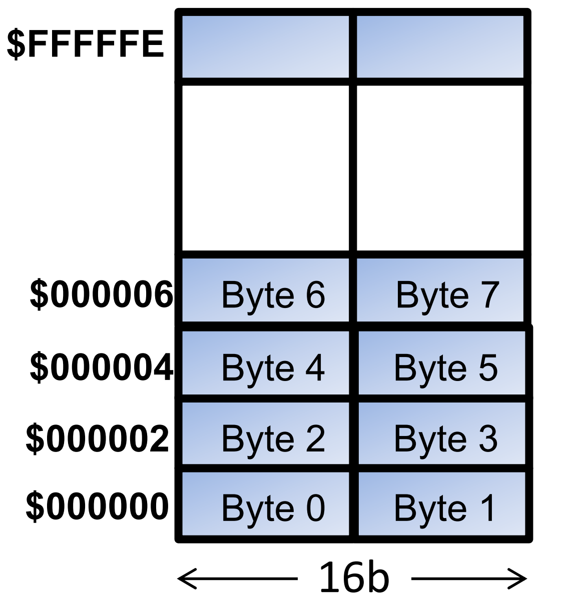
Processor



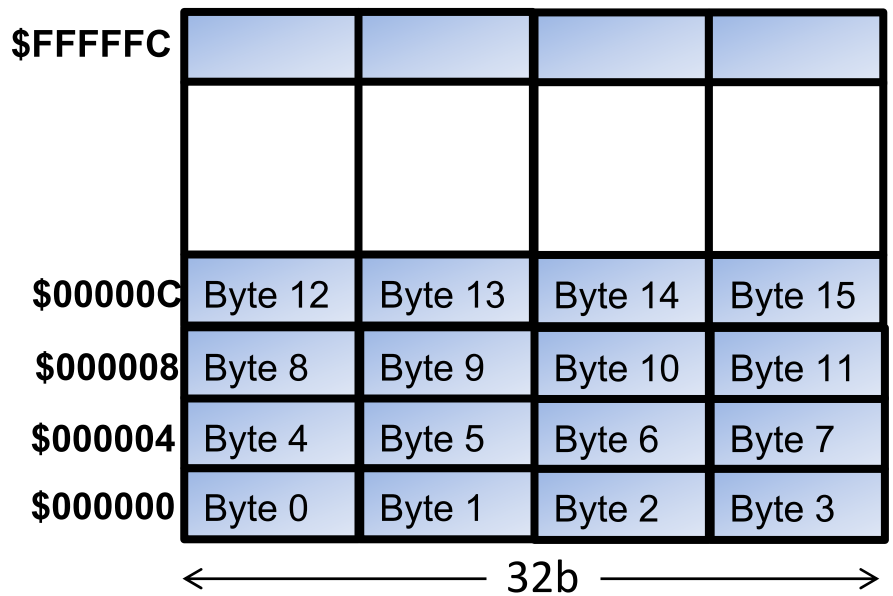
Byte View of Memory

Word and Long-Word Views of Memory

- The M68000 also supports word (2-byte) and long-word (4-byte) operations on memory



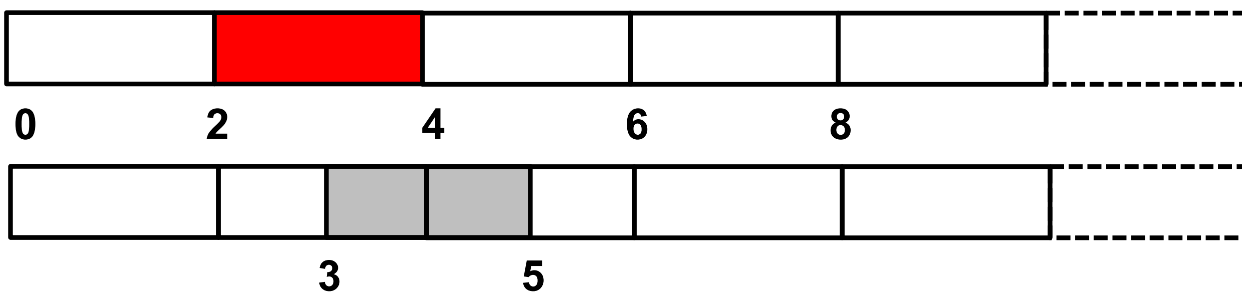
Word View of Memory



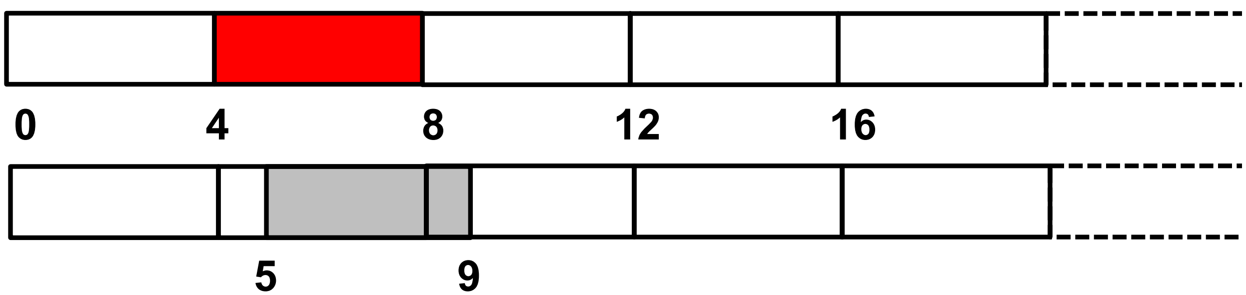
Long-word View of Memory

Alignment Issues

- Word and long-word operations must be at an even address
  - Consider the following word (2-byte) access



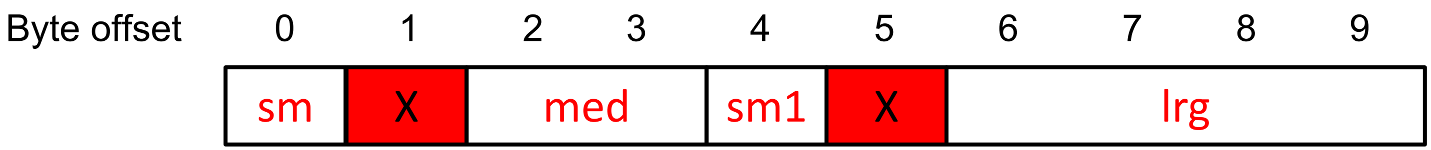
  - Consider the following long-word (4-byte) access



C Example

What is the size of this structure?

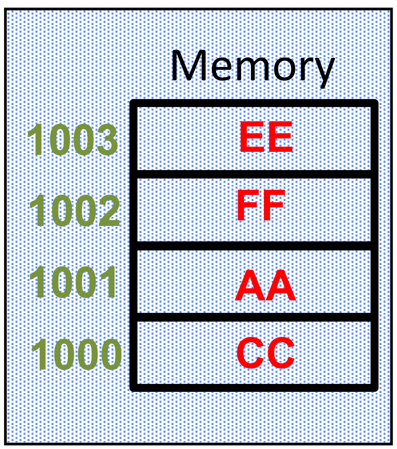
```
struct foo {
    char   sm;      /* 1 byte  */
    short  med;     /* 2 bytes */
    char   sm1;     /* 1 byte  */
    int    lrg;     /* 4 bytes */
}
```



- Historically
  - Early machines (IBM 360, circa 1964) required alignment
  - Removed in the 1970s to reduce impact on programmers (IBM 370, x86)
  - Reintroduced by RISC to improve performance
  - Removed by some RISCs to simplify software

Endianness: Big or Little?

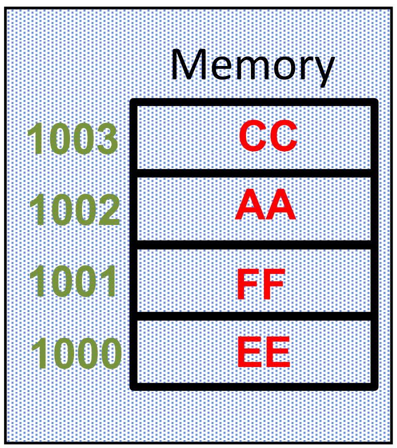
- Big Endian
  - Address of most-significant byte = address of word/long-word
  - IBM 360, M68000, MIPS, SPARC



0xCCAAFFEE → memory[1000]

Endianness: Big or Little?

- Big-Endian and Little-Endian are terms that refer to the order in which a sequence of bytes are stored in memory

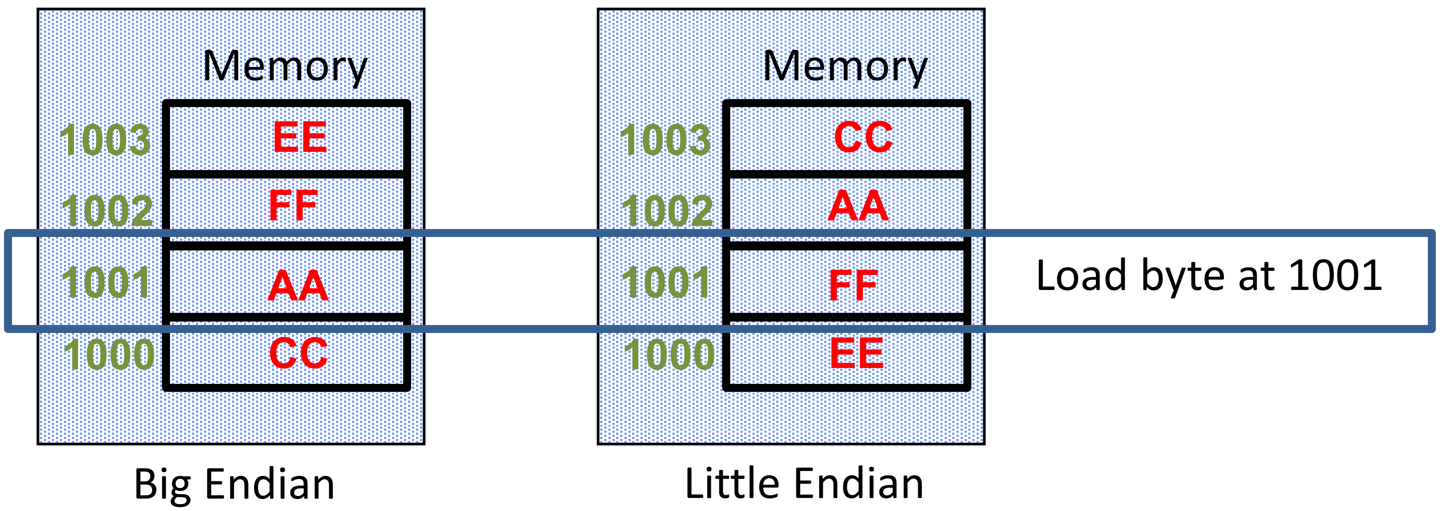


CCAAFFEE → memory[1000]

- Little Endian
  - Address of least-significant byte = address of word/long-word
  - Intel x86, ARM, DEC Vax & Alpha

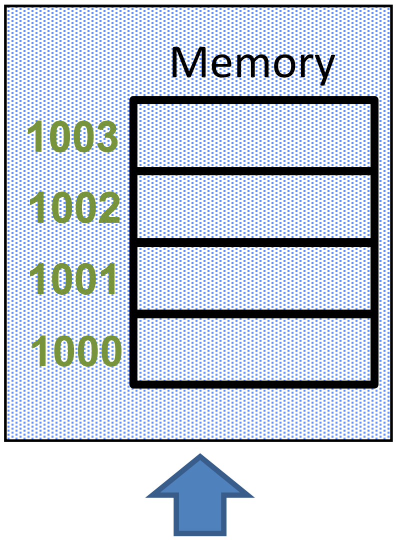
When Endianness Matters

- When you
  - Store word/long-words then load bytes
  - Communicate between different systems



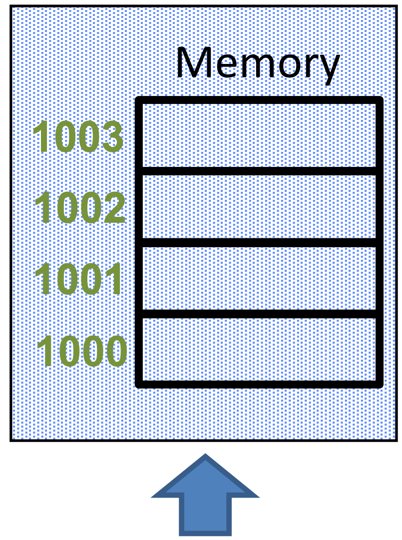
Most processors are bi-endian (configuration register)

Examples: Storing Data in Memory



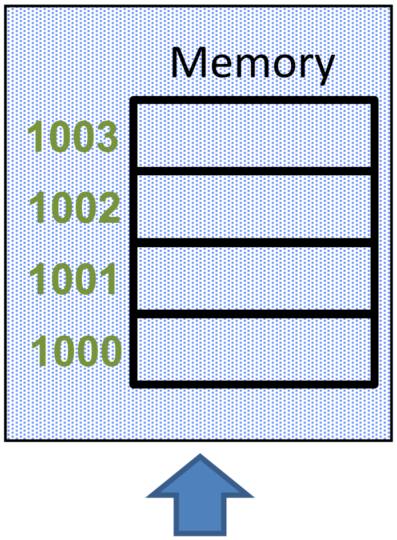
`move.b d0,$1000`

D0 | 12345678



`move.w d1,$1000`

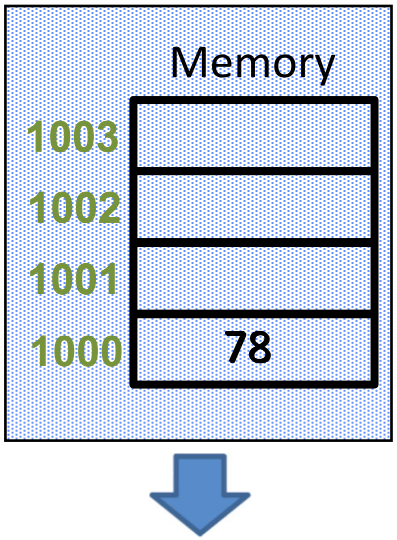
D1 | 1E3CBA0A



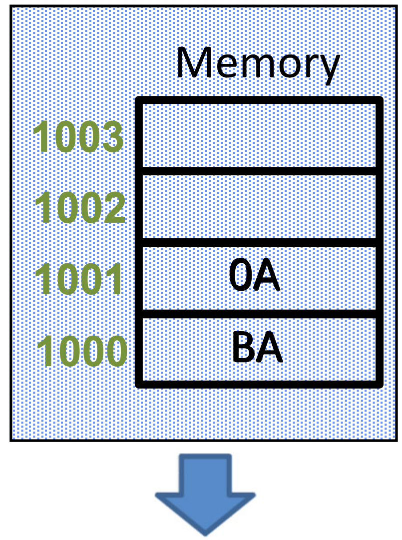
`move.l d2,$1000`

D2 | 3D3CA127

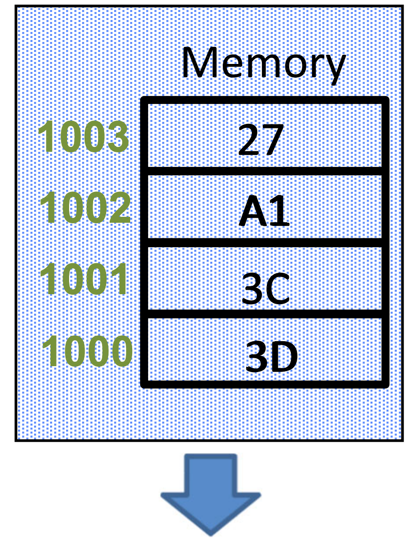
Examples: Loading Data from Memory



`move.b $1000,d0`



`move.w $1000,d1`

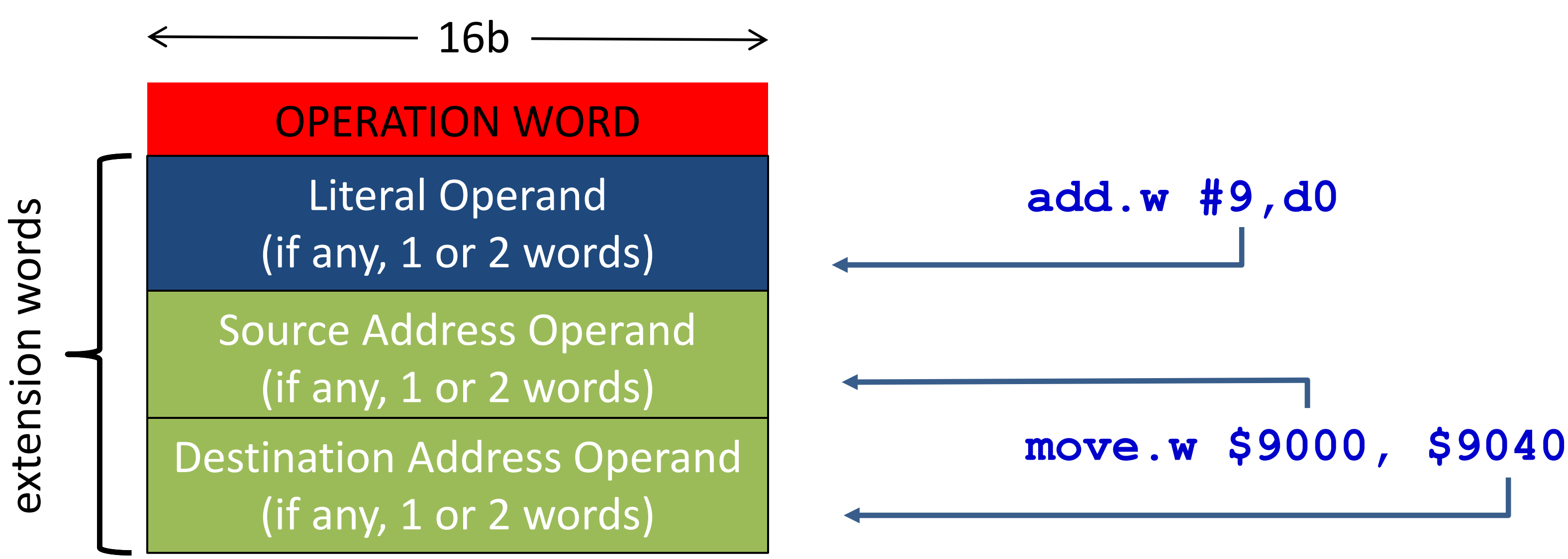


`move.l $1000,d2`

D0	123456
D1	1E3C
D2	

Instruction Formats

- A M68000 instruction consists of one to five words (16 bits)
  - An instruction may have 0, 1, or 2 operands
  - Operands may be bytes, words, or long-words
  - An instruction may access memory 0, 1, or 2 times
  - Destination operands are changed by the execution of an instruction
  - Source operands are not changed by the execution of an instruction



In-Class Examples

- Hand assemble the following M68000 instructions

```
move.b d0,d1
move.w $4,d1
move.l #4,$123456
```

Summary

- Data registers hold (integer) data apart from memory
  - D0-D7, homogeneous, 32-bits
  - Allow byte (.B), word (.W), and longword (.L) operations
- Arithmetic instructions
  - Not all operations are commutative (so order of operands matters)
  - Signed versus unsigned instructions
  - Many have specific data sizes (sign extension may be required)
- Condition-Code Register
  - Updated after most instructions execute (so check datasheet)
  - Carry acts as a borrow in case of subtraction
- Memory
  - Addresses are 32-bit internally, 24-bit externally
  - Total size of memory is 16 Megabytes
  - Allows byte (.B), word (.W), and longword (.L) operations
    - Word and Longword must be aligned on even address boundary
  - Byte ordering is Big-Endian

Summary

- Instructions consists of one to five words (16 bits)
  - An instruction may have 0, 1, or 2 operands (check datasheet)
  - Operands may be bytes, words, or long-words (check datasheet)
  - An instruction may access memory 0, 1, or 2 times (check datasheet)
  - Destination operands are changed by the execution of an instruction
  - Source operands are not changed by the execution of an instruction