Memory Locations, Address, Instructions and Instruction Sequencing

Read pages 28-40

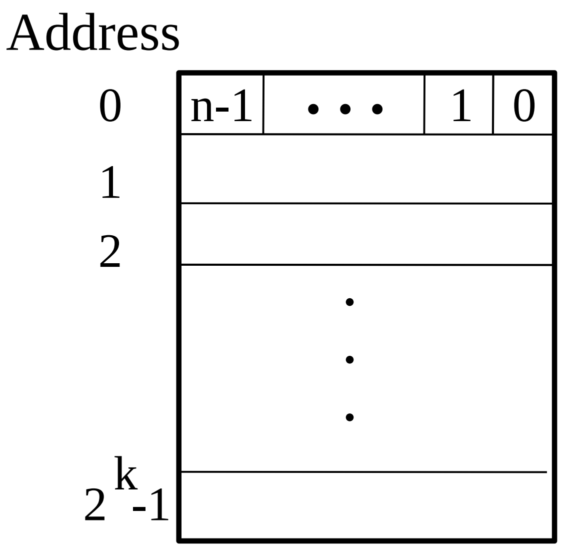
Memory locations and addresses

- The simple computer is a good start to understand computer organizations
- We need to study
  - how data/instructions are organized in the main memory?
  - how is memory addressed? – addressing mode

Memory

- Holds both instructions and data
- With *k* address bits and *n* bits per location

k	Number of locations
10	2^{10} = 1024 = 1K
16	2^{16} = 65,536 = 64K
20	2^{20} = 1,048,576 = 1M
24	2^{24} = 16,777,216 = 16M



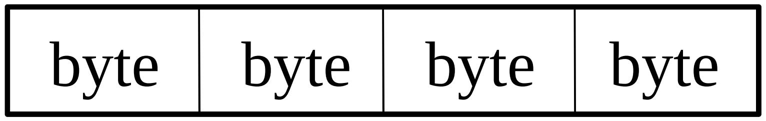
- n is typically 8 (byte), 16 (word), 32 (long word), ….

Memory stores both data and instructions

- Consider 32-bit long word in each location which can store
  - 32-bit 2's complement number (integer):

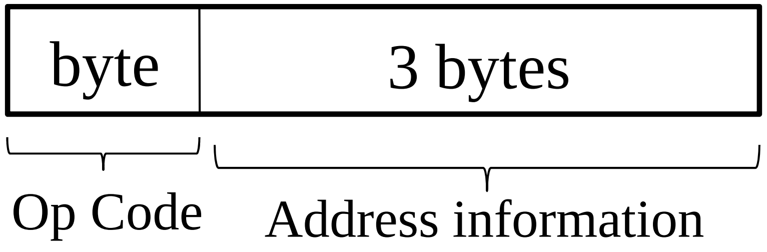
$$(-2^{n-1}) – (2^{n-1} – 1)$$

    - If n = 32: - 2G – 2G-1 (recall that G = 2^{30})
  - 4 ASCII characters



- It is often convenient to address operands which are as short as 1 byte

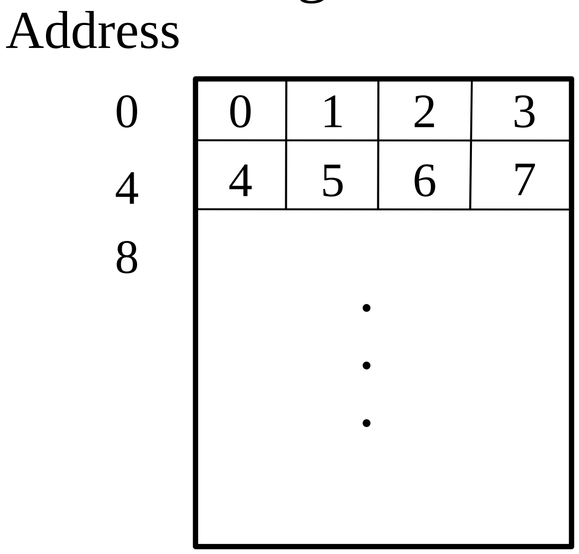
  - A machine instruction



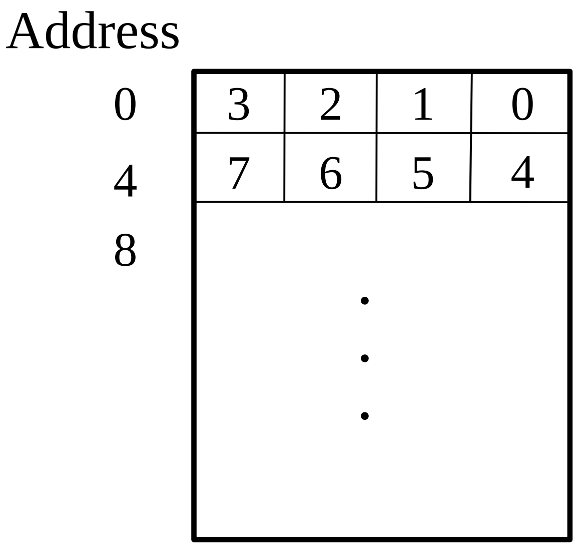
Dealing with strings of characters

- Byte addressable machine is almost universal
  - Successive addresses refer to successive byte locations
  - There are two different schemes for addressing byte:

big-endian



little – endian



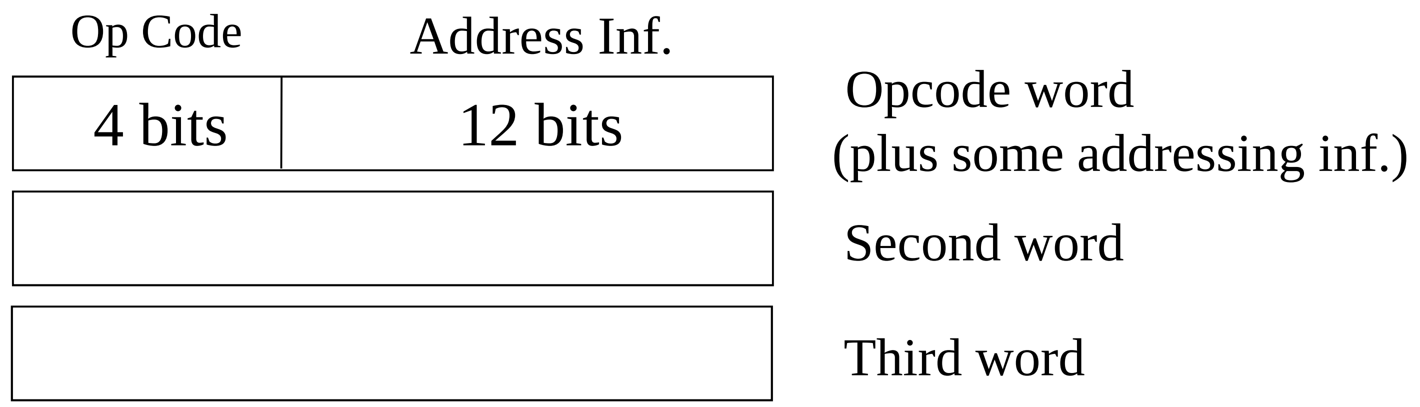
- Also bit can be numbered the other way around: bit 0 is the MSB

Memory addressing Example – for a Computer

- Word = 16 bits
- Byte addressable – uses big-endian
- Long word = 4 bytes
- 24 bits used for address $\longrightarrow$ 16 M bytes or 8 M words

Instructions and instruction sequencing

- Example computer instruction format:
  - Uses multiple words of 16 bits



  - Typical instruction is Add: C = A+B
  - Most general instruction is to add 2 numbers in memory and store in a 3rd location

    Add A, B, C $\quad$ $[A]+[B] \rightarrow C$

Problems for instructions with multiple memory locations

1. Long instructions
   - Address of an operand = 24 bits
   - Instruction length = 3 x 24 bits + opcode (4 bits) = 76 bits – too much memory space
   - Solutions: a) Use one- or two-address instruction:

     Add A, B: [A]+[B] $\rightarrow$ B

     Add A: [A]+[AC] $\rightarrow$ AC

     b) Use general-purpose CPU register

     Often 8-64 bits of them

     - 8 registers $\rightarrow$ use only 3 bits to select a CPU reg.

Problems for instructions with multiple memory locations (continued)

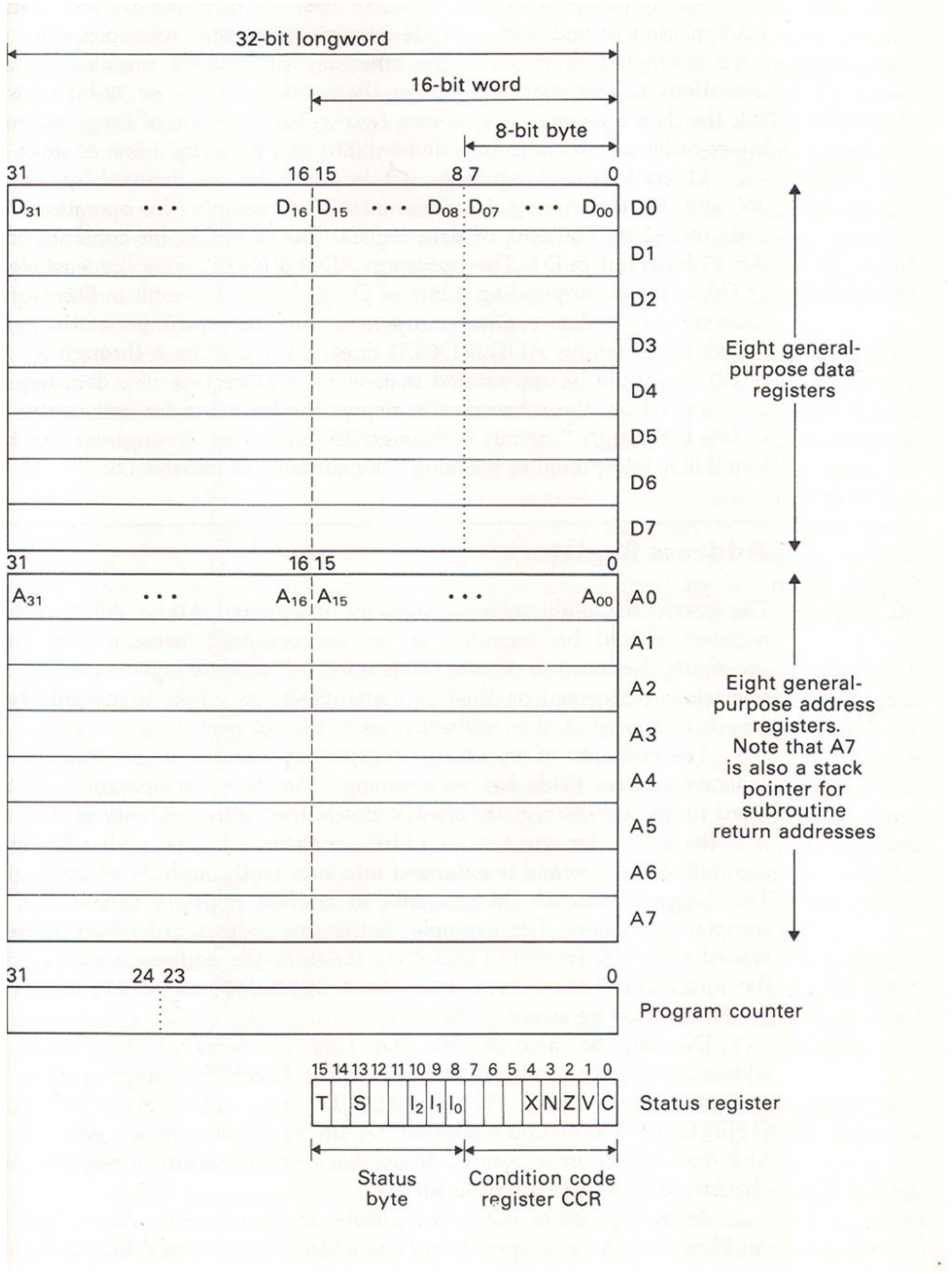
2. Memory access time is too long
   - Recall that memory access is always slow
   - Multiple memory access consumes a lot of time
   - Solutions: use CPU registers to store operands and temporary results
   - Minimizing the frequency with which data moved back and forth between main memory

Modern CPUs are designed with the above two issues in mind

e.g. ADD A, D1 $[A]+[D1] \rightarrow D1$

Registers on CPU

All are visible to programmer.



32-bit longword
16-bit word
8-bit byte
31
16 15
8 7
0
D_{31} • • • D_{16} D_{15} • • • D_{08} D_{07} • • • D_{00}
D0
D1
D2
D3
D4
D5
D6
D7
Eight general-purpose data registers
31
16 15
0
A_{31} • • • A_{16} A_{15} • • • A_{00}
A0
A1
A2
A3
A4
A5
A6
A7
Eight general-purpose address registers. Note that A7 is also a stack pointer for subroutine return addresses
31
24 23
0
Program counter
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
T S I_2 I_1 I_0 X N Z V C
Status register
Status byte
Condition code register CCR

Compare with our simple computer

1. D7 – D0 are equivalent to AC (accumulator)
2. A7 - A0 are equivalent to ? – none in our simple computer
3. PC (program counter) is the same as PC
4. SR (status register) – none in our simple computer
5. Where are X, MDR, MAR, and IR?
6. Those are not visible to the programmer
7. Guess how many invisible registers in the CPU?

Some instructions for illustration

- ADD -- 2 operand instruction
  - e.g. ADD B, D0 $[B]+[D0] \rightarrow D0$

    Memory location, data register 0

  - Either source or destination must be one of the 8 data registers
  - Both can be data register
- MOVE – similar to ADD
  - e.g. MOVE A, D0 $[A] \rightarrow D0$
  - To do C = A+B

```
MOVE.L   A, D0
ADD.L    B, D0
MOVE.L   D0, C
```

Some instructions for illustration (continued)

- SUB (subtract)
  - e.g. SUB B, D0 [D0] - [B] → D0
- CMP (compare)
  - e.g. CMP B, D0 [D0] - [B] and set/reset N, Z, V, C
- MOVEA (move address)
  - e.g. MOVEA ADDR, A3 [ADDR] → A3
- CLR A 0 → A (clear data register or memory location)
  - Can use .B, .W, or .L
- TST A [A] – 0 and set/reset N, Z; make V, C=0
  - Can use .B, .W, or .L
- ADDQ #2, D5 [D5] + 2 → D5 quick addition
- SUBQ #1, D4 [D4] - 1 → D4 quick subtraction