MACHINE INSTRUCTIONS AND PROGRAMS

Objectives

- Machine instructions and program execution, including branching and subroutine call and return operations.
- Addressing methods for accessing register and memory operands.
- Assembly language for representing machine instructions, data, and programs.
- Program-controlled Input/Output operations.

MEMORY LOCATIONS, ADDRESSES, AND OPERATIONS

Memory Location, Addresses, and Operation

- Memory consists of many millions of storage cells, each of which can store 1 bit.
- Data is usually accessed in *n*-bit groups. *n* is called word length.

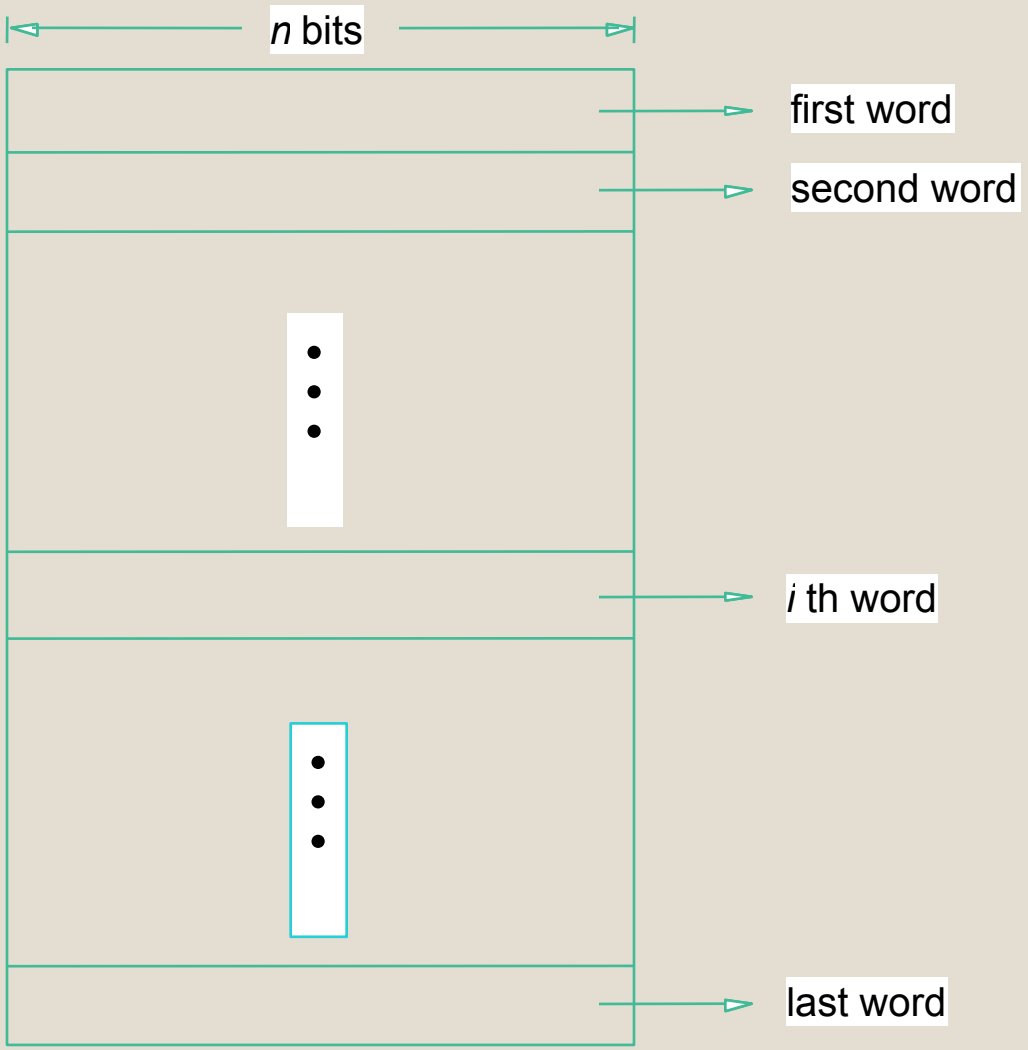
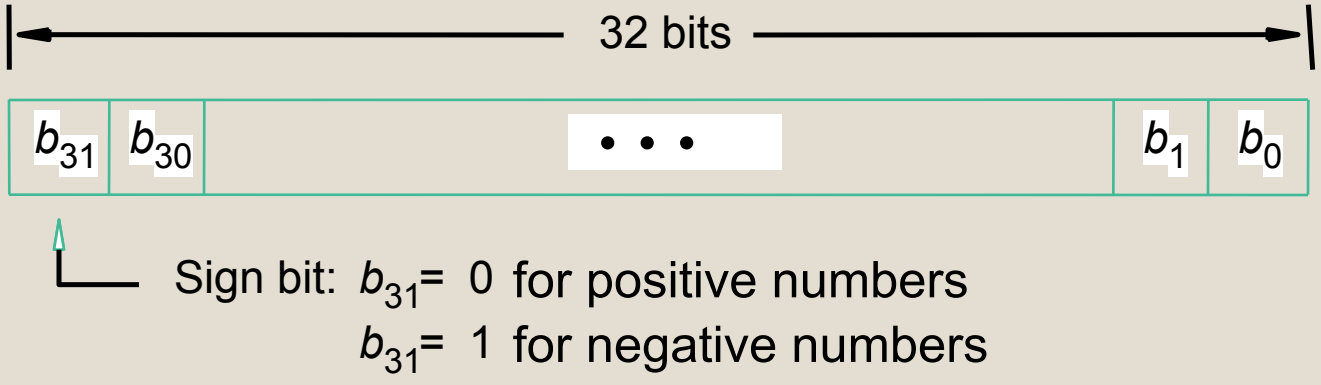


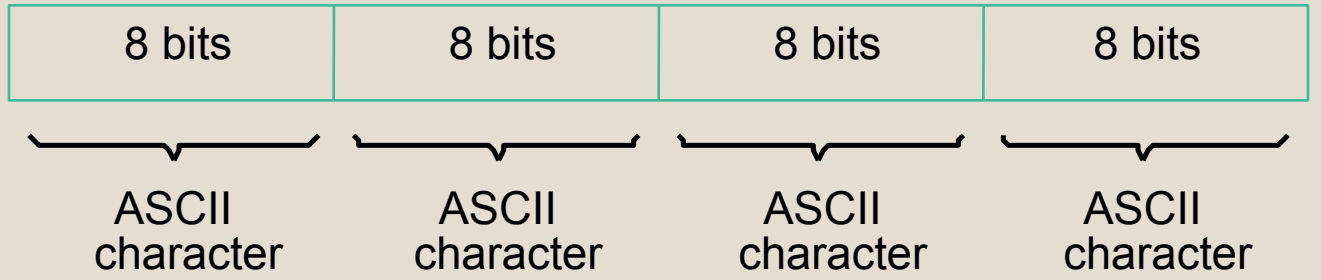
Fig: Memory words.

Memory Location, Addresses, and Operation

- 32-bit word length example



(a) A signed integer



(b) Four characters

Memory Location, Addresses, and Operation

- To retrieve information from memory, either for one word or one byte (8-bit), addresses for each location are needed.
- A *k*-bit address memory has 2^k memory locations, namely 0 – 2^k-1, called memory space.
- 24-bit memory: 2^{24} = 16,777,216 = 16M (1M=2^{20})
- 32-bit memory: 2^{32} = 4G (1G=2^{30})
- 1K(kilo)=2^{10}
- 1T(tera)=2^{40}

Memory Location, Addresses, and Operation

- It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory – byte-addressable memory.
- Byte locations have addresses 0, 1, 2, … If word length is 32 bits, they successive words are located at addresses 0, 4, 8,…

Big-Endian and Little-Endian Assignments

Big-Endian: lower byte addresses are used for the most significant bytes of the word

Little-Endian: opposite ordering. lower byte addresses are used for the less significant bytes of the word

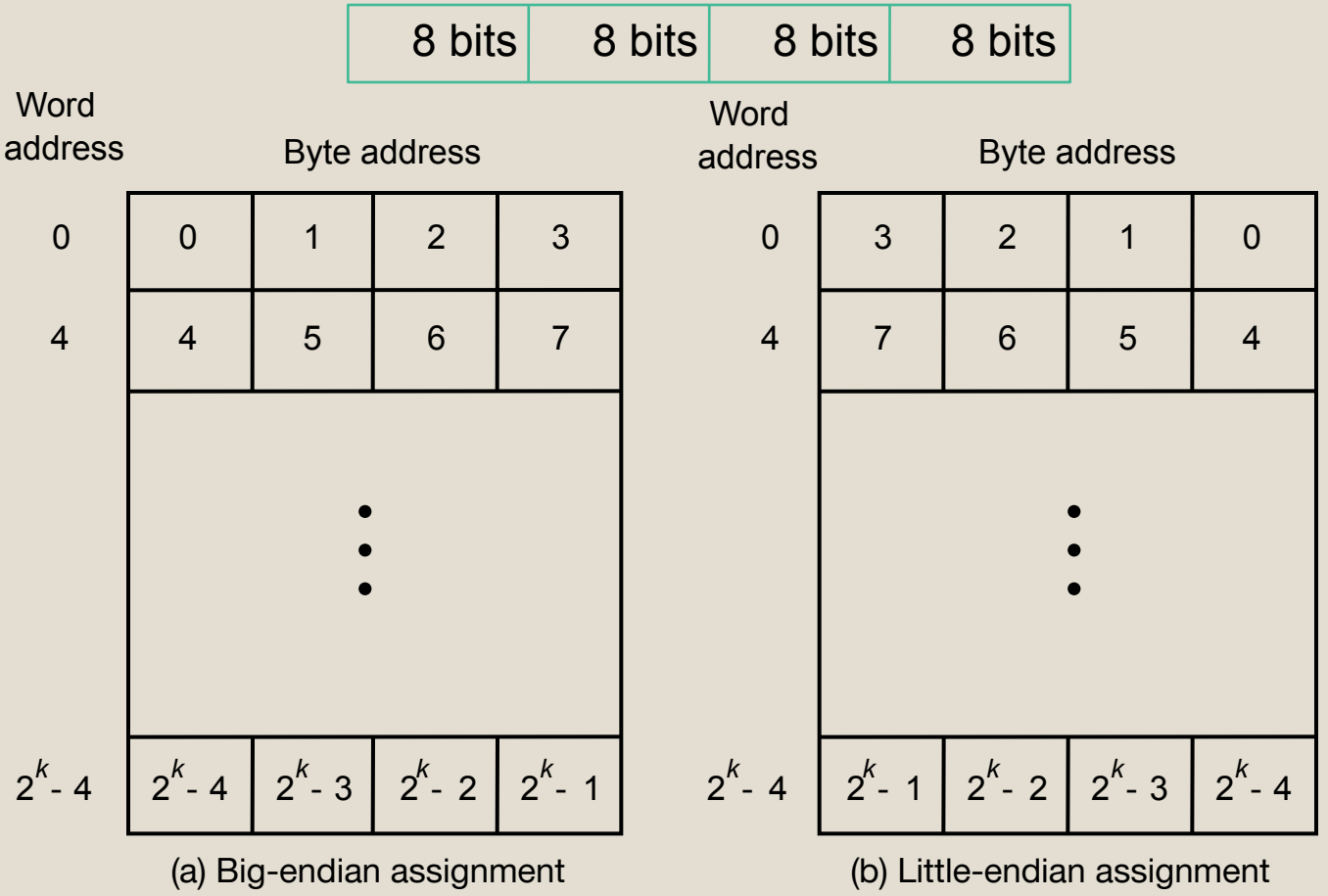


Figure 2.7. Byte and word addressing.

Memory Location, Addresses, and Operation

- Address ordering of bytes
- Word alignment
  - Words are said to be aligned in memory if they begin at a byte addr. that is a multiple of the num of bytes in a word.
    - 16-bit word: word addresses: 0, 2, 4,….
    - 32-bit word: word addresses: 0, 4, 8,….
    - 64-bit word: word addresses: 0, 8,16,….
- Access numbers, characters, and character strings

Memory Operation

- Load (or Read or Fetch)
  - Copy the content. The memory content doesn't change.
  - Address – Load
  - Registers can be used
- Store (or Write)
  - Overwrite the content in memory
  - Address and Data – Store
  - Registers can be used

Instruction and Instruction Sequencing

“Must-Perform” Operations

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

Register Transfer Notation

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location (R1←[LOC], R3 ←[R1]+[R2])
- Register Transfer Notation (RTN)

Assembly Language Notation

- Represent machine instructions and programs.
- Move LOC, R1 => R1←[LOC]
- Add R1, R2, R3 => R3 ←[R1]+[R2]

Assembly Language Notation

- Represent machine instructions and programs.
- Move LOC, R1 => R1←[LOC]
- Add R1, R2, R3 => R3 ←[R1]+[R2]

CPU Organization

- Single Accumulator
  - Result usually goes to the Accumulator
  - Accumulator has to be saved to memory quite often
- General Register
  - Registers hold operands thus reduce memory traffic
  - Register bookkeeping
- Stack
  - Operands and result are always in the stack

Instruction Formats

- Three-Address Instructions
  - ADD R2, R3, R1 R1 ← [R2] + [R]3
- Two-Address Instructions
  - ADD R2, R1 R1 ← [R1] + [R2]
- One-Address Instructions
  - ADD M AC ← [AC] + M[AR]
- Zero-Address Instructions
  - ADD TOS ← [TOS] + [TOS – 1]
- RISC Instructions
  - Lots of registers. Memory is restricted to Load & Store

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- Three-Address
  1. ADD A, B, R1 ; R1 ← M[A] + M[B]
  2. ADD C, D, R2 ; R2 ← M[C] + M[D]
  3. MUL R1, R2, X ; M[X] ← [R1] * [R2]

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- Two-Address

```
1.  MOV   A, R1    ; R1 ← M[A]
2.  ADD   B, R1    ; R1 ← [R1] + M[B]
3.  MOV   C, R2    ; R2 ← M[C]
4.  ADD   D, R2    ; R2 ← [R2] + M[D]
5.  MUL   R2, R1   ; R1 ← [R1] * [R2]
6.  MOV   R1, X    ; M[X] ← [R1]
```

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- One-Address

```
1.  LOAD   A    ; AC ← M[A]
2.  ADD    B    ; AC ← [AC] + M[B]
3.  STORE  T    ; M[T] ← [AC]
4.  LOAD   C    ; AC ← M[C]
5.  ADD    D    ; AC ← [AC] + M[D]
6.  MUL    T    ; AC ← [AC] * M[T]
7.  STORE X ; M[X] ← [AC]
```

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- Zero-Address

```
1.  PUSH   A    ; TOS ← [A]
2.  PUSH   B    ; TOS ← [B]
3.  ADD         ; TOS ← [A + B]
4.  PUSH   C    ; TOS ← [C]
5.  PUSH   D    ; TOS ← [D]
6.  ADD         ; TOS ← [C + D]
7.  MUL         ; TOS ← [C+D]*[A+B]
8.  POP    X    ; M[X] ← [TOS]
```

Instruction Formats

Example: Evaluate (A+B) * (C+D)

- RISC

```
1.  LOAD   A, R1        ; R1 ← M[A]
2.  LOAD   B, R2        ; R2 ← M[B]
3.  LOAD   C, R3        ; R3 ← M[C]
4.  LOAD   D, R4        ; R4 ← M[D]
5.  ADD    R1, R2, R1   ; R1 ← [R1] + [R2]
6.  ADD    R3, R4, R3   ; R3 ← [R3] + [R4]
7.  MUL    R1, R3, R1   ; R1 ← [R1] * [R3]
8.  STORE  X, R1        ; M[X] ← [R1]
```