Connection Between the Processor and the Memory

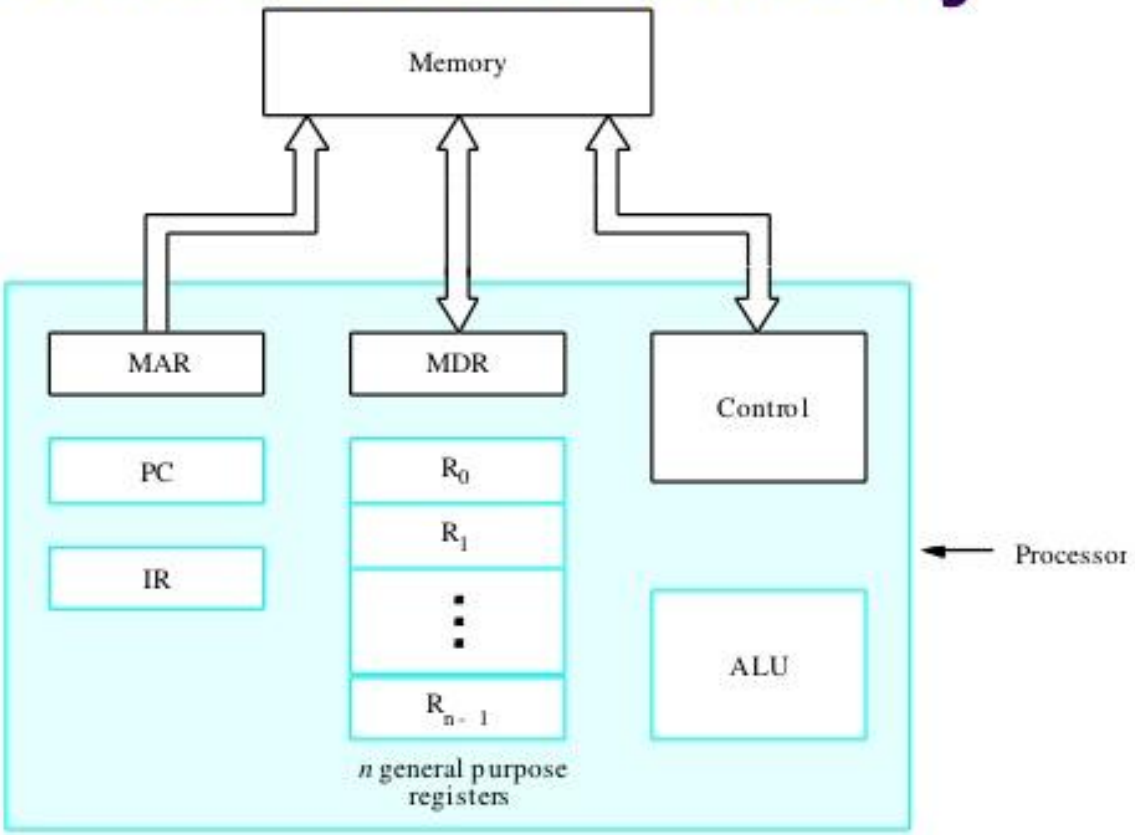

Figure 1.2. Connections between the processor and the memory.

Registers

- Instruction register (IR)
- Program counter (PC)
- General-purpose register (R_0 – R_{n-1})
- Memory address register (MAR)
- Memory data register (MDR)

A Typical Instruction

Add R0, LOCA

- Add the operand at memory location LOCA to the operand in a register R0 in the processor.
- Place the sum into register R0.
- The original contents of LOCA are preserved.
- The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.

Typical Operating Steps

- Programs reside in the memory through input devices
- PC is set to point to the first instruction
- The contents of PC are transferred to MAR
- A Read signal is sent to the memory
- The first instruction is read out and loaded into MDR
- The contents of MDR are transferred to IR
- Decode and execute the instruction

Typical Operating Steps

- Get operands for ALU
  - General-purpose register
  - Memory (address to MAR – Read – MDR to ALU)
- Perform operation in ALU
- Store the result back
  - To general-purpose register
  - To memory (address to MAR, result to MDR – Write)
- During the execution, PC is incremented to the next instruction

Memory Location and Addresses

- 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.

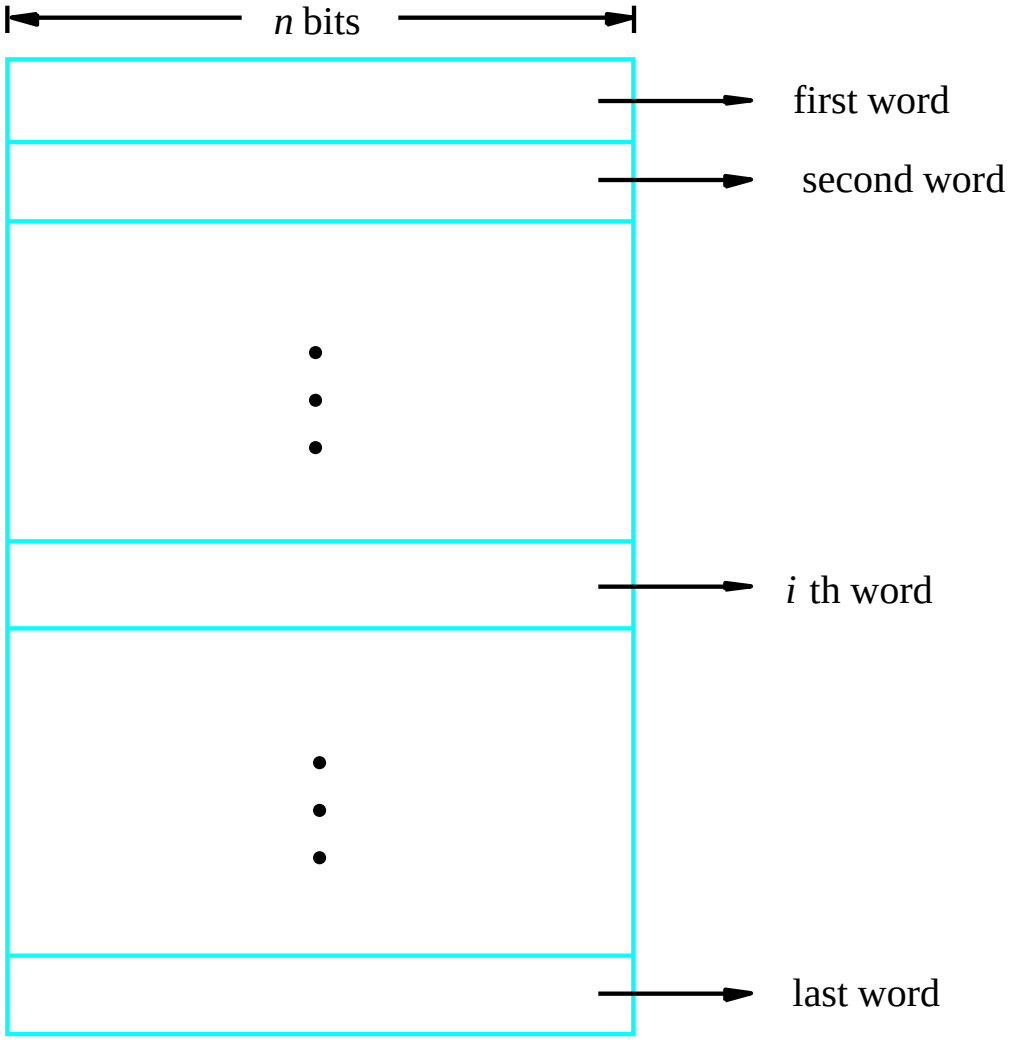

Figure 2.5. Memory words.

Memory Location and Addresses

- 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 and Addresses

- 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,…

Memory Location and Addresses

- 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,….

Instruction Formats

- Three-Address Instructions
  - ADD R1, R2, R3 R1 ← R2 + R3
- Two-Address Instructions
  - ADD R1, R2 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 R1, A, B ; R1 ← M[A] + M[B]
  2. ADD R2, C, D ; R2 ← M[C] + M[D]
  3. MUL X, R1, R2 ; M[X] ← R1 ∗ R2

Instruction Formats

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

- Two-Address
  1. MOV R1, A ; R1 ← M[A]
  2. ADD R1, B ; R1 ← R1 + M[B]
  3. MOV R2, C ; R2 ← M[C]
  4. ADD R2, D ; R2 ← R2 + M[D]
  5. MUL R1, R2 ; R1 ← R1 * R2
  6. MOV X, R1 ; 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. PUSHA ; TOS ← A
  2. PUSHB ; TOS ← B
  3. ADD ; TOS ← (A + B)
  4. PUSH C ; TOS ← C
  5. PUSHD ; 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 R1, A ; R1 ← M[A]
  2. LOAD R2, B ; R2 ← M[B]
  3. LOAD R3, C ; R3 ← M[C]
  4. LOAD R4, D ; R4 ← M[D]
  5. ADD R1, R1, R2 ; R1 ← R1 + R2
  6. ADD R3, R3, R4 ; R3 ← R3 + R4
  7. MUL R1, R1, R3 ; R1 ← R1 * R3
  8. STORE X, R1 ; M[X] ← R1