MEGHE GROUP
VALUES REDEFINED

DTEL (Department for Technology Enhanced Learning)

The Centre for Technology enabled Teaching & Learning , N Y S S, India

Teaching Innovation - Entrepreneurial - Global

MEGHE GROUP
VALUES REDEFINED

DEPARTMENT OF COMPUTER TECHNOLOGY

IV-SEMESTER

COMPUTER ARCHITECTURE AND ORGANIZATION(CT 2201)

Unit I

Basic Structure of Computer Hardware and Software

UNIT I:- SYLLABUS

1. Functional Units
2. Basic Operational Concepts
3. Bus Structures
4. Addressing Methods
5. Memory Locations, addressing
5. and encoding of information
6. Instruction and Instruction sequencing

The student will be able to:

1. To Understand Internal working of Computer System, its basic principles & execution of machine instructions.
2. To Understand basic processor design using Hardwired and microprogrammed control unit.
3. To Know Organization of main memory, cache memory.
4. To Know Various ways in which I/O operations are performed.

Functional Units

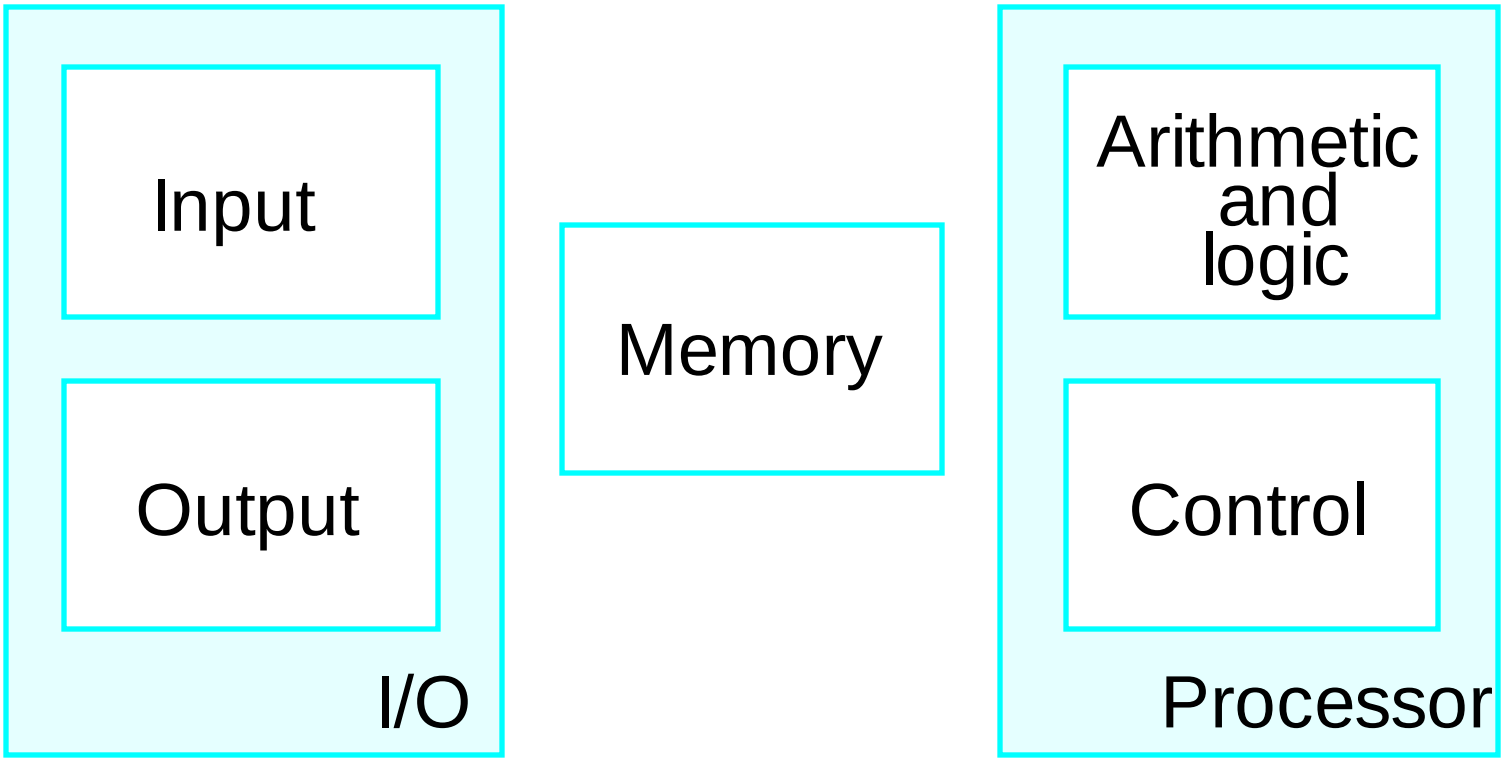


Figure 1.1. Basic functional units of a computer.

Functional Units

Information Handled by a Computer

- Instructions/machine instructions
  - Govern the transfer of information within a computer as well as between the computer and its I/O devices
  - Specify the arithmetic and logic operations to be performed
  - Program
- Data
  - Used as operands by the instructions
  - Source program
- Encoded in binary code – 0 and 1

Functional Units

Memory Unit

- Memory Unit Stores programs and data
- Two classes of storage

➢ Primary storage

- Fast
- Programs must be stored in memory while they are being executed
- Large number of semiconductor storage cells
- Processed in words
- Address
- RAM and memory access time
- Memory hierarchy – cache, main memory

➢ Secondary storage – larger and cheaper

Functional Units

Arithmetic and Logic Unit (ALU)

- Most computer operations are executed in ALU of the processor.
- Load the operands into memory – bring them to the processor – perform operation in ALU – store the result back to memory or retain in the processor.
- Registers
- Fast control of ALU

Functional Units

Control Unit

- All computer operations are controlled by the control unit.
- The timing signals that govern the I/O transfers are also generated by the control unit.
- Control unit is usually distributed throughout the machine instead of standing alone.
- Operations of a computer:

➢ Accept information in the form of programs and data through an input unit and store it in the memory

➢ Fetch the information stored in the memory, under program control, into an ALU, where the information is processed

➢ Output the processed information through an output unit

➢ Control all activities inside the machine through a control unit

Basic Operational Concepts

Review

- Activity in a computer is governed by instructions.
- To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.

Basic Operational Concepts

A Typical Instruction

- Instruction:
- Opcode S, D
- Add LOCA, R0
- LOCA=10, R0=20
- 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.

Basic Operational Concepts

Separate Memory Access and ALU Operation

- Add LOCA, R0
- Sub R0,R1

-FETCH THE INST-1000

- Read/ Fetch the operand- LOCA=2000=>10
- Perform operation(Sub)

-Store the result

Basic Operational Concepts

Connection Between the Processor and the Memory

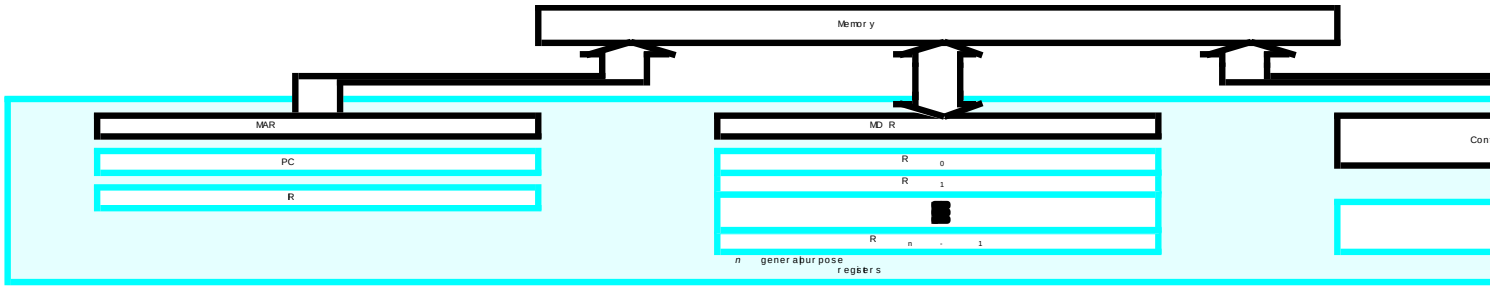


Figure 12. Connections between the processor and the memory.

Basic Operational Concepts

Registers

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

Basic Operational Concepts

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

Basic Operational Concepts

Typical Operating Steps (Cont.)

- 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

Basic Operational Concepts

Interrupt

- Normal execution of programs may be preempted if some device requires urgent servicing.
- The normal execution of the current program must be interrupted – the device raises an *interrupt* signal.
- Interrupt-service routine
- Current system information backup and restore (PC, general-purpose registers, control information, specific information)

THANK YOU

Bus Structures

- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a *bus*.
- Address/data/control

Bus Structures

- Single-bus

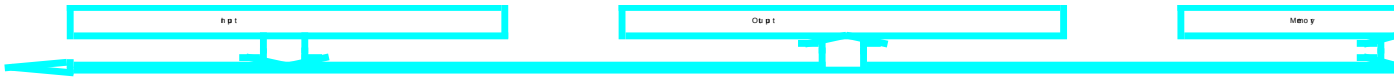


Figure 3. Single-bus structure.

Bus Structures

Speed Issue

- Different devices have different transfer/operate speed.
- If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.
- How to solve this?
- A common approach – use buffers.

THANK YOU

- 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.

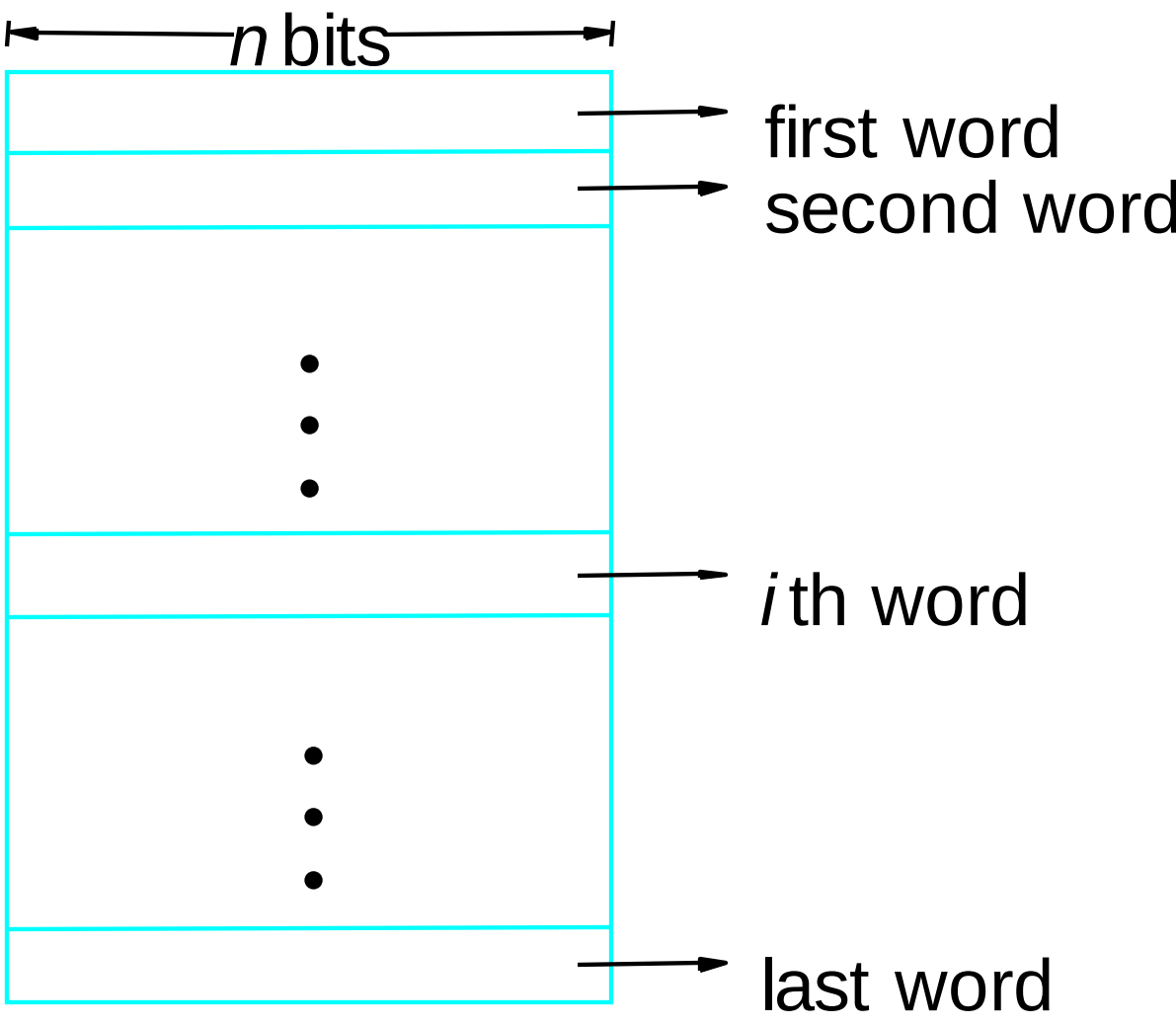
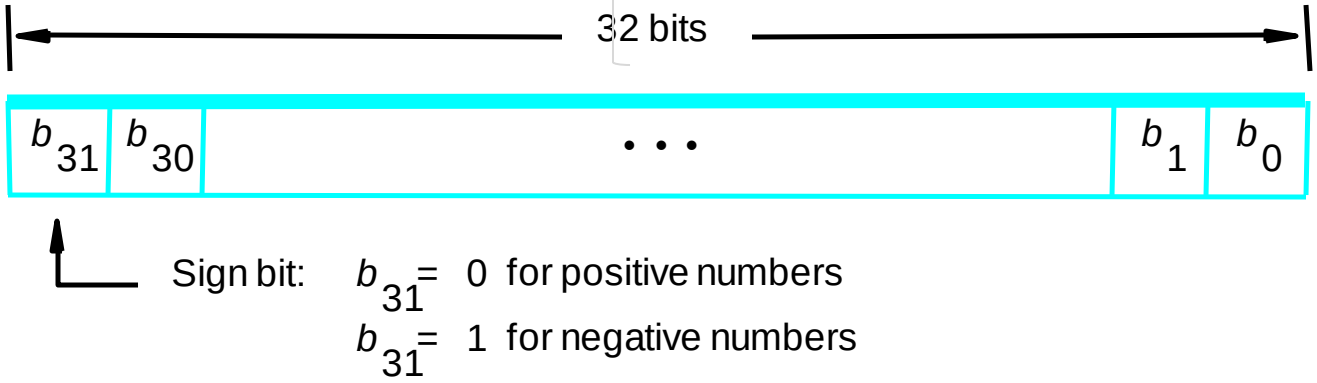
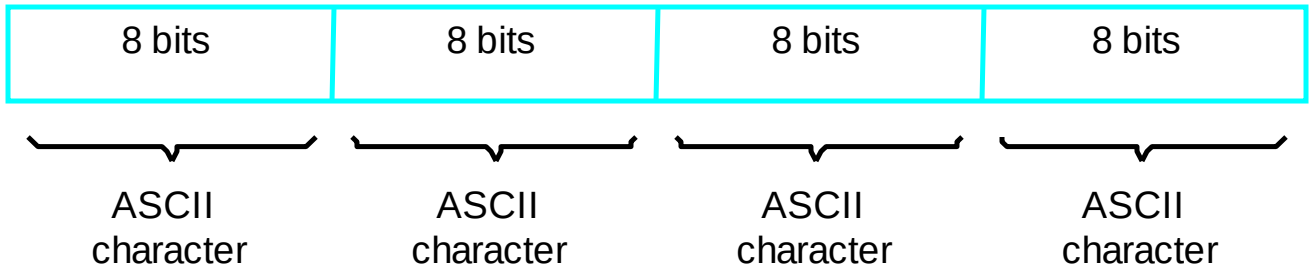


Figure 2.5. Memory words.

- 32-bit word length example



(a) A signed integer



(b) Four characters

Memory Locations, Addresses and Operations

- 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}

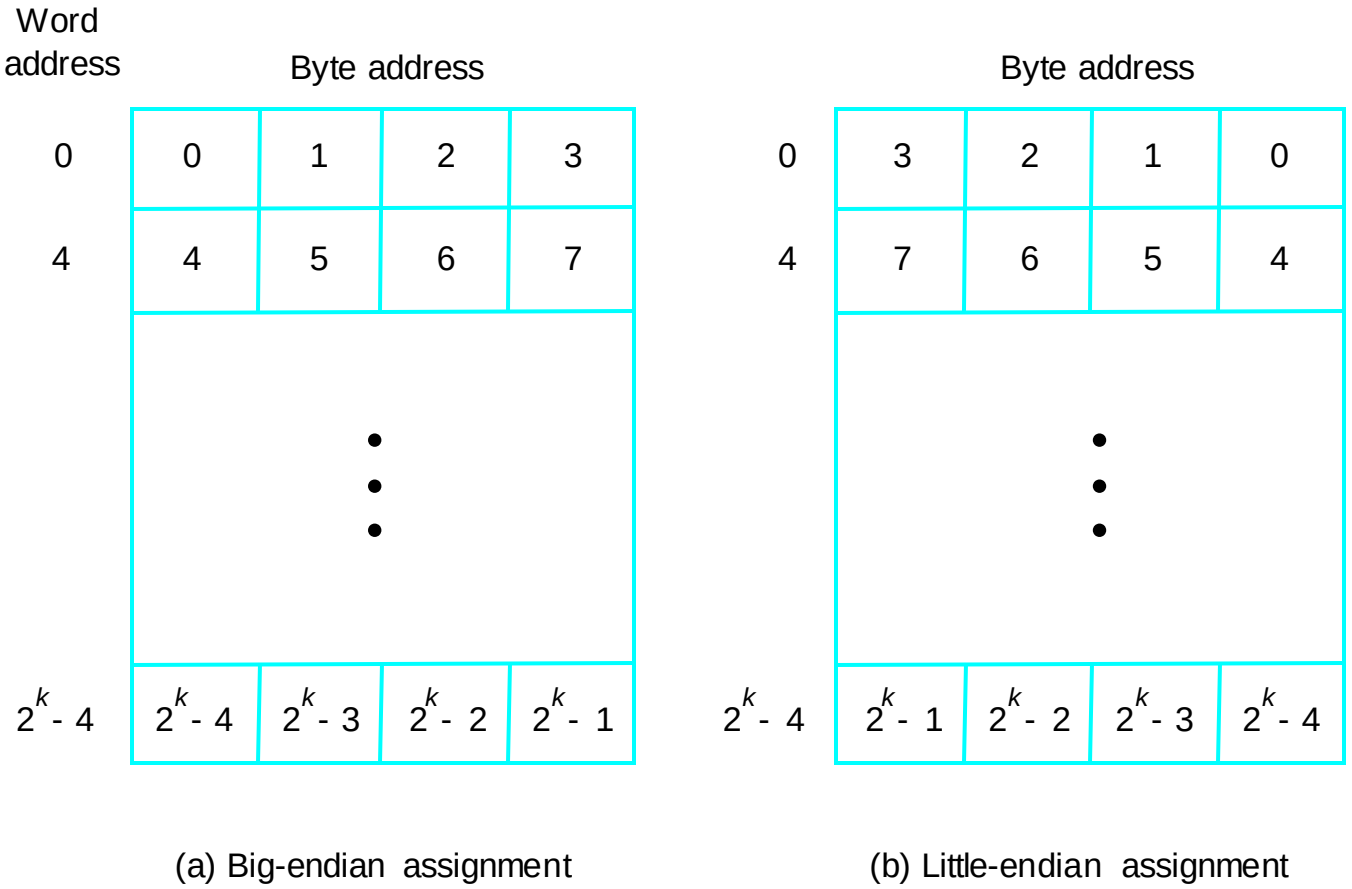
Memory Locations, Addresses and Operations

- 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



(a) Big-endian assignment (b) Little-endian assignment

Figure 2.7. Byte and word addressing.

Memory Locations, Addresses and Operations

- 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 Locations, Addresses and Operations

Consider a 16-bit internet that is made up of 2 bytes. Two ways to store this value −

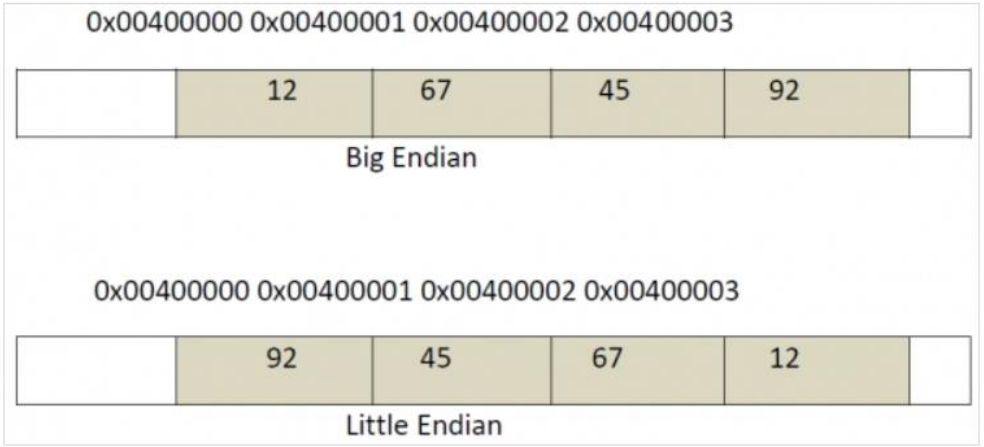
Little Endian − In this scheme, low-order byte is stored on the starting address (A) and high-order byte is stored on the next address (A + 1).

Big Endian − In this scheme, high-order byte is stored on the starting address (A) and low-order byte is stored on the next address (A + 1).

By these definitions, a 32-bit data pattern, which is regarded as a 32-bit unsigned integer. The "high-Order" byte is the one for the largest powers of 2: 2^{31}, ..., 2^{24}. The "low-order" byte is the one for the smallest powers of 2: 2^{7}, ..., 2^{0}.

Example

0x12674592 in 32-bit representation can be stored as −



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

THANK YOU

Addressing Methods

Instruction Formats

- Three-Address Instructions
  - ADD R1, R2, R3 — R3 ← [R1] + [R2]
- Two-Address Instructions
  - ADD R1, R2 — R2 ← [R1] + [R2]
- One-Address Instructions
  - ADD M — AC ← AC + M[AR]
  - ADD A — AC ← [AC]+ [A]
- Zero-Address Instructions
  - ADD — TOS ← TOS + (TOS – 1)

Addressing Methods

Instruction Formats

C=A+B

C←[A]+[B]

- Three-Address Instructions
  - ADD A,B,C C ← [A] + [B]
- Two-Address Instructions
  - ADD A,B B ← [A] + [B]

PBM OF OVERWITTEN

MOVE B,C C←[B]

ADD A,C C ← [A] + [C]

One-address:

LOAD A AC←[A]

ADD B AC←[B]+[AC]

STORE C C←[AC]

Addressing Methods

Instruction Formats

C=A+B

C←[A]+[B]

Postfix:AB+

Prefix:+AB

Infix:A+B

- ZERO-Address Instructions:

C=AB+

PUSH	A	TOS←[A]
PUSH	B	TOS←[B]
ADD		TOS←TOS+(TOS-1)
POP	C	C←TOS

Addressing Methods

Instruction Formats

EXAMPLE: Solve using three, two, one and zero address instruction

1. Z=(A+B)*C

THREE:

```
ADD     A,B,R1
MUL     R1,C,Z
```

ZERO:Z=AB+C*

```
PUSH    A
PUSH    B
ADD
PUSH    C
MUL
POP     Z
```

2. Z=(A+B-C)*(C-D+E)/(A*B-C)
3. Z=(A+B)*(C-D+E)/(A+D)

Instruction Formats (Cont.)

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 (Cont.)

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 (Cont.)

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 (Cont.)

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 Execution and Straight-Line Sequencing

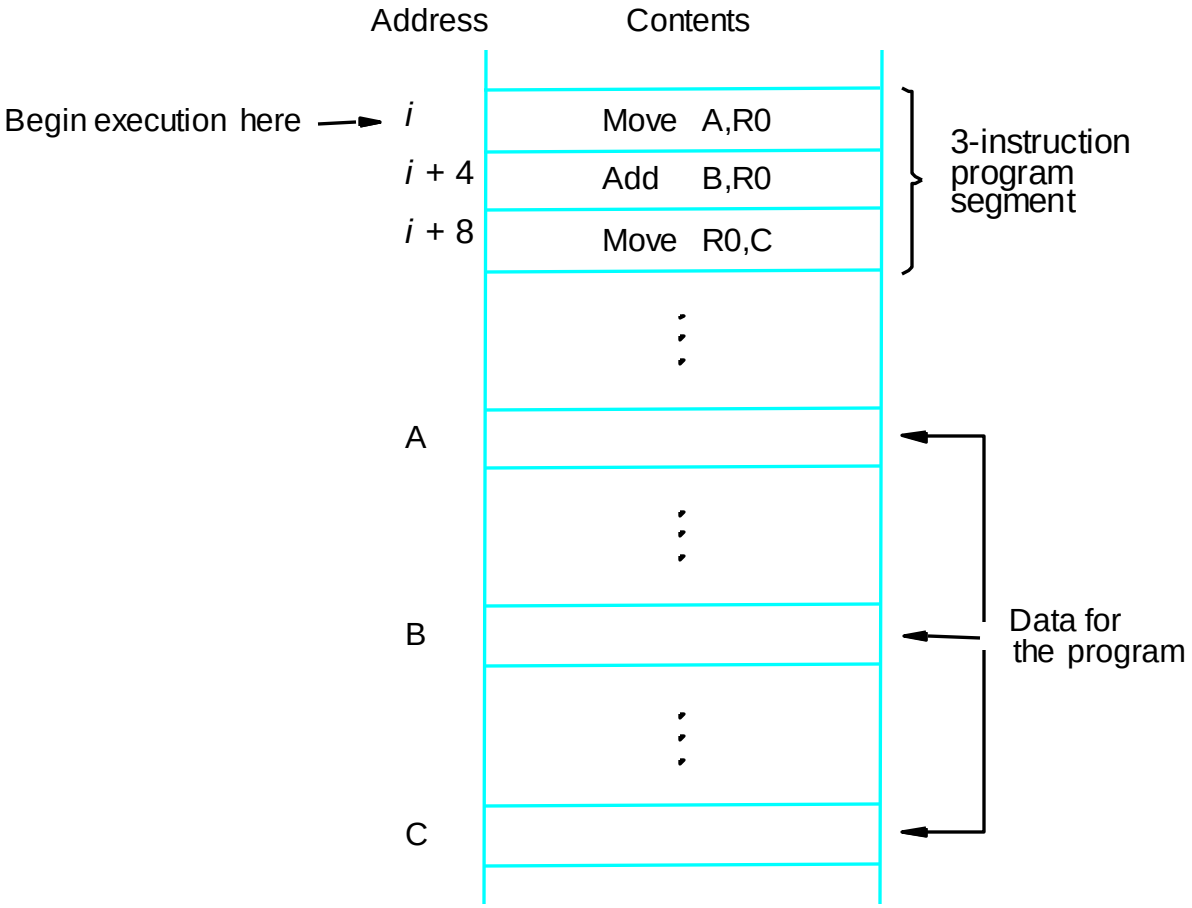


Figure 2.8. A program for C ← [A] + [B].

Assumptions:
- One memory operand per instruction
- 32-bit word length
- Memory is byte addressable
- Full memory address can be directly specified in a single-word instruction

Two-phase procedure
-Instruction fetch
-Instruction execute

Page 43

Instructions And Instruction Sequencing

Branching

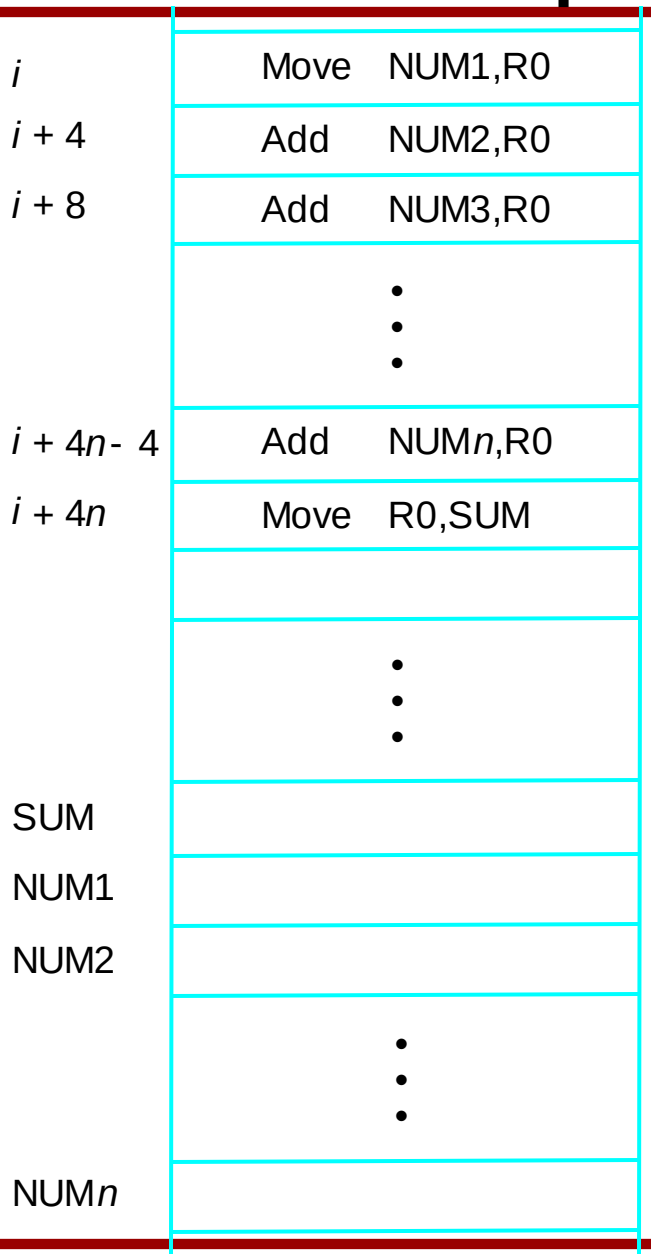


Figure 2.9. A straight-line program for adding *n* numbers.

Branching

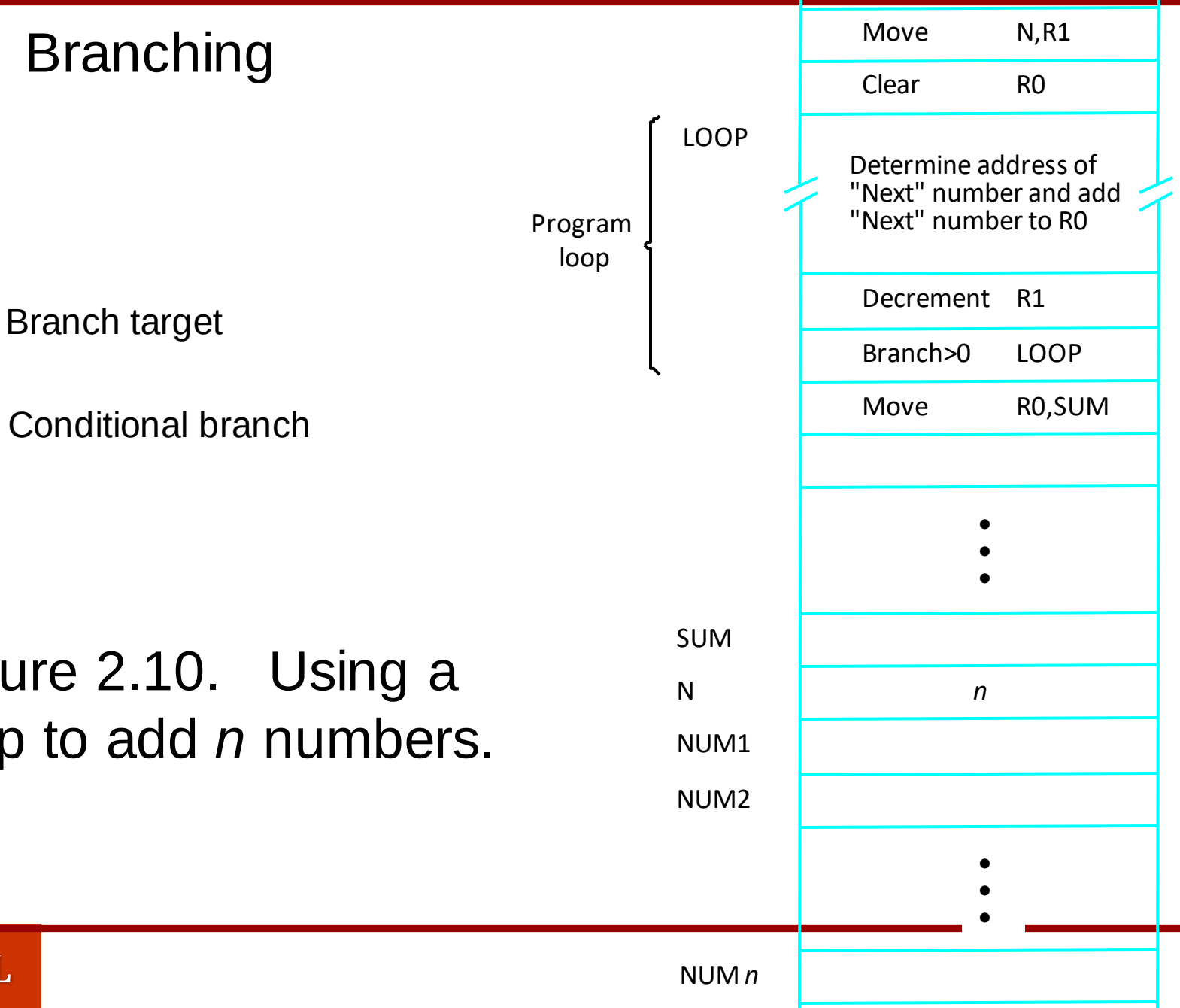


Figure 2.10. Using a loop to add *n* numbers.

THANK YOU

References (Book):

Computer Organization by Carl Hamacher, 5th Edition, McGraw Hill Education.

References (Web):

Online Author PPTs (http://www.technolamp.co.in/2011/04/computer-organization-carl-hamacher.html)