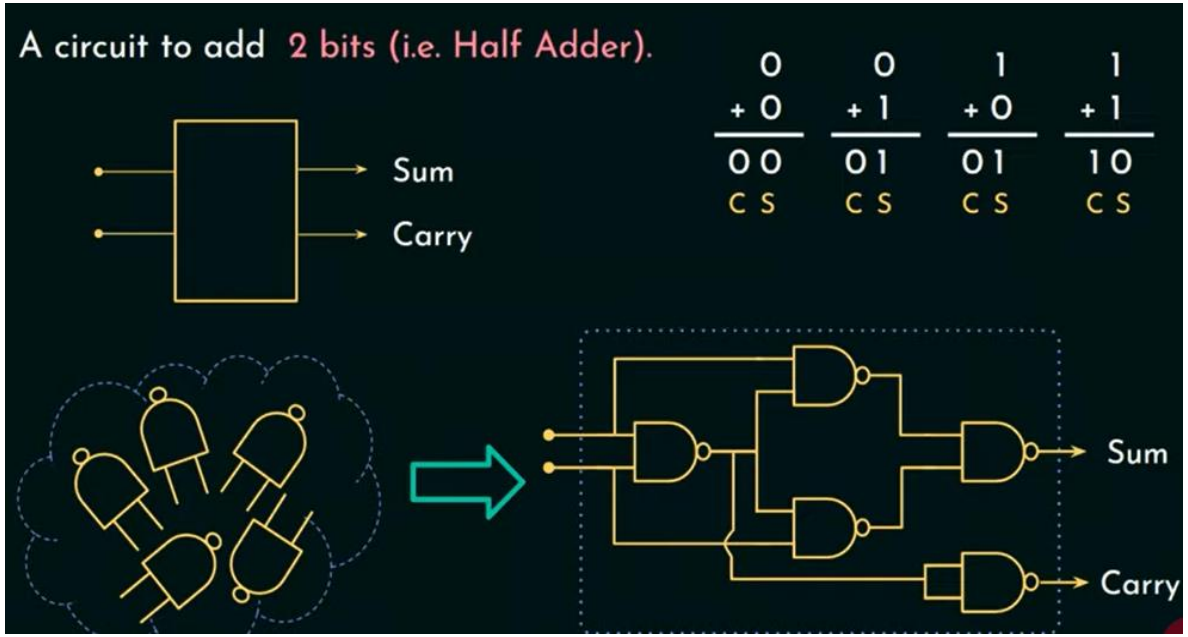
Digital Circuits & Computer Organization (NCAM32)

Module-1

Dr. Pramod T.C
Assistant Professor
Dept. of CSE
SIT

Computer Architecture and Organization



Computer Architecture	Computer Organization
Describes what the computer does (functional behaviour)	Describes how it does it (structural relationship)
For Designing a computer- its architecture is fixed first	For Designing a computer- organization decided after its architecture
We will decide - Instruction sets, number of bits used for data representation, registers, data types and addressing modes	Consists of physical units like circuit designs, peripherals, Control signals, interfaces, memory technology
e.g. Is there a multiply instruction?	-e.g. Is there a hardware multiply unit or is it done by repeated addition?

Parameters	Microprocessors	Microcontroller
Applications	Gaming, web browsing, document creation etc.	Dedicated for specific tasks (Camera, washing machine etc.)
Internal Structure	Memory and I/O devices connected externally	CPU, memory and I/O are present internally
Cost	High	Low
Power Consumption	High	Low
Memory (RAM)	512 MB to 32 GB	2KB to up to 256 KB
Storage	Hard Disk (128 GB to up to 2 TB)	Flash memory (32 KB to 2 MB)

Prescribed Book- examples are based on following processors: ARM, Motorola, Intel Pentium & Sun UltraSPARC

- **Computer:** It is a fast electronic calculating machine that accepts digitized input information, processes it according to the list of internally stored instructions & produces the resulting output information. The list of internally stored instructions is called as a Computer program & internal storage is called as Computer memory.

- **Functional units:** A computer consists of five functionally independent parts: Input, output, ALU, memory & control units

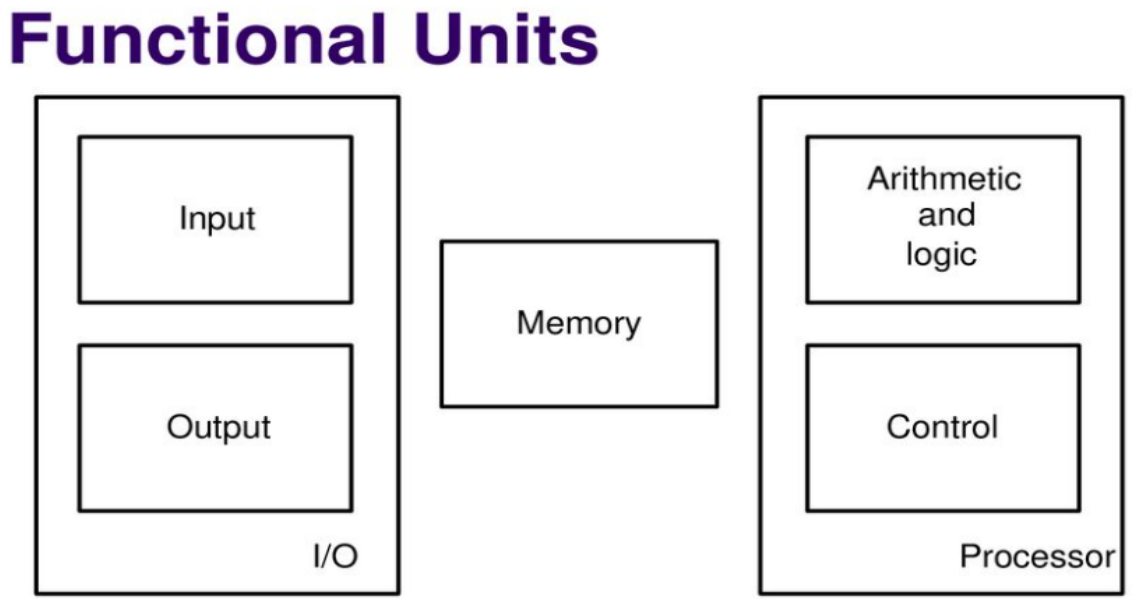


Figure 1.1. Basic functional units of a computer.

Input Unit

- Computers reads the data through input unit.
- The most common Input devices are Keyboard, joystick, trackballs, microphone and mouse
- When a key is pressed (keyboard), corresponding letter/digit is translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Alphanumeric characters are also expressed in the form binary codes. Two commonly used codes are:

- **ASCII** (American Standard code for Information Interchange)
- **EBCDIC** (Extended Binary-Coded Decimal Interchange Code)

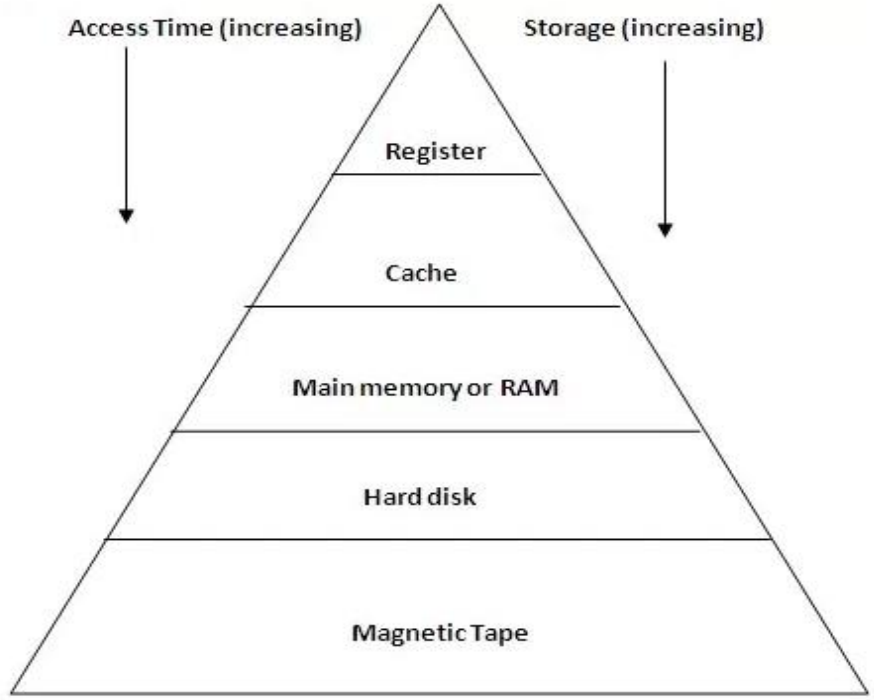
Memory Unit

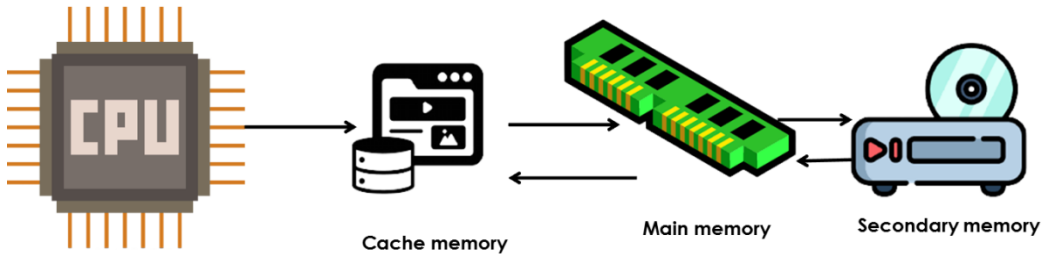
- Function: is to store programs and data
- There are 2 classes of storage:
  - Primary/Main memory (RAM, cache memory)
  - Secondary (ROM, flash drives, hard disk drives, magnetic tapes)
- **Primary Storage**
  - Fast memory that operates at electronic speeds
  - The memory contains a large number of semiconductor storage cells, each capable of storing 1 bit of information
  - These cells are processed in groups of fixed size called **word**
  - The number of bits in each word is known as **word length**
    - Range from 16 to 64 bits
  - Word size desribe the no. of bits of data processed by the microprocessor in one go (at a time) (Modern Computers: 32/64 bits).

Memory Unit

- Memory of a computer is normally implemented as a memory hierarchy of 3 or 4 levels of semiconductor RAM units with different speeds & sizes.
- Time required to access one word is called the **memory access time**
- The small, fast, RAM units are called caches.
- Register: used for temporary storage of calculations

Memory Hierarchy

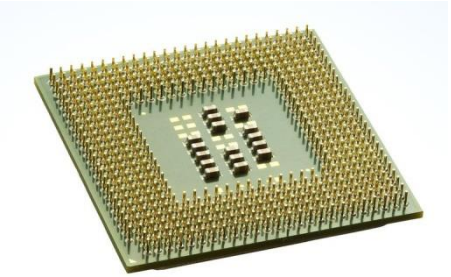




Memory Unit

- **Secondary Storage**
  - Is used when large amount of data and many programs have to be stored
  - It contains infrequently accessed information
  - Additional & cheaper memory
  - Ex: Magnetic disks and tapes & optical disks (CD-ROMs)

CPU: Arithmetic and Logic Unit



- ALU performs all the arithmetic and logic operations
- For ex: addition, multiplication, division, comparison etc
- Any operation is initiated by bringing the required operands into the processor, where the operation is performed by the ALU (Data transfer b/w memory & processor)
- Suppose two numbers located in the memory are to be added. They are brought into the processor, and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.
- When operands are brought into the processor, they are stored in high- speed storage elements called registers. Each register can store one word of data

Control Unit

- The memory, arithmetic and logic, and input and output units store and process information and perform input and output operations. **The operation of these units is coordinated by control unit.**

- The control unit is effectively the **nerve center** that sends control signals to other units and senses their states.

- It directs the operation of the other units by providing timing and control signals.
  - Timing signals are signals that determine when a given action is to take place.
  - Control signals supervise execution of the instructions

Output Unit

- Counterpart of Input unit
- Its function is to send processed results to outside world
- The familiar example of output device is printer (various types), monitor, speakers

Operation of a Computer - Summarized

- The computer accepts information in the form of pgms & data through an input unit & stores it in the memory
- Information stored in the memory is fetched, under pgm control, into an ALU, where it is processed
- Processed information leaves the computer through an Output unit
- All activities inside the machine are directed by the Control Unit

Basic Operational Concepts

- The information handled by a computer may be either *instruction* or *data*.

- **Instructions** are the explicit commands that

- Govern the transfer of information within a computer as well as between the computer and its I/O devices.
- Specify the arithmetic & logic operations to be performed.

Eg: Add LOCA, R0

Eg: ADD R1,R2

- A list of instructions that performs a task is called a **program**.

- Data are the numbers and encoded characters that are used as **operands** by the instructions.

Basic Operational Concepts

Consider the instruction **Add LOCA, R0**

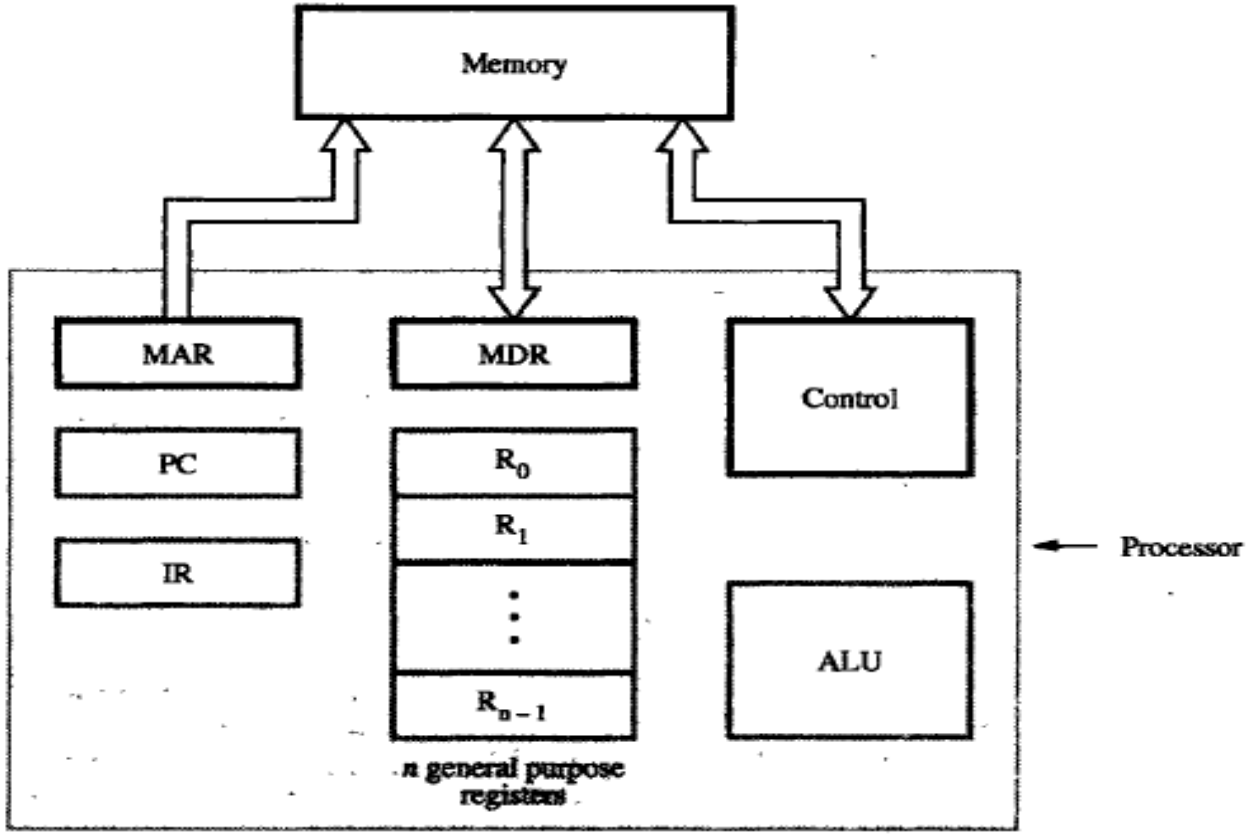
This instruction adds the operand at memory location LOCA to the operand in a register in the processor, R0, and places the sum into register R0.

This instruction requires the performance of several steps.

- First, the instruction is fetched from the memory into the processor.
- Next, the operand at LOCA is fetched and added to the contents of R0.
- Finally, the resulting sum is stored in register R0.

- Transfers between the memory and the processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data are then transferred to or from the memory.

Connection between the processor and memory



MAR - Memory Address Register

MDR - Memory Data Register

PC - Program Counter

IR - Instruction Register

Connection between the processor and memory

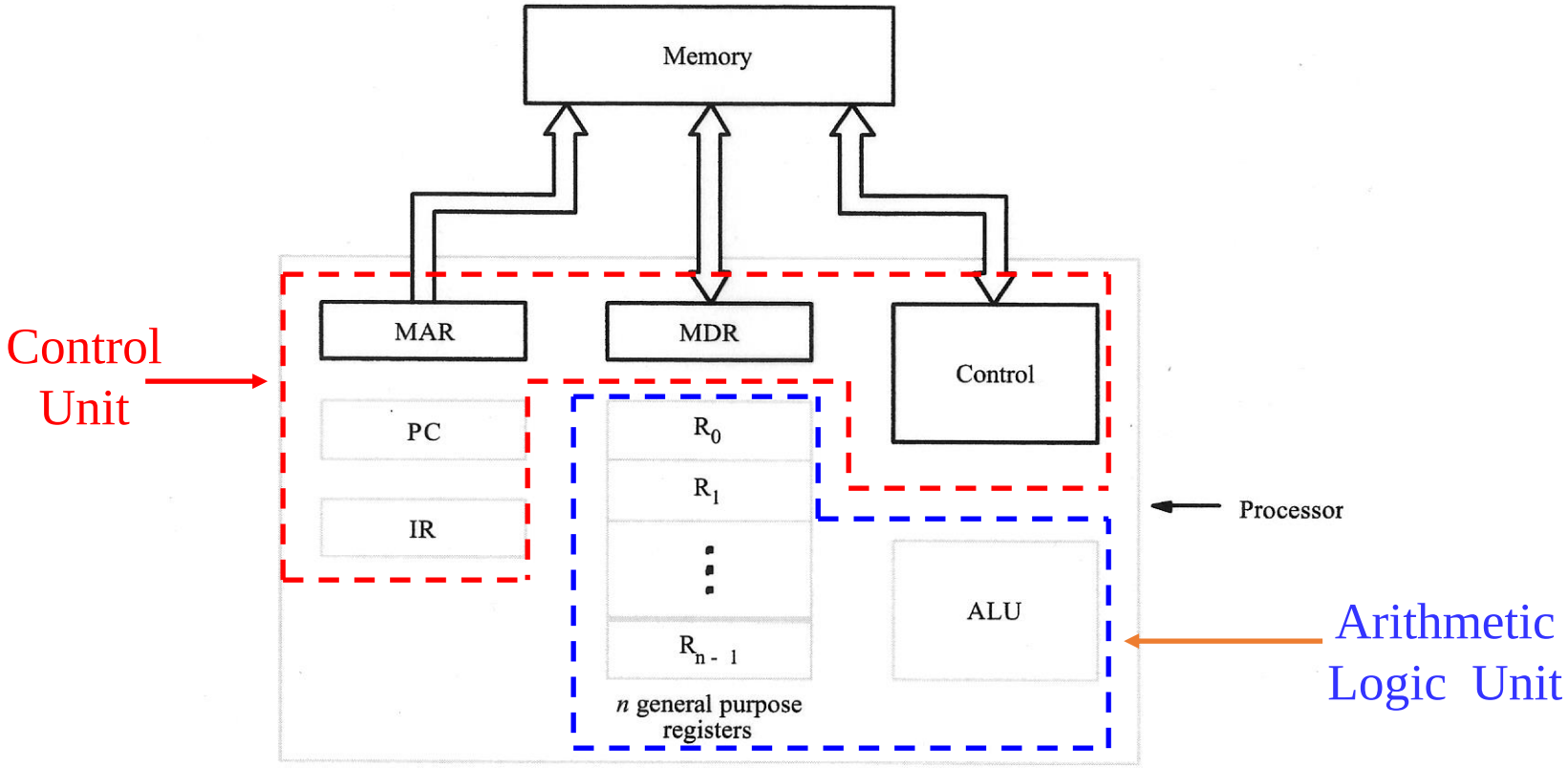


Figure 1.2. Connections between the processor and the memory.

MAR - Memory Address Register

MDR - Memory Data Register

PC - Program Counter

IR - Instruction Register

Types of registers

CPU = ALU + CU + Registers

In addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes.

- **PC (program counter)**
  - Keeps track of the execution of a program
  - Contains the memory address of **next** instruction to be fetched & executed

- **IR (instruction register)**
  - Holds the instruction that is **currently** being executed
  - Its o/p is available to control circuits, which generate the timing signals that control the various processing elements involved in executing the instruction

- **General- purpose registers (R0 to Rn-1):**
  - are used for holding data, intermediate results of operations.

Types of registers

CPU = ALU + CU + Registers

Two registers facilitate communication with the memory.

- **MAR (memory address register)**:
  - Facilitates communication with memory
  - Holds the address of the location to be accessed

- **MDR (memory data register):**
  - Facilitates communication with memory
  - Contains data to be written into or read out of the addressed location

Steps involved in executing a program/instruction

Step 1: PC is set to point to the first instruction of the program

Step 2: The contents of PC are transferred to the MAR

Step 3: Read control signal is sent to the memory.

Step 4: Instruction (first) of the program is read out of the memory and loaded into the MDR.

Step 5: The contents of MDR are transferred to the IR.

(At this point instruction is fetched from the memory)

Step 6: Decode and execute the instruction.

Step 7: If the instruction involves an operation to be performed by ALU, it is necessary to obtain the required operands. If an operand resides in the memory, it has to be fetched by sending its address to the MAR and initiating a Read signal.

Steps involved in executing a program/instruction

Step 8: When the operand has been read from the memory into the MDR, it is transferred from the MDR to the ALU. After one or more operands are fetched in this way, the ALU can perform the desired operation.

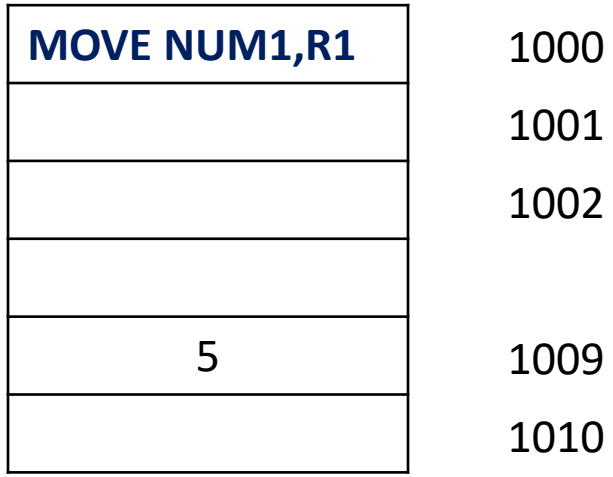
Step 9: Store the result back

- To general-purpose register
- To memory (address of the location where the result is to be stored is sent to MAR, result is sent to MDR – Write cycle is initiated)

-During the execution of current instruction, PC is incremented and points to the next instruction to be executed

Example: Steps involved in executing a Instruction

MOVE NUM1(1009),R1



- Fetch
  - MAR ← [PC]
  - PC ← [PC] + 1
  - MDR ← [MEM([MAR])]
  - IR ← [MDR]
- Decode
- Execute
  - MAR ← NUM1
  - MDR ← [MEM([MAR])]
  - R1 ← [MDR]

Another Example

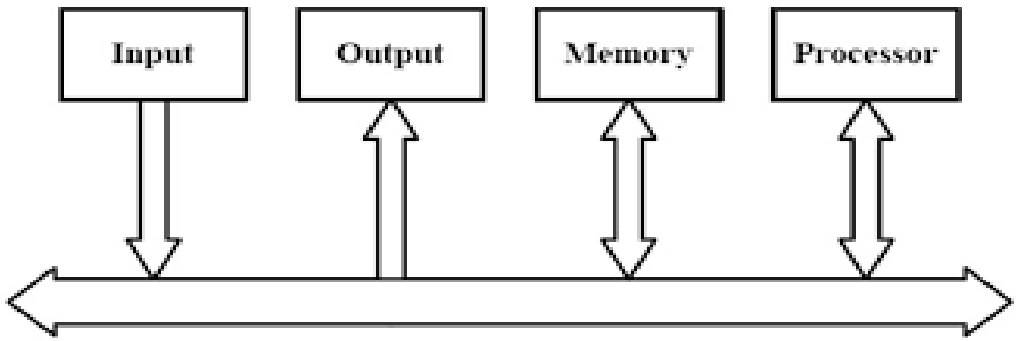
ADD #1,R1

- Fetch
  - MAR ← [PC]
  - PC ← [PC] + 1
  - MDR ← [MEM([MAR])]
  - IR ← [MDR]
- Execute
  - R1 ← 1 + [R1]

Bus Structures

- **Bus:** Set of wires/lines, that interconnects all the components (subsystems) of a computer
- In addition to the lines that carry the data, the bus must have lines for address and control information.

Single Bus Structure



All units are connected to the same bus. Since the bus can be used for only one transfer at a time, only two units can actively use the bus at any given time.

Bus Structures

Single Bus Structure

Advantages:

- Simple
- Low cost
- Flexibility to attach peripheral devices.

Disadvantages:

- Concurrent operations cannot be performed.
- Lower performance

Multiple Bus Structure

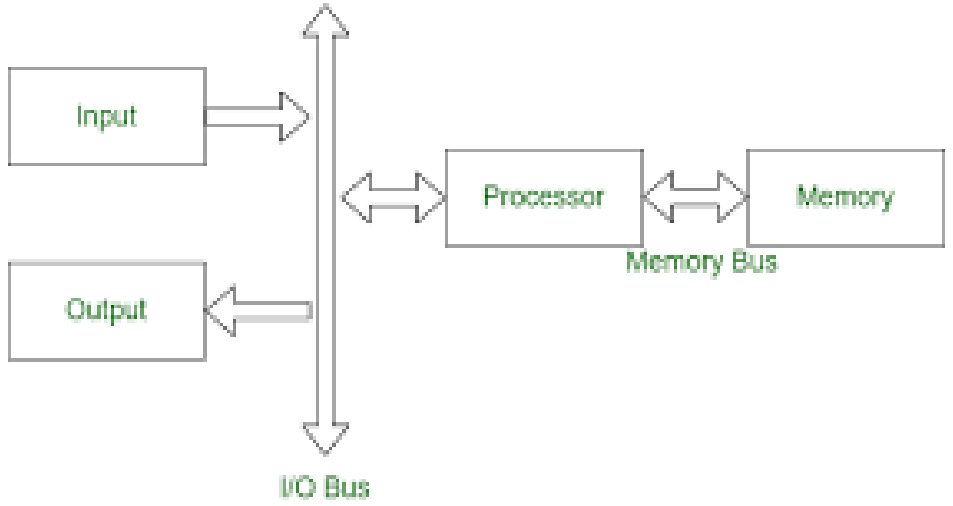
- Multiple Buses are used to interconnect functional units.

Advantages:

- Concurrent operations can be performed
- Higher performance

Disa*dvantages:*

- Complex
- Costlier



Double Bus Structure

Bus Structures

- The devices connected to a bus **vary widely in their speed of operation**.
  - Some electromechanical devices, such as keyboards and printers, are relatively slow.
  - Others, like magnetic or optical disks, are considerably faster.

- Results in
  - **Timing differences** among processors, memories, and I/O devices.
  - **High-speed processor being locked** to a slow I/O device during a sequence of data transfers.

- Solution- To smooth out the differences in timing among processors, memories, and external devices - **buffer registers are used**

Eg: Printing a file

- The processor sends the contents of the file over the bus to the printer buffer
- Since the buffer is an electronic register, this transfer requires relatively little time. Once the buffer is loaded, the printer can start printing without further intervention by the processor.
- The bus and the processor are no longer needed and can be used for other activities.

PERFORMANCE

The most important measure of the performance of a computer is how quickly it can execute programs.

The speed with which a computer executes programs is affected by:

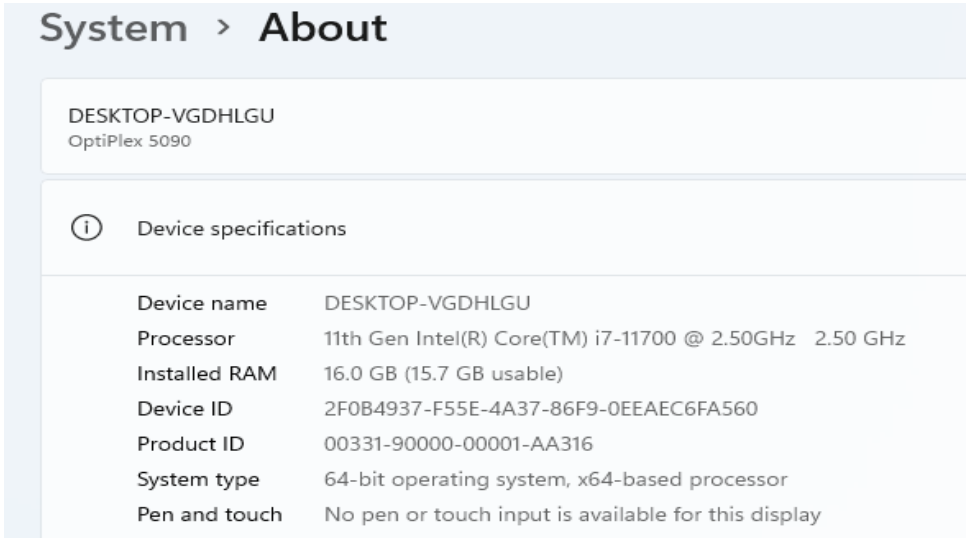
1. The design of its hardware (memory and processor)
2. Its machine language instructions.
3. The compiler that translates programs (high level code) into machine language.

PERFORMANCE

. PROCESSOR CLOCK

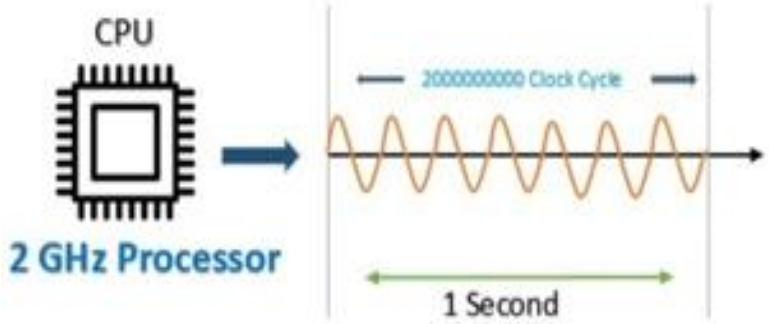
- Processor circuits are controlled by a timing signal called a *clock*.
- The clock defines regular time intervals, called *clock cycles*.
- To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps, such that each step can be completed in one clock cycle.
- The length P of one clock cycle is an important parameter that affects processor performance. Its inverse is the clock rate,

$$R = 1 / P,$$ which is measured in cycles per second.



Example

Note



Suppose, this processor required 4 clock cycle to execute 1 instruction (task & process).
Please note, the actual capacity depends on processor technology.

$$\frac{2000000000 \text{ Clock Cycle}}{4 \text{ Clock cycle per instruction}}$$

Means, this processor is capable to handle 5000000 instructions in one minute.

Suppose, your processor has to execute 7000000 instructions in one second because of heavy use. But your processor can execute only 5000000 instructions

Then, 2000000 instructions will need to wait in the queue.

BASIC PERFORMANCE EQUATION

$$T = (N * S) / R$$

T-processor time required to execute a program

N-actual number of instruction executions

S-steps needed to execute one machine instruction

R- clock rate (*R* cycles per second)

To achieve **high performance**, the computer designer must seek ways to **reduce the value of *T*,** which means reducing *N* and S, and increasing *R*.

- The value of *N* is reduced if the source program is compiled into fewer machine instructions.
- The value of S is reduced if instructions have a smaller number of basic steps to perform or if the execution of instructions is overlapped.
- Using a higher-frequency clock increases the value or *R*, which means that the time required to complete a basic execution step is reduced.

Clock Rate

$$T = (N * S) / R$$

There are two possibilities for increasing the clock rate, *R*.

- First, improving the ***integrated-circuit*** **(IC) technology makes logic circuits** faster, which reduces the time needed to complete a basic step. This allows the clock period, *P*, to be reduced and the clock rate, *R*, to be increased.

- Second, **reducing the amount of processing done in one basic step** also makes it possible to reduce the clock period, *P*. However, if the actions that have to be performed by an instruction remain the same; the number of basic steps needed may increase.

PERFORMANCE MEASUREMENT

T (execution time) – measure the performance of a computer

- But computing the value of T is not simple. Parameters such as the clock speed (R) and various architectural features are not reliable indicators of the expected performance.

- Computer community adopted the idea of measuring computer performance using **benchmark programs**. The **performance measure** is the time it takes a computer to execute a given benchmark.

- A nonprofit organization called **System Performance Evaluation Corporation (SPEC**) selects the benchmark programs - game playing, compiler, and database applications to numerically intensive programs in astrophysics and quantum chemistry.

- The program is compiled for the computer under test, and the running time on a real computer is measured. The same program is also compiled and run on one computer selected as a reference.

PERFORMANCE MEASUREMENT

$$\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

- SPEC rating of 50 means that the computer under test is 50 times faster than the reference computer. The test is repeated for all the programs in the SPEC suite, and the geometric mean of the results is computed.

- The overall SPEC rating for the computer is given by

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}_i \right)^{\frac{1}{n}}$$

  Where ***n*** is the number of programs in the suite.

- SPEC rating is a measure of the combined effect of all the factors affecting performance, including the compiler, the operating system, the processor, and the memory of the computer being tested

$$\left(\prod_{i=1}^{n} x_i \right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \ldots x_n}$$

Note:

RISC (Reduced Instruction Set Computer) Architectures
CISC (Complex Instruction Set Computer) Architectures

Parameter	RISC	CISC
Instruction types	Simple	Complex
Number of instructions	Reduced (30-40)	Extended (100-200)
Duration of an instruction	One cycle	More cycles (4-120)
Instruction format	Fixed	Variable
Instruction execution	In parallel (pipeline)	Sequential
Addressing modes	Simple	Complex
Instructions accessing the memory	Two: Load and Store	Almost all from the set
Register set	multiple	unique
Complexity	In compiler	In CPU (micro-program)

RISC: Used in mobile phones , used in Apple
CISC: Laptops, used in Intel, AMD

```
CISC
mult A,B
```

```
load m1,A
load m2,B
Prod A,B
Store m3,A
```

Problems

1. For a RISC machine, the effective value of S is 1.25 and the average value of N is 200. If the clock rate is 500 MHz, calculate the total program execution time.

$$T = (N \times S)/R = (200 \times 1.25)/(500 \times 10^6) = 0.5\ \mu s$$

2.The effective value of S for a RISC machine is 1.2 and for CISC is 1.5. Both machines have the same clock rate R. The time for execution on CISC machine is to be no longer than that of RISC machine. For this to happen, what is the largest allowable value for N, i.e., no. of instructions executed on CISC machine expressed as a percentage of N value for the RISC machine.

Given, $S_C=1.5$, $S_R=1.2$, $R_C=R_R=R$

$$(1.2 \times N_R)/R = (1.5 \times N_C)/R$$

$$1.2N_R = 1.5N_C$$

$$N_C/N_R = 0.8$$

No. of instructions executed on CISC machine is 80% of the no. of instructions executed on RISC machine.

3.Assuming that the reference computer is Ultra SPARC10 workstation with 300 MHz Ultra SPARC processor. A company has to purchase 1000 new computers, hence ordered testing of new computers with SPEC 2000. Following observations were made:

Programs	Runtime on reference computer	Runtime on new computer
1	50 mins	5 mins
2	75 mins	4 mins
3	60 mins	6 mins
4	30 mins	3 mins

- The company system manager will place the order for purchasing new computers only if the overall SPEC rating is at least 12. After the said test, will system manager place order for purchase of new computer?

$SPEC_1 = 50/5 = 10$

$SPEC_2 = 75/4 = 18.75$

$SPEC_3 = 60/6 = 10$

$SPEC_4 = 30/3 = 10$

Overall SPEC rating = $(\prod_1^n SPECi)^{1/n}$ = 11.70

Since overall SPEC rating is 11.70, the purchase order will not be placed.

4.List the steps needed to execute the machine instruction **Add LOCA,R0** in terms of transfers between the components of the processor and some control commands. Assume that the instruction itself is stored in the memory at location INSTR and that this address is initially in register PC. Also include the steps needed to update the contents of PC from INSTR to INSTR+1 so that the next instruction can be fetched.

- Transfer the contents of PC to MAR.
- Issue Read signal to memory and wait until requested word is transferred into MDR.
- Transfer MDR content to IR and decode it.
- Transfer the address LOCA to MAR.
- Issue Read signal and wait until MDR is loaded with the data.
- Transfer MDR content to the ALU.
- Transfer contents of R0 to the ALU.
- Perform addition of the two operands in the ALU and transfer the result into R0.
- Transfer the contents of PC to ALU.
- Add 4 to operand in ALU and transfer incremented address to PC.

NUMBER REPRESENTATION

Consider an n-bit vector

$$B = b_{n-1} \ldots b_1 b_0$$

Where, $b_i = 0$ or 1 for $0 <= i <= n-1$.

There is a need to represent both positive and negative numbers. ***Three systems are used for representing such numbers***:

- Sign-and-magnitude
- 1's-complement
- 2's-complement

0 for positive numbers
1 for negative numbers

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

NUMBER REPRESENTATION

B	Values represented		
$b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

sign-and-magnitude system, negative values are represented by changing the most significant bit from 0 to 1

1's-complement representation :The operation of forming the1's-complement of a given number is equivalent to subtracting that number from 2^*n* – 1
(In this example 2^*3* =*8-1=7)*

2's-complement system, forming the 2's-complement of a number is done by subtracting that number from 2^*n*

ADDITION OF POSITIVE NUMBERS

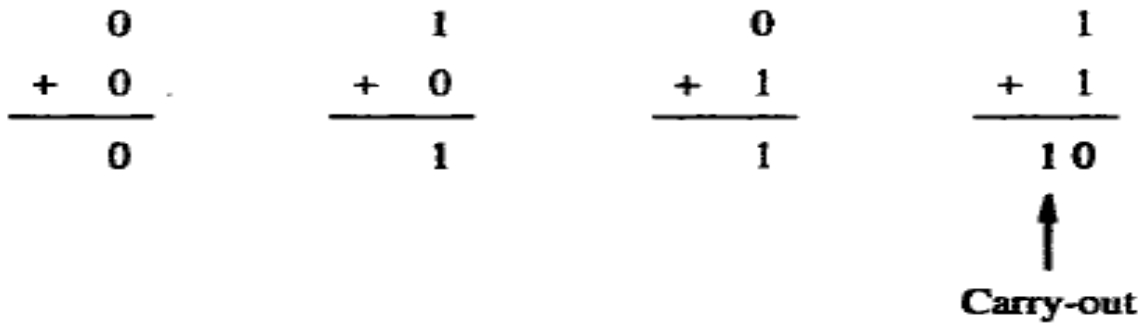
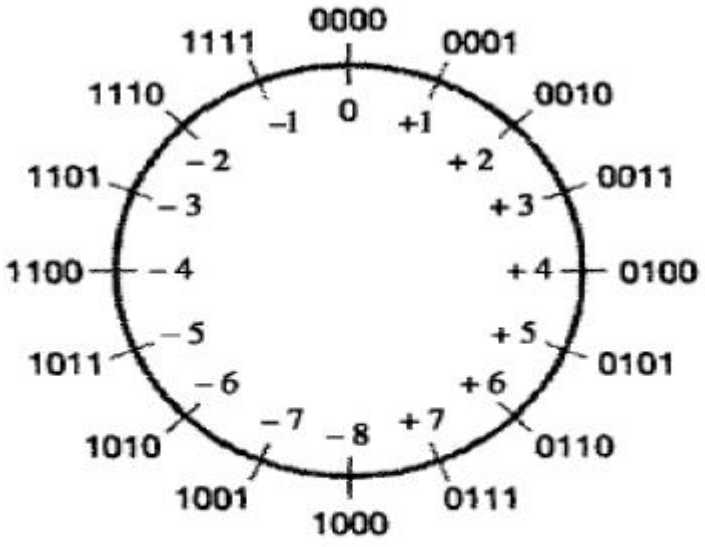


Figure 2.2 Addition of 1-bit numbers.

The sum of 1 and 1 requires the 2-bit vector 10 to represent the value 2. We say that the *sum* is 0 and the *carry-out* is 1.

ADDITION AND SUBTRACTION OF SIGNED NUMBERS



(b) Mod 16 system for 2's-complement numbers

Figure 2.3 Modular number systems and the 2's-complement system.

$N = 16$.

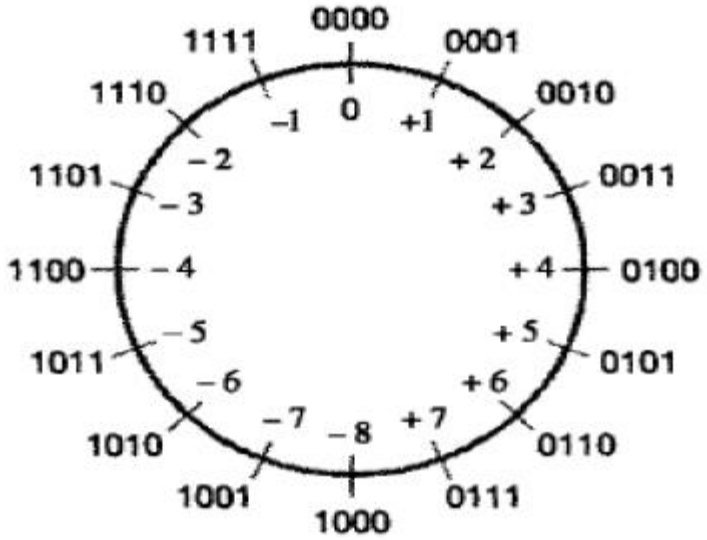
(7+4) mod 16 yields the value 11.

Locate 7 on the circle and then move 4 units in the clockwise direction to arrive at the answer 11.

(9 + 14) mod16 = 7; this is modeled on the circle by locating 9 and moving 14 units in the clockwise direction to arrive at the answer 7.

For the computation of $(a + b)$ mod 16 for any positive numbers a and b, that is, to perform addition, locate a and move b units in the clockwise direction to arrive at $(a + b)$ mod 16.

ADDITION AND SUBTRACTION OF SIGNED NUMBERS



(b) Mod 16 system for 2's-complement numbers

Figure 2.3 Modular number systems and the 2's-complement system.

addition of +7 to -3.

2's-complement representation for these numbers is 0111 and 1101

locate 0111 on the circle then move 1101 (13) steps in the clockwise direction to arrive at 0100, which yields the correct answer of +4.

```
  0 1 1 1
+ 1 1 0 1
---------
1 0 1 0 0
↑
Carry-out
```

The rules for addition and subtraction of *n*-bit signed numbers using the 2's-complement recomplement representation system are:

1. To *add* two numbers, add their *n*-bit representations, **ignoring the carry-out signal from the *most significant bit* (MSB) position**. The sum will be the algebraically correct value in the 2's-complement representation as long as the answer is in the range -2^*n*-1 through +2^*n*-1 – 1

   EX: (-2^4-1 to +2^4-1 -1 = -8 to +7).

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

(a)	0010	(+2)	(b)	0100	(+4)
	+ 0011	(+3)		+ 1010	(−6)
	0101	(+5)		1110	(−2)
(c)	1011	(−5)	(d)	0111	(+7)
	+ 1110	(−2)		+ 1101	(−3)
	1001	(−7)		0100	(+4)

The rules for addition and subtraction of *n*-bit signed numbers using the 2's-complement recomplement representation system are:

2. To subtract two numbers X and Y, that is, to perform X - Y, form the 2'scomplement of Y and then add it to X, as in rule 1. Again, the result will be the algebraically correct value in the 2's-complement representation system if the answer is in the range -2^n-1 through +2^n-1 - 1.

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

(e)

1101	(−3)	⇒	1101		
− 1001	(−7)		+ 0111		
			0100	(+4)	

(f)

0010	(+2)	⇒	0010		
− 0100	(+4)		+ 1100		
			1110	(−2)	

The rules for addition and subtraction of *n*-bit signed numbers using the 2's-complement recomplement representation system are:

2. To subtract two numbers X and Y, that is, to perform X - Y, form the 2'scomplement of Y and then add it to X, as in rule 1. Again, the result will be the algebraically correct value in the 2's-complement representation system if the answer is in the range -2^n-1 through +2^n-1 - 1.

B	Values represented		
$b_3b_2b_1b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

(g)
```
  0 1 1 0      (+6)
- 0 0 1 1      (+3)
---------      ----
                          0 1 1 0
                        + 1 1 0 1
                        ---------      ----
  0 0 1 1      (+3)      0 0 1 1      (+3)
```

(h)
```
  1 0 0 1      (−7)
- 1 0 1 1      (−5)
---------      ----
                          1 0 0 1
                        + 0 1 0 1
                        ---------      ----
  1 1 1 0      (−2)      1 1 1 0      (−2)
```

(i)
```
  1 0 0 1      (−7)
- 0 0 0 1      (+1)
---------      ----
                          1 0 0 1
                        + 1 1 1 1
                        ---------      ----
  1 0 0 0      (−8)      1 0 0 0      (−8)
```

(j)
```
  0 0 1 0      (+2)
- 1 1 0 1      (−3)
---------      ----
                          0 0 1 0
                        + 0 0 1 1
                        ---------      ----
  0 1 0 1      (+5)      0 1 0 1      (+5)
```

Why 2’s-complement representation is used in modern computers?

- The simplicity of either adding or subtracting signed numbers in 2’s-complement representation is the reason why this number representation is used in modern computers.

- In 1's there is a -0 (1111) and a +0 (0000), i.e two value for the same 0. On the other hand, in 2's complement, there is only one value for 0 (0000).

+0 --> 0000 and

-0 --> 1000 --> 0111 + 1 --> 000

- While doing arithmetic operations like addition or subtraction using 1's, we have to add an extra carry bit, i.e 1 to the result to get the correct answer, +7 (0111) add -7 (1000) = 1111 +1 =0000

- Whereas in 2’s complement +7 (0111) add -7 (1001) = 0000

OVERFLOW IN INTEGER ARITHMETIC

- In the 2's-complement number representation system, n bits can represent values in the range -2^n-1 to +2^n-1 - 1.

- 4 bit number system, the range of numbers that can be represented is -8 through +7. If the result of an arithmetic operation is outside the representable range, then we say that *arithmetic overflow* has occurred.

- +7 add +4 = 1011 (-5, an incorrect result) carry=0
- –4 and -6, we get S = 0110 = +6 (another incorrect result) carry=1

- Thus, overflow may occur if both summands have the same sign. Clearly, the addition of numbers with different signs cannot cause overflow.

This leads to the following conclusions:

1. Overflow can occur only when adding two numbers that have the same sign.

2. The carry-out signal from the sign-bit position is not a sufficient indicator of overflow when adding signed numbers.

MEMORY LOCATIONS AND ADDRESSES

- Memory consists of many millions of storage c*ells*, each of which can store a bit of information having the value 0 or 1.
- Bits are not handled individually, a single bit represents a very small amount of information
- The usual approach is to deal with them in groups of fixed size.
- Each group of *n* bits is referred to as a *word* of information, and *n* is called the *word length*.

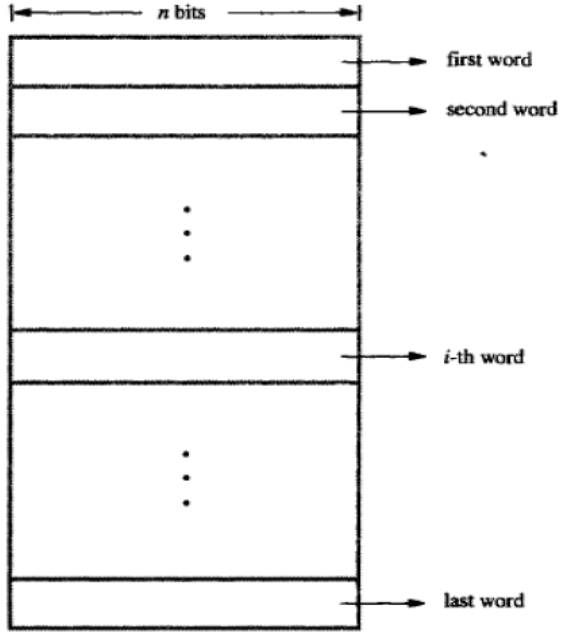


Figure 2.5 Memory words.

MEMORY LOCATIONS AND ADDRESSES

- The word length of the computer typically ranges from 16 to 64 bits.
- If the word length of a computer is 32 bits, a single word can store a 32-bit 2's-complement number or four ASCII characters, each occupying 8 bits. A unit of 8 bits is called a byte.

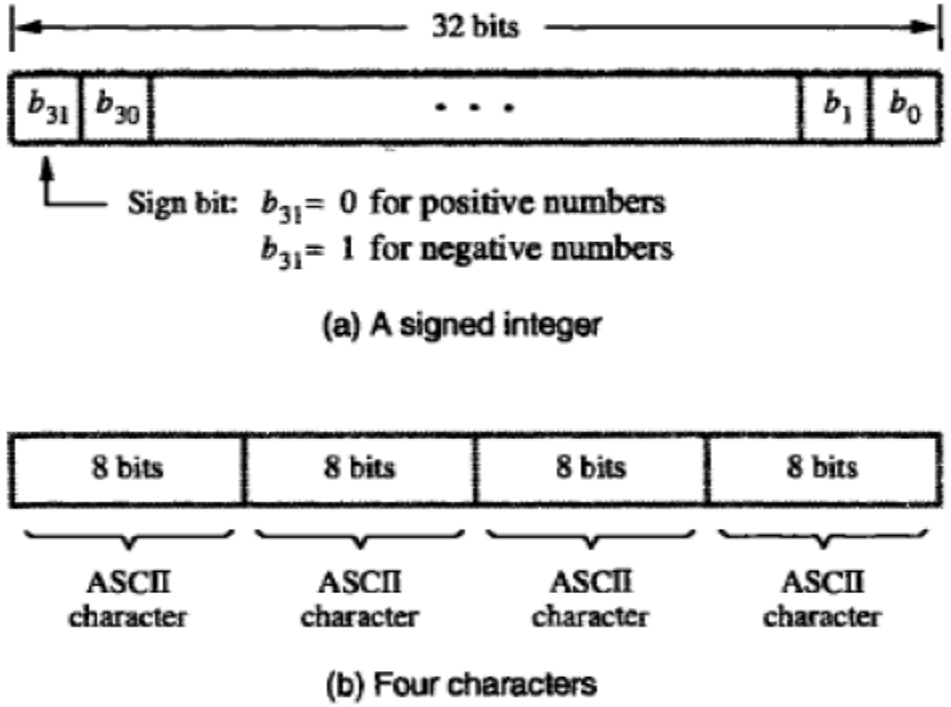


Figure 2.6 Examples of encoded information in a 32-bit word.

MEMORY LOCATIONS AND ADDRESSES

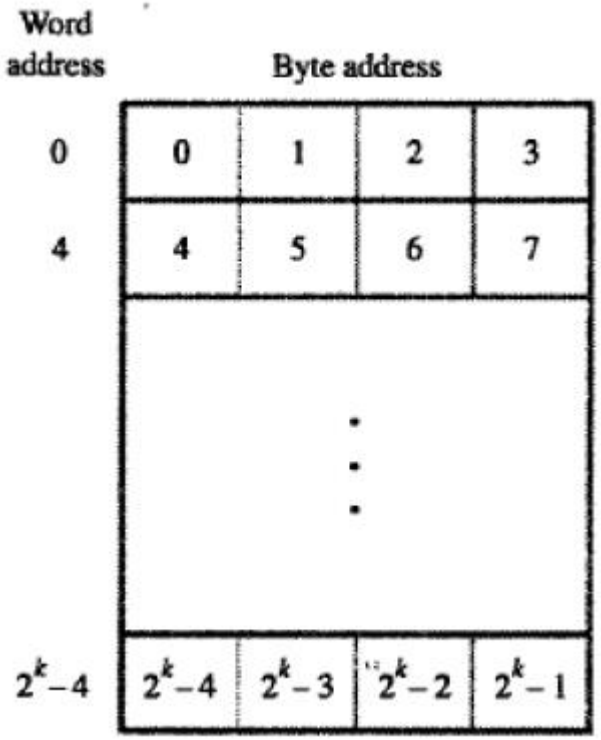
- Accessing the memory - Word or a byte, requires distinct names or *addresses* for each item location.
- Range is from 0 through 2^*k*-1, for some suitable value of *k*.
- ***address space*** of the computer: 2^*k* addresses and the memory can have up to 2^*k* addressable locations.
- An address space is a range of valid addresses in memory that are available for a program or process
- 24-bit address generates an address space of 2^24 (16,777,216) locations or 16M (16 Mega) , where 1M is the number 2^20 (1,048,576).
- A 32-bit address creates an address space of 2^32 (4294967296) or 4G (4 Giga) locations, where 1G is 2^30.
- Tera- 2^40

BYTE ADDRESSABILITY

- A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits.

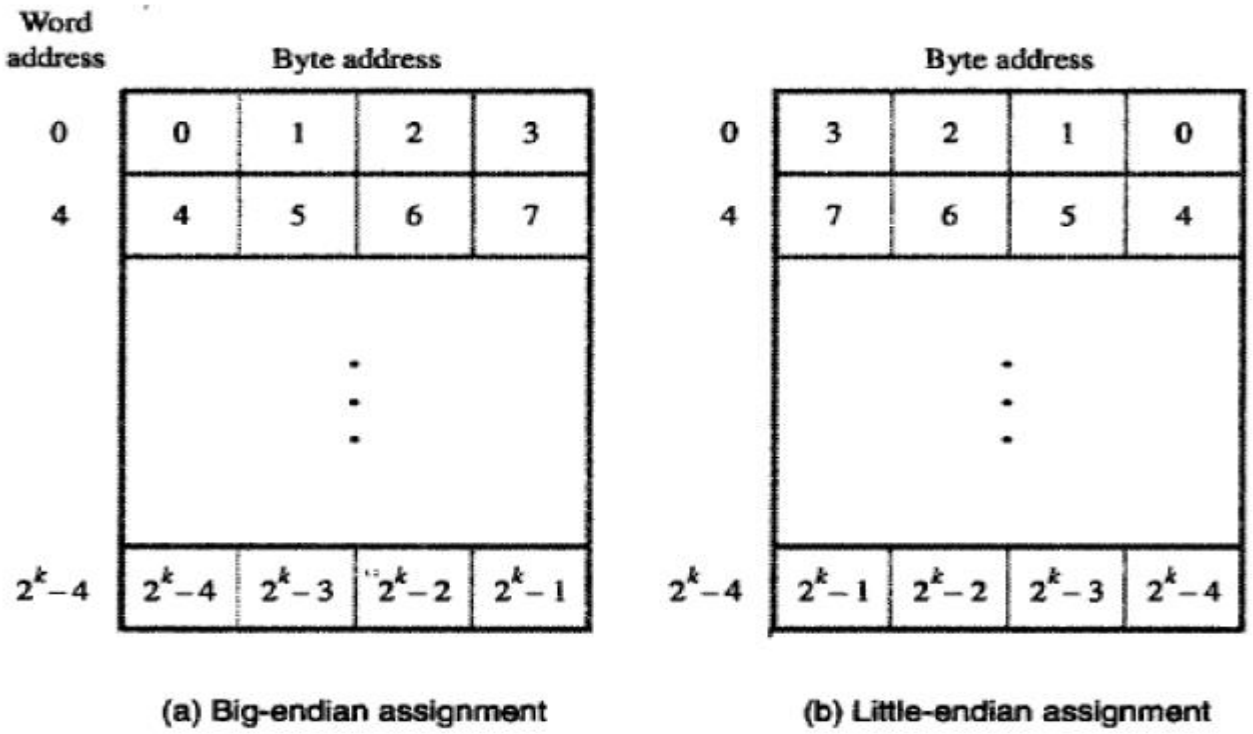
- It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to **successive byte locations in the memory** - term byte-addressable memory is used for this assignment

- Byte locations have addresses 0, 1, 2, ... Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8, . . . with each word consisting of four bytes.



BIG-ENDIAN AND LITTLE-ENDIAN ASSIGNMENTS

- Two ways that byte addresses can be assigned across words
- The name **big-endian** is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.
- The name **little-endian** is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.



(a) Big-endian assignment

(b) Little-endian assignment

WORD ALIGNMENT

- If the word length is 32-bit, then word boundaries occur at addresses 0, 4, 8, 12,16 . .

- We say that the word locations have *aligned* addresses. In general, ***words are said to be aligned*** in memory if they begin at a byte address that is a multiple of the number of bytes in a word.

- In general, the number of bytes in a word is a power of 2. Hence, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4, 6. . . , and for a word length of 64 (2^3 bytes), aligned words begin at byte addresses 0, 8, 16, ...

- If words begin at an arbitrary byte address then they are said to have ***unaligned addresses***.

MEMORY OPERATIONS

- Both program instructions and data operands are stored in the memory.
- To execute an instruction by processor control circuits - transfer is required between memory to the processor.
- Two basic operations involving the memory are ***Load*** (or *Read* or *Fetch*) and ***Store*** (or *Write*).

Load operation

- Transfers a copy of the contents of a specific **memory location to the processor**. The memory contents remain unchanged.
- To start a Load operation, the processor sends the address of the desired location to the memory and requests that its contents be read. The memory reads the data stored at that address and sends them to the processor.

Store operation

- Transfers an item of information from **the processor to a specific memory location**, destroying the initial contents of that location.
- The processor sends the address of the desired location to the memory, together with the data to be written into that location.

INSTRUCTIONS AND INSTRUCTION SEQUENCING

The instruction set provides commands to the processor, to tell it what it needs to do.

A computer must have instructions capable of performing four types of operations: (Instruction Set Architecture)

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

Note

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data transfer instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate	NEG

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Arithmetic and Logical

TABLE 10-7
Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Call procedure	CALL
Return from procedure	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TEST

Program Control

Computer Instructions

- To transfer the information from one location in the computer to another the following locations are involved: **memory locations, processor registers, or registers in the I/O subsystem.**

The location is identified by a symbolic name. For example,

- names for the addresses of memory locations may be LOC, PLACE, A, VAR2;
- processor register names may be R0, R5; and
- I/O register names may be DATAIN, OUTSTATUS, and so on.

REGISTER TRANSFER NOTATION

- The contents of a location are denoted by placing square brackets around the name of the location. Thus, the expression

$R1 \leftarrow [LOC]$

contents of memory location LOC
are transferred into processor register R1.

$R3 \leftarrow [R1] + [R2]$

- Transfers among registers
- The right-hand side of an RTN expression always denotes a value, and the left-hand side is the name of a location where the value is to be placed, overwriting the old contents of that location.

ASSEMBLY LANGUAGE NOTATION

Move LOC,R1	transfer from memory location LOC to processor register R1
Add R1,R2,R3	two numbers contained in processor registers R1 and R2 and placing their sum in R3

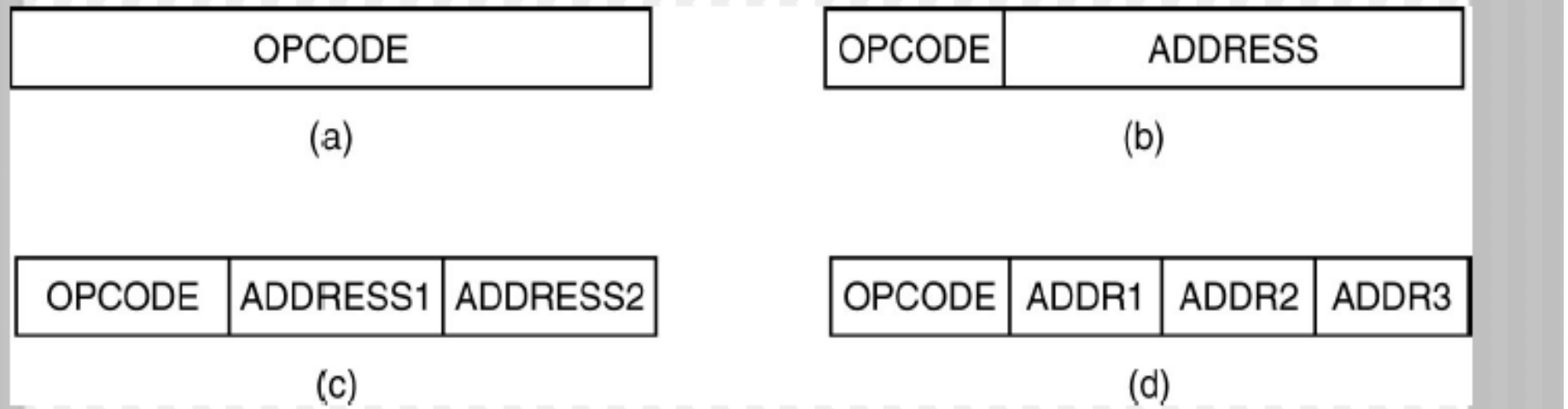
Assembly Language

- MOVE NUM1,R1
- MOVE #1,R2
- ADD #1,R1
- ADD R1,R2

Register Transfer Notation

- R1 ← [NUM1]
- R2 ← 1
- R1 ← 1 + [R1]
- R2 ← [R1] + [R2]

BASIC INSTRUCTION TYPES (Common Instruction Formats)



Four common instruction formats:

(a) Zero-address instruction.

(b) One-address instruction

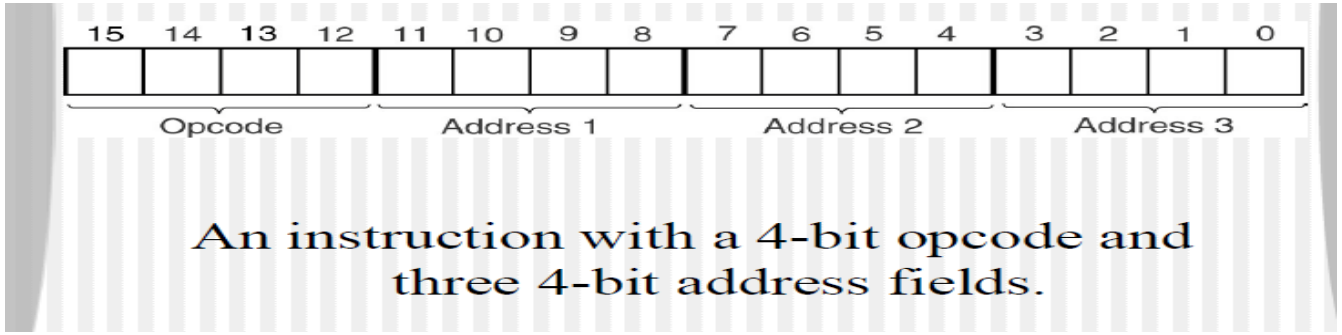
(c) Two-address instruction.

(d) Three-address instruction

Three-address instruction:

Format: Operation Source1,Source2,Destination

- If *k* bits are needed to specify the memory address of each operand, the encoded form of the above instruction must contain 3*k* bits for addressing purposes in addition to the bits needed to denote the Add operation.



An instruction with a 4-bit opcode and three 4-bit address fields.

Example X=(A+B)(C+D)

ADD A, B,T1	M[A]+M[B] →M[T1]
ADD C, D, T2	M[C]+M[D] → M[T2]
MUL T1, T2,X	M[T1]+M[T2] → M[X]

Two-address instruction:

- **Format: Operation Source, Destination**

Add A,B (B is both a source and a destination)

The problem in two address instruction is one of the operand is destroyed.

Move B,C
Add A,C

Example: X=(A+B)(C+D)

MOVE A,T1	M[A] → M[T1]
ADD B,T1	M[T1]+M[B] → M[T1]
MOVE C,X	M[C] → M[X]
ADD D,X	M[X]+M[D] → M[X]
MUL T1,X	M[X]+M[T1] → M[X]

One-address instruction:

- Machine instructions that specify only one memory operand. The second operand is implicitly specified in the instruction.
- A processor register, called the **accumulator**, is used for this purpose.
- An **accumulator** is primarily used as a register in a CPU to store intermediate logical or arithmetic data in multistep calculations.

Add A	adds the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator.
Load A	copies the contents of memory location A into the accumulator.
Store A	copies the contents of the accumulator into memory location A.

the operation C ← [A]+[B] can be performed by

Load A

Add B

Store C

One-address instruction:

Example X=(A+B)(C+D)

- LD A M[A] → AC
- ADD B ACC+M[B] → AC
- ST X AC→ M[X]
- LD C M[C] → AC
- ADD D AC+M[D] → AC
- MUL X AC*M[X] → AC
- ST X AC → M[X]

Zero-address instruction

- Stack architecture
- store operands in a structure called a pushdown stack

C = A + B

PUSH A
PUSH B
ADD
POP C

Zero-address instruction

Example X=(A+B)(C+D)

- PUSH A M[A] → TOS
- PUSH B M[B] → TOS
- ADD TOS+TOS-1 → TOS
- PUSH C M[C] → TOS
- PUSH D M[D] → TOS
- ADD TOS+TOS-1 → TOS
- MUL TOS*TOS-1 → TOS
- POP X TOS → M[X]

TOS: Top of the Stack

INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

Consider the task, $C \leftarrow [A] + [B]$

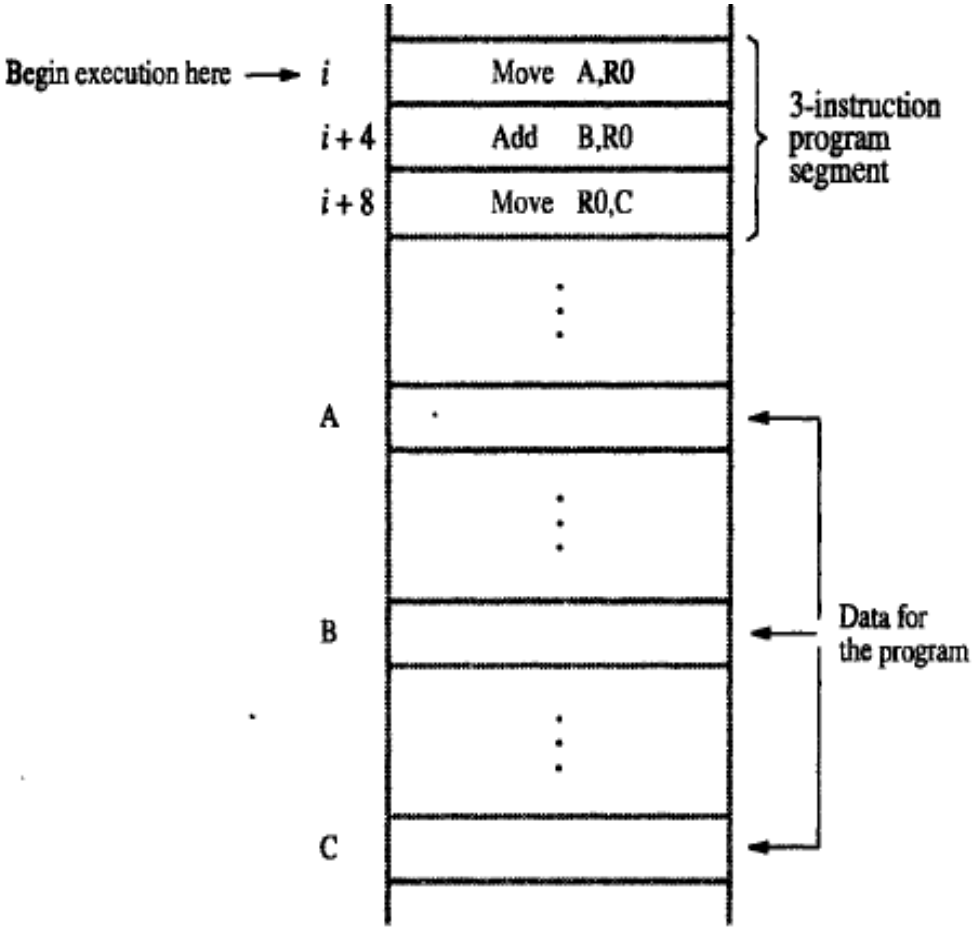


Figure 2.8 A program for C ← [A] + [B].

- Word length is 32 bits and the memory is byte addressable
- The three instructions of the program are in successive word locations, starting at location i. Since each instruction is 4 bytes long, the second and third instructions start at addresses i + 4 and i + 8.

INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

Consider the task, $C \leftarrow [A] + [B]$

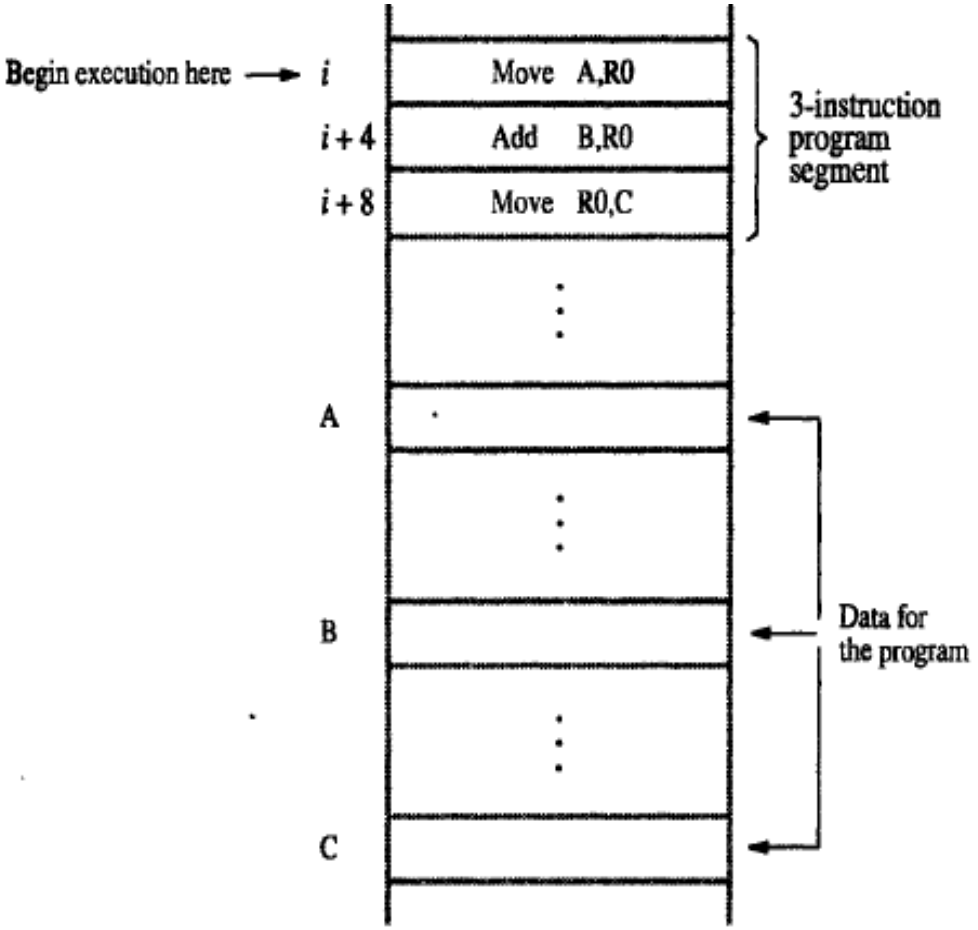


Figure 2.8 A program for C ← [A] + [B].

- To begin executing a program, the address of its first instruction (i) must be placed into the PC.
- Then, the processor control circuits use the information in the PC to fetch and execute instructions, **one at a time, in the order of increasing addresses. This is called Straight line sequencing**
- During the execution of each instruction, the PC is incremented by 4 to point to the next instruction.
- **Straight line sequencing:** means the instruction of a program is executed in **a sequential manner**(i.e. every time PC is incremented by a fixed offset). And no branch address is loaded on the PC

INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

Consider the task, $C \leftarrow [A] + [B]$

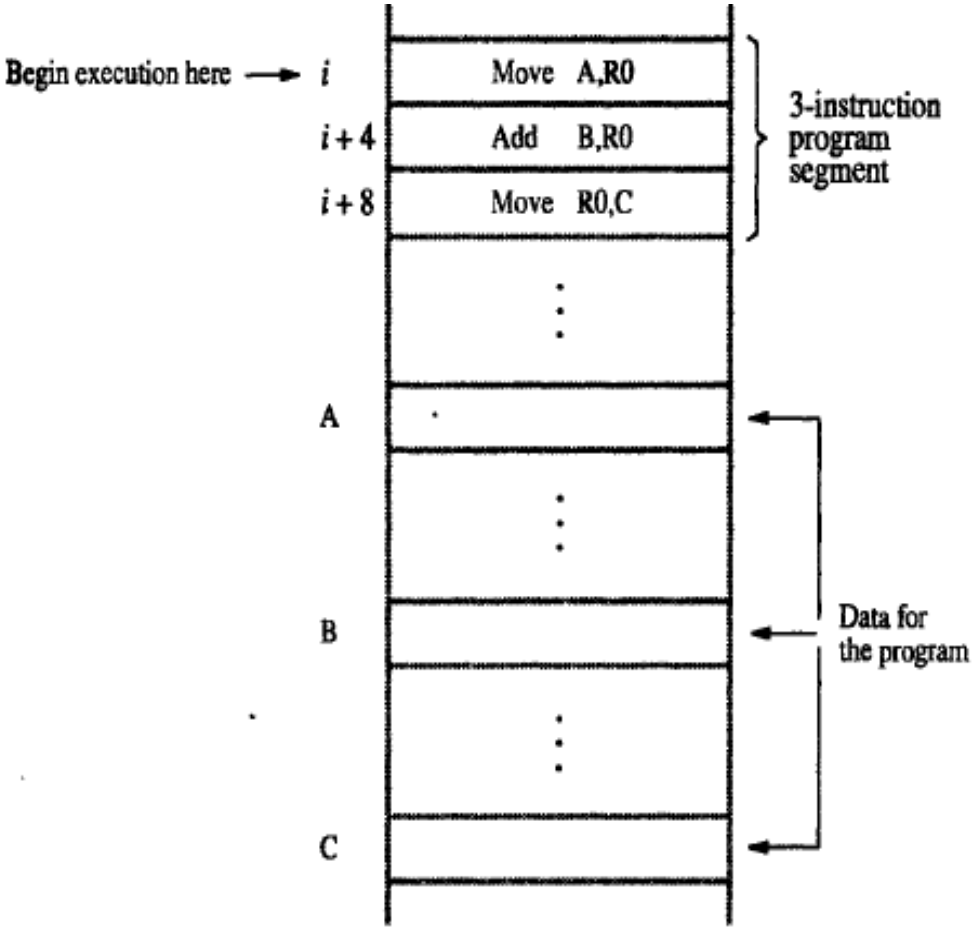


Figure 2.8 A program for C ← [A] + [B].

Executing a given instruction is a ***two-phase procedure***

Phase 1: *instruction fetch*

Registers : PC – MAR- Memory (read signal) – MDR- IR

Phase 2: *instruction execute*

- Determine which operation is to be performed (decode)
- Operation is then performed by the processor
- This often involves fetching operands from the memory or from processor registers, performing an arithmetic or logic operation, and storing the result in the destination location.
- PC - incremented

BRANCHING

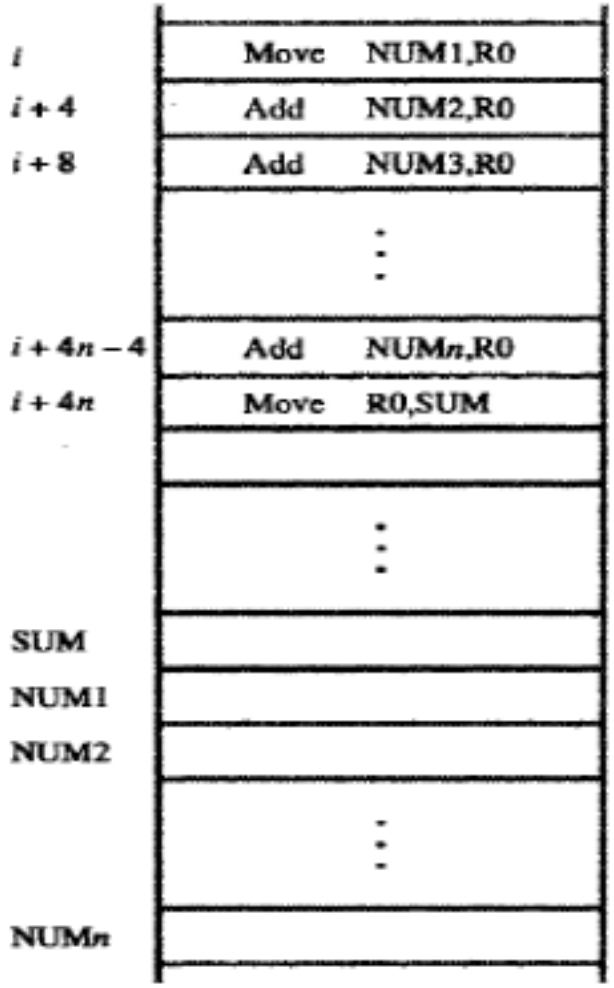


Figure 2.9 A straight-line program for adding *n* numbers.

Adding a list of *n* numbers

Addresses of the memory locations containing the *n* numbers are symbolically given as NUM1, NUM2, . . NUM*n*,

Add instruction is used to add each number to the contents of register R0. The result is placed in memory location SUM

BRANCHING

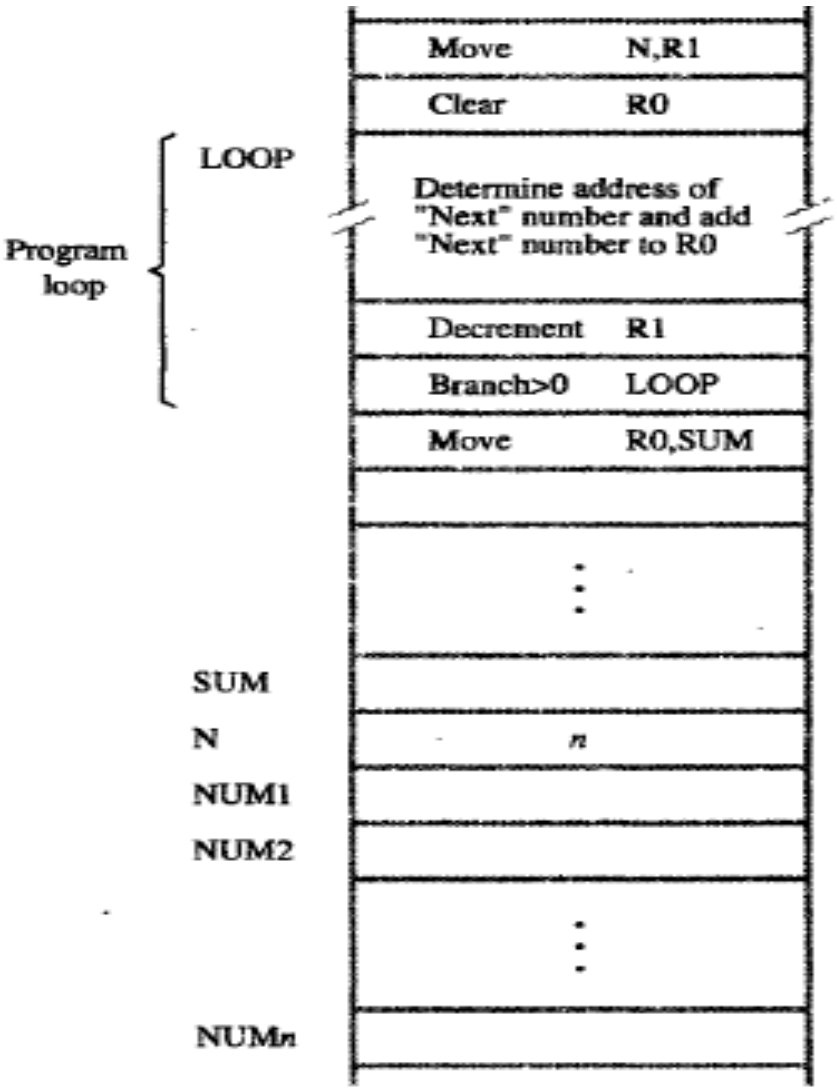


Figure 2.10 Using a loop to add *n* numbers.

- It is possible to place **a single Add instruction** in a program loop
- It starts at location LOOP and ends at the instruction Branch>0.
- During each pass through this loop, the address of the next list entry is determined, and that entry is fetched and added to R0.
  - Memory location **N - number of entries** in the list, *n*,
  - Register **R1 is used as a counter** to determine the number of times the loop is executed.
  - Hence, the contents of location N are loaded into register R1 at the beginning of the program.
  - Decrement R1 reduces the contents of R1 by 1 each time through the loop.
  - Execution of the loop is repeated **as long as the result of the decrement operation is greater than zero.**

Branch instruction

- This type of instruction loads a new value into the program counter.
- As a result, the processor fetches and executes the instruction at this new address, called the ***branch target***, instead of the instruction at the location that follows the branch instruction in sequential address order.

Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Call procedure	CALL
Return from procedure	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TEST

Conditional branch instruction

- Causes a branch only if a **specified condition is satisfied**.
- If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

Branch>0 **LOOP**

causes a branch to **location LOOP** if the result of the immediately preceding instruction, which is the decremented value in register R1, is greater than zero.

Branch Condition	Mnemonic	Test Condition
Branch if zero	BZ	$Z = 1$
Branch if not zero	BNZ	$Z = 0$
Branch if carry	BC	$C = 1$
Branch if no carry	BNC	$C = 0$
Branch if minus	BN	$N = 1$
Branch if plus	BNN	$N = 0$
Branch if overflow	BV	$V = 1$
Branch if no overflow	BNV	$V = 0$

CONDITION CODES

- Condition code flags in a processor register – **status register or condition code register**

Condition code register or status register

- Keeps track of information about the results of various operations for use by subsequent conditional branch instructions
- Condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed.

Four commonly used flags are

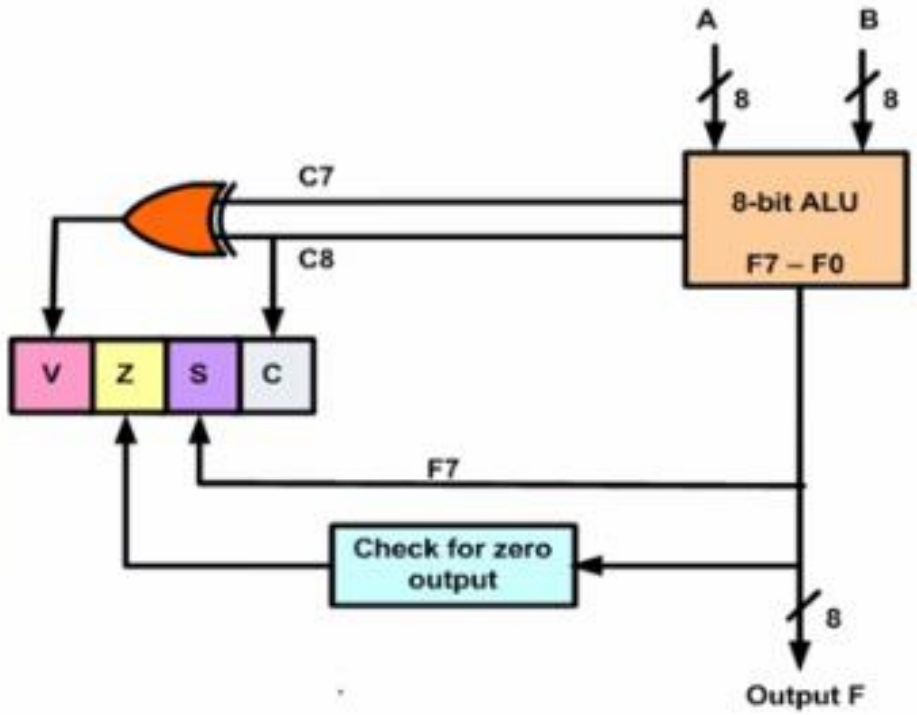
N (negative	Set to 1 if the result of an arithmetic operation is negative; otherwise, cleared to 0.
Z (zero)	Z (zero) Set to 1 if the result of an arithmetic or logic operation is 0; otherwise, cleared to 0.
V (overflow)	Set to 1 if arithmetic overflow occurs i.e., if the result of an arithmetic operation is outside the representable range (-2^n-1 to +2^n-1 -1 where n is the no. of bits used to represent the operands); otherwise, cleared to 0
C (carry)	Set to 1 if a carry occurs from the most significant bit position during an arithmetic operation.

Note

CONDITION CODES

status register

Status bit	Condition
Bit C (carry)	Set to 1 if the end carry C_8 is 1 Cleared to 0 if the end carry is 0
Bit S (sign)	Set to 1 if the highest-order bit F_7 is 1 Cleared to 0 if the bit is 0
Bit Z (zero)	Set to 1 if the output of the ALU contains all 0's Cleared to 0 otherwise
Bit V (overflow)	Set to 1 if the exclusive-OR of the last two carries is 1 Cleared to 0 otherwise



Problems

- Represent -29 using 6 bits in all three representation systems. Convert the following pair of decimal numbers to 7-bit, signed 2's-complement binary numbers and perform subtraction operation on them. Also, state whether overflow occurs or not.

Sign-and-Magnitude System: 111101

1's-complement system: 100010

2's-complement system: 100011

Subtraction of -56 and +34 using 7-bit 2's-complement system:

1001000 (-56)	1001000
- 0100010(+34)	+ 1011110
	1] 0100110 (+38)

Overflow has occurred.

Problems

- Give the significance of the four commonly used condition code flags. Also, find the status of each of these flags after the addition of the numbers -5 and -4 in a 4-bit, signed 2's-complement system.

```
   1011(-5)
+ 1100(-4)
1 0111(+7)
```

Status of condition code flags:

N = 0
Z = 0
V = 1
C = 1