**Computer Organization and Architecture (BCA-HC-4016)**

**UNIT 1: Introduction**

**Functional units of a computer**

Functional units are a part of a CPU that performs the operations and calculations called for by the computer program. Functional units of a computer system are parts of the CPU (Central Processing Unit) that performs the operations and calculations called for by the computer program. A computer consists of five main components namely, Input unit, Central Processing Unit, Memory unit Arithmetic & logical unit, Control unit and an Output unit.



Fig: - interconnection of functional units

Input unit

* Input units are used by the computer to read the data. The most commonly used input devices are keyboards, mouse, joysticks, trackballs, microphones, etc.
* However, the most well-known input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Central processing unit

* Central processing unit commonly known as CPU can be referred as an electronic circuitry within a computer that carries out the instructions given by a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

Memory unit

* The Memory unit can be referred to as the storage area in which programs are kept which are running, and that contains data needed by the running programs.
* The Memory unit can be categorized in two ways namely, primary memory and secondary memory.
* It enables a processor to access running execution applications and services that are temporarily stored in a specific memory location.
* Primary storage is the fastest memory that operates at electronic speeds. Primary memory contains a large number of semiconductor storage cells, capable of storing a bit of information. The word length of a computer is between 16-64 bits.
* It is also known as the volatile form of memory, means when the computer is shut down, anything contained in RAM is lost.
* Cache memory is also a kind of memory which is used to fetch the data very soon. They are highly coupled with the processor.
* The most common examples of primary memory are RAM and ROM.
* Secondary memory is used when a large amount of data and programs have to be stored for a long-term basis.
* It is also known as the Non-volatile memory form of memory, means the data is stored permanently irrespective of shut down.
* The most common examples of secondary memory are magnetic disks, magnetic tapes, and optical disks.

Arithmetic & logical unit

* Most of all the arithmetic and logical operations of a computer are executed in the ALU (Arithmetic and Logical Unit) of the processor. It performs arithmetic operations like addition, subtraction, multiplication, division and also the logical operations like AND, OR, NOT operations.

Control unit

* The control unit is a component of a computer's central processing unit that coordinates the operation of the processor. It tells the computer's memory, arithmetic/logic unit and input and output devices how to respond to a program's instructions.
* The control unit is also known as the nerve center of a computer system.
* Let's us consider an example of addition of two operands by the instruction given as Add LOCA, RO. This instruction adds the memory location LOCA to the operand in the register RO and places the sum in the register RO. This instruction internally performs several steps.

Output Unit

* The primary function of the output unit is to send the processed results to the user. Output devices display information in a way that the user can understand.
* Output devices are pieces of equipment that are used to generate information or any other response processed by the computer. These devices display information that has been held or generated within a computer.
* The most common example of an output device is a monitor.

**Basic instructions**:-

 Computer perform task on the basis of instruction provided. An instruction in computer comprises of groups called fields. These fields contains different information as for computers everything is in 0 and 1 so each field has different significance on the basis of which a CPU decide what so perform. The most common fields are:

* Operation field which specifies the operation to be performed like addition.
* Address field which contain the location of operand, i.e., register or memory location.
* Mode field which specifies how operand is to be founded.

An instruction is of various length depending upon the number of addresses it contain. On the basis of number of address, instruction are classified as

**Zero Address Instructions –** A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

|  |  |  |
| --- | --- | --- |
| PUSH | A | TOP = A |
| PUSH | B | TOP = B |
| ADD |  | TOP = A+B |
| PUSH | C | TOP = C |
| PUSH | D | TOP = D |
| ADD |  | TOP = C+D |

**One Address Instructions –**
This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.

|  |  |  |
| --- | --- | --- |
| LOAD | A | AC = M[A] |
| ADD | B | AC = AC + M[B] |
| STORE | T | M[T] = AC |
| LOAD | C | AC = M[C] |
| ADD | D | AC = AC + M[D] |
| MUL | T | AC = AC \* M[T] |
| STORE | X | M[X] = AC |

**Two Address Instructions –**
This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

|  |  |  |
| --- | --- | --- |
| MOV | R1, A | R1 = M[A] |
| ADD | R1, B | R1 = R1 + M[B] |
| MOV | R2, C | R2 = C |
| ADD | R2, D | R2 = R2 + D |
| MUL | R1, R2 | R1 = R1 \* R2 |
| MOV | X, R1 | M[X] = R1 |

**Three Address Instructions –**
This has three address field to specify a register or a memory location. Program created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.

|  |  |  |
| --- | --- | --- |
| ADD | R1, A, B | R1 = M[A] + M[B] |
| ADD | R2, C, D | R2 = M[C] + M[D] |
| MUL | X, R1, R2 | M[X] = R1 \* R2 |

**Addressing modes**: -

 The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. Different types of addressing modes are:

* 1. Implied Mode: - In this mode the operands are specified implicitly in the definition of the instruction.
	2. Immediately Mode: - In this mode the operand is specified in the instruction itself.
	3. Register Mode: - In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction.
	4. Register Indirect Mode: - In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
	5. Autoincrement or Autodecrement Mode: - This is similar to the register indirect mode except that the register is incremented or decremented after or before its value is used to access memory.
	6. Direct Address Mode: - In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and the address is given directly by the address field of the instruction.
	7. Indirect Address Mode: - In this mode the address field of the instruction give the address where the effective address is stored in memory.
	8. Relative Address Mode: -In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
	9. Indexed Addressing Mode: - In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
	10. Base Register Addressing Mode: - In this mode the content of the base register is added to the address part of the instruction to obtain the effective address.

**Bus structure**: -

 The computer **system bus** is the method by which data is communicated between all the internal pieces of a computer. It connects the processor to the RAM, to the hard drive, to the video processor, to the I/O drives, and to all the other components of the computer.

There are three types of buses in a computer,

1. Data Bus: - The data bus allows data to flow between devices;

2. Address Bus: - The address bus tells devices where the data should go or is coming from.

3. Control bus: The control bus coordinates activity between various devices to prevent data collisions. “Collisions” is a term that identifies the corruption of data resulting from simultaneous use of the data and/or address bus.



**Memory locations and memory addresses**,

 Memory locations and addresses determine how the computer’s memory is organized so that the user can efficiently store or retrieve information from the computer. The computer’s memory is made of a silicon chip which has millions of storage cell, where each storage cell is capable to store a bit of information which value is either 0 or 1.

 But the fact is, computer memory holds instructions and data. And a single bit is very small to hold this information so bits are rarely used individually. As a solution to this, the bits are grouped in fixed sizes of n bits. The memory of the computer is organized in such a way that the group of these n bits can be stored and retrieved easily by the computer in a single operation.

 The group of n bit is termed as **word** where n is termed as the word length. The word length of the computer has evolved from 8, 16, 24, 32 to 64 bits. General-purpose computers nowadays have 32 to 64 bits. The group of 8 bit is called a byte.

 

Now, whenever you want to store any instruction or data may it be of a byte or a word you have to access a memory location. To access the memory location either you must know the memory location by its unique name or it is required to provide a unique address to each memory location.

The memory locations are addressed from 0 to 2K-1 i.e. a memory has 2K addressable locations. And thus the address space of the computer has 2K addresses. Let us try some suitable values for K.

210 = 1024 = 1K (Kilobyte)
220 = 1,048,576 = 1M (Megabyte)
230 = 1073741824 = 1G (Gigabyte)
240 = 1.0995116e+12 = 1T (Terabyte)

**Memory operations**

 Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor. Operands and results must also be moved between the memory and the processor. Thus, two basic operations involving the memory are needed, namely, Read and Write.

• Two memory operations are:
**1) Load (Read/Fetch) &**
**2) Store (Write).**

• The Load operation transfers a copy of the contents of a specific memory-location to the processor.
The memory contents remain unchanged.

• **Steps for Load operation:**
1) Processor sends the address of the desired location to the memory.
2) Processor issues „read‟ signal to memory to fetch the data.
3) Memory reads the data stored at that address.
4) Memory sends the read data to the processor.

The Store operation transfers the information from the register to the specified memory-location. This will destroy the original contents of that memory-location.

•**Steps for Store operation are:**
1) Processor sends the address of the memory-location where it wants to store data.
2) Processor issues „write‟ signal to memory to store the data.
3) Content of register(MDR) is written into the specified memory-location.

**Instruction and instruction sequencing** (straight line sequencing and branching),

**Instruction sequencing** is the order in which the instructions in a program are carried out. Normally the sequence proceeds in a linear fashion through the program, and the address of the instructions is obtained from the program counter in the [control unit](https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/control-unit). This sequence is interrupted when a [branch instruction](https://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/branch-instruction) is executed; at such a time the address field of the branch instruction is inserted into the program counter and the process continues. In the case of an indirect branch instruction, the memory content referred to by the address field of the instruction is inserted into the program counter

**Fixed point number representation**: -

 This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field. parts of a fixed-point number representation: the sign field, integer field, and fractional field.



We can represent these numbers using:

* Signed representation: range from -(2(k-1)-1) to (2(k-1)-1), for k bits.
* 1’s complement representation: range from -(2(k-1)-1) to (2(k-1)-1), for k bits.
* 2’s complementation representation: range from -(2(k-1)) to (2(k-1)-1), for k bits.

2’s complementation representation is preferred in computer system because of unambiguous property and easier for arithmetic operations.

**Example −**Assume number is using 32-bit format which reserve 1 bit for the sign, 15 bits for the integer part and 16 bits for the fractional part.

Then, -43.625 is represented as following:



Where, 0 is used to represent + and 1 is used to represent. 000000000101011 is 15 bit binary value for decimal 43 and 1010000000000000 is 16 bit binary value for fractional 0.625

**Floating point representation of numbers: -**

 This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two part: the first part represents a signed fixed point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be fraction or an integer. Floating -point is always interpreted to represent a number in the following form: Mxre.

Only the mantissa m and the exponent e are physically represented in the register (including their sign). A floating-point binary number is represented in a similar manner except that is uses base 2 for the exponent. A floating-point number is said to be normalized if the most significant digit of the mantissa is 1.



So, actual number is (-1)s(1+m)x2(e-Bias), where s is the sign bit, m is the mantissa, e is the exponent value, and Bias is the bias number.

Note that signed integers and exponent are represented by either sign representation, or one’s complement representation, or two’s complement representation.

**Example −**Suppose number is using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. The leading bit 1 is not stored (as it is always 1 for a normalized number) and is referred to as a “hidden bit”.

Then −53.5 is normalized as  -53.5=(-110101.1)2=(-1.101011)x25 , which is represented as following below,



Where 00000101 is the 8-bit binary value of exponent value +5.

**Normalized floating point representation**: -

 A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero. For example, the decimal lnumber 350 is normalized but 00035 is not. The 8 bit binary number 00011010 is not normalized because of the three leading 0’s. the number can be normalized by shifting it three position to the lef and discarding the leading 0’s to obtain 11010000. The three shifts multiply the number by 23 = 8. To keep the same value for the floating point number, the exponent must be subtracted by 3.

 Normalized numbers provide the maximum possible precision for the floating-point number.

**Arithmetic operations using normalized floating point numbers**,

Arithmetic operations on floating point numbers consist of addition, subtraction, multiplication and division. The operations are done with algorithms similar to those used on sign magnitude integers (because of the similarity of representation) — example, only add numbers of the same sign. If the numbers are of opposite sign, must do subtraction.

IEEE standard for binary floating point representation,

,

**Stack**: -

 Stack is also known as the Last In First Out (LIFO) list. It is the most important feature in the CPU. It saves data such that the element stored last is retrieved first. A stack is a memory unit with an address register. This register influence the address for the stack, which is known as Stack Pointer (SP). The stack pointer continually influences the address of the element that is located at the top of the stack.

It can insert an element into or delete an element from the stack. The insertion operation is known as push operation and the deletion operation is known as pop operation. In a computer stack, these operations are simulated by incrementing or decrementing the SP register.

**Register Stack**

The stack can be arranged as a set of memory words or registers. Consider a 64-word register stack arranged as displayed in the figure. The stack pointer register includes a binary number, which is the address of the element present at the top of the stack. The three-element A, B, and C are located in the stack.

The element C is at the top of the stack and the stack pointer holds the address of C that is 3. The top element is popped from the stack through reading memory word at address 3 and decrementing the stack pointer by 1. Then, B is at the top of the stack and the SP holds the address of B that is 2. It can insert a new word, the stack is pushed by incrementing the stack pointer by 1 and inserting a word in that incremented location.



The stack pointer includes 6 bits, because 26 = 64, and the SP cannot exceed 63 (111111 in binary). After all, if 63 is incremented by 1, therefore the result is 0(111111 + 1 = 1000000). SP holds only the six least significant bits. If 000000 is decremented by 1 thus the result is 111111.

Therefore, when the stack is full, the one-bit register ‘FULL’ is set to 1. If the stack is null, then the one-bit register ‘EMTY’ is set to 1. The data register DR holds the binary information which is composed into or readout of the stack.

First, the SP is set to 0, EMTY is set to 1, and FULL is set to 0. Now, as the stack is not full (FULL = 0), a new element is inserted using the push operation.

The push operation is executed as follows –

|  |  |
| --- | --- |
| SP←SP + 1 | It can increment stack pointer |
| K[SP] ← DR | It can write element on top of the stack |
| If (SP = 0) then (FULL ← 1) | Check if stack is full |
| EMTY ← 0 | Mark the stack not empty |

The stack pointer is incremented by 1 and the address of the next higher word is saved in the SP. The word from DR is inserted into the stack using the memory write operation. The first element is saved at address 1 and the final element is saved at address 0. If the stack pointer is at 0, then the stack is full and ‘FULL’ is set to 1. This is the condition when the SP was in location 63 and after incrementing SP, the final element is saved at address 0. During an element is saved at address 0, there are no more empty registers in the stack. The stack is full and the ‘EMTY’ is set to 0.

A new element is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation includes the following sequence of micro-operations −

|  |  |
| --- | --- |
| DR←K[SP] | It can read an element from the top of the stack |
| SP ← SP – 1 | It can decrement the stack pointer |
| If (SP = 0) then (EMTY ← 1) | Check if stack is empty |
| FULL ← 0 | Mark the stack not full |

The top element from the stack is read and transfer to DR and thus the stack pointer is decremented. If the stack pointer reaches 0, then the stack is empty and ‘EMTY’ is set to 1. This is the condition when the element in location 1 is read out and the SP is decremented by 1.

Subroutine: -

 A set of common instructions that can be used in a program many times is called a subroutine. Each time that a subroutine is used in the main part of the program, a branch is executed to the beginning of the subroutine. After the subroutine has been executed, a branch is made back to the main program.

 A subroutine consists of a self-contained sequence of instructions that carries out a given task. A branch can be made to the subroutine from any part the main program.

 In the basic computer, the link between the main program and a subroutine is the BSA instruction (branch and save return address)

I/O instructions

**UNIT 2: Register Transfer Logic**

Introduction,

 inter-register transfer,

**Arithmetic micro-operation**: -

 In general, the Arithmetic Micro-operations deals with the operations performed on numeric data stored in the registers.

The basic Arithmetic Micro-operations are classified in the following categories:

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Shift

Some additional Arithmetic Micro-operations are classified as:

1. Add with carry
2. Subtract with borrow
3. Transfer/Load, etc.

The following table shows the symbolic representation of various Arithmetic Micro-operations.

|  |  |
| --- | --- |
| **Symbolic Representation** | **Description** |
| R3 ← R1 + R2 | The contents of R1 plus R2 are transferred to R3. |
| R3 ← R1 - R2 | The contents of R1 minus R2 are transferred to R3. |
| R2 ← R2' | Complement the contents of R2 (1's complement) |
| R2 ← R2' + 1 | 2's complement the contents of R2 (negate) |
| R3 ← R1 + R2' + 1 | R1 plus the 2's complement of R2 (subtraction) |
| R1 ← R1 + 1 | Increment the contents of R1 by one |
| R1 ← R1 - 1 | Decrement the contents of R1 by one |

**Logic micro-operation**: -

 Logic Microoperations Logic Microoperations specify binary operations performed for strings of bits in registers. These operations consider each bit of the register separately and treat them as binary variables.

* 

**Shift micro-operation**: -

 Shift micro-operations are used for the serial transfer of data and also support in conjunction with arithmetic, logic, and several data-processing operations. The contents of a register can be shifted to the left or the right.

Simultaneously, that the bits are shifted, the first flip-flop holds its binary data from the serial input. In shift-left operation, the serial input transfers a bit into the rightmost position. In shift-right operation, the serial input transfers a bit into the leftmost position. The data transferred by the serial input decides the type of shift.

##  Types of Shift Micro-operations

There are three types of shift micro-operations are as follows −

**Logical Shift**

It transfers 0 by the serial input. The symbol **"shl"** can be used for logical shift left and **"shr"** can be used for logical shift right.

R1 ←R1 shl R1

R2 ←R1 shr R2

The register symbol should be the equivalent on both sides of the arrows.

**Circular Shift**

This circulates or pivots the bits of register around the two ends without any trouble of data or contents. In circular shift, the serial output of the shift register is linked to its serial input. **"cil"** and **"cir"** is used for circular shift left and right respectively. The symbolic documentation for the shift micro-operations is demonstrated in the table.

**Shift Micro-operations**

| **Symbolic Designation** | **Description** |
| --- | --- |
| R ←R1 shl R | Shift-left register R |
| R ←R1 shr R | Shift-right register R |
| R ←R1 cil R | Circular shift-left register R |
| R ←R1 cir R | Circular shift-right register R |
| R ←R1 ashl R | Arithmetic shift-left R |
| R ←R1 ashr R | Arithmetic shift-right R |

**Arithmetic Shift**

This shifts a signed binary number to left or right. An arithmetic shift left multiplies a signed binary number by 2 and shift left divides the number by 2. Arithmetic shift micro-operation leaves the sign bit constant due to the signed number remains equal when it is multiplied or divided by 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rn-1 | Rn-2 | R→ | R1 | R0 |

Sign bit

**Arithmetic Shift Right**

The leftmost bit in a register influences the sign bit, and the remaining bits influence the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form. The figure displays a symbolic register of n bits.

Bit Rn-1 in the leftmost position holds the sign bit.

Rn-2 is the most significant bit of the number and

R0 is the least significant bit. The arithmetic shift-right leaves the sign bit constant and shifts the number (involving the sign bit) to the right. Therefore

Rn-1 remains the equal,

Rn-2 receives the bit from

Rn-1 and so on for the other bits in the register. The bit in

R0 is lost.

Conditional control statements,

fixed point binary data,

 instruction code,

design of a simple computer.

**UNIT 3: Processor Logic Design**

Processor organization,

design of arithmetic and logic circuit,

status register,

design of accumulator.

**UNIT 4: Control Logic Design**

Hardware control,

**Micro-programmed control: -**

 A control unit whose binary control values are saved as words in memory is called a microprogrammed control unit.

A controller results in the instructions to be implemented by constructing a definite collection of signals at each system clock beat. Each of these output signals generates one micro-operation including register transfer. Thus, the sets of control signals are generated definite micro-operations that can be saved in the memory.

Each bit that forms the microinstruction is linked to one control signal. When the bit is set, the control signal is active. When it is cleared the control signal turns inactive. These microinstructions in a sequence can be saved in the internal ’control’ memory. The control unit of a microprogram-controlled computer is a computer inside a computer.

The following image shows the block diagram of a Microprogrammed Control organization.

 

**Advantages of Microprogrammed Control Unit**

There are the following advantages of microprogrammed control are as follows −

* It can more systematic design of the control unit.
* It is simpler to debug and change.
* It can retain the underlying structure of the control function.
* It can make the design of the control unit much simpler. Hence, it is inexpensive and less error-prone.
* It can orderly and systematic design process.
* It is used to control functions implemented in software and not hardware.
* It is more flexible.
* It is used to complex function is carried out easily.

**Disadvantages of Microprogrammed Control Unit**

There are the following disadvantages of microprogrammed control are as follows −

* Adaptability is obtained at more cost.
* It is slower than a hardwired control unit.

Symbolic micro-program,

micro-programmed CPU organization

UNIT 5: I/O Subsystem

Program controlled I/O,

Interrupts: enabling and disabling interrupts,

handling interrupts from multiple sources (priority control),

DMA.

UNIT 6: Memory Subsystem

Semiconductor memory,

SRAM,

DRAM,

ROM,

speed size and cost,

Cache memory,

mapping functions