Name : Farzana Chowdhury jangmi

Id : 2119160021

Ans to the qus no. 1 a

Each assembly language statement is split into an opcode and an operand. The opcode is the instruction that is executed by the CPU and the operand is the data or memory location used to execute that instruction. An x86 instruction can have zero to three operands. Operands are separated by commas (,) (ASCII 0x2C). For instructions with two operands, the first (lefthand) operand is the source operand, and the second (righthand) operand is the destination operand (that is, source->destination).

The table below is a small assembly language program with a description of the opcode and operand components during execution.

| **Original assembly language** | **Opcode** | **Operand** | **Description** |
| --- | --- | --- | --- |
| INP | INP |  | Input value and store in the accumulator |
| STA 1C | STA | 1C | Store the number at memory address 1C |
| INP | INP |  | Input value and store in the accumulator |
| ADD 1C | ADD | 1C | Add this number to the number stored at memory address 1C |
| OUT | OUT |  | Output the result |
| HLT | HLT |  | Stop the program |

Some operands are:

Expressions as operands

Order of operations

Positioning of operands

Infix and the order of operation

Ans to the qus no. 1 b

High-level language

It can be easily interpreted as well as compiled in comparison to low-level language.

It can be considered as a programmer-friendly language.

It is easy to understand.

It is easy to debug.

It is simple in terms of maintenance.

It requires a compiler/interpreter to be translated into machine code.

It can be run on different platforms.

It can be ported from one location to another.

It is less memory efficient, i.e it consumes more memory in comparison to low-level languages.

Examples of high level languages include C, C++, Java, Python.

It is used widely in today's times.

Low-level language

It is also known as machine level language.

It can be understood easily by the machine.

It is considered as a machine-friendly language.

It is difficult to understand.

It is difficult to debug.

Its maintenance is also complex.

It is not portable.

It depends on the machine; hence it can't be run on different platforms.

It requires an assembler that would translate instructions.

It is not used widely in today's times.

The differences between High-Level and Low-Level programming languages −

| **High-Level Language** | **Low-level language** |
| --- | --- |
| It can be considered as a programmer-friendly language. | It is considered as a machine-friendly language. |
| It requires a compiler/interpreter to be translated into machine code. | It requires an assembler that would translate instructions. |
| It can be ported from one location to another. | It is not portable. |
| It is easy to understand. | It is difficult to understand. |
| It is easy to debug. | It is difficult to debug. |
| It is less memory efficient, i.e., it consumes more memory in comparison to low-level languages. | It consumes less memory. |

Ans to the qus no 2 a

Direct Memory Access (DMA) :

DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor. DMA controller needs the same old circuits of an interface to communicate with the CPU and Input/Output devices.

Typical examples are disk controllers, Ethernet controllers, USB controllers, and video controllers. Usually the DMA controller built into these devices can only move data between the device itself and main memory – that is, it's not intended to be used as a general system DMA controller.

Ans to the qus no. 2 b

An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands in computer instruction words. In some processors, the ALU is divided into two units: an arithmetic unit (AU) and a logic unit (LU).

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

The characteristics of ALU?

The ALU performs simple addition, subtraction, multiplication, division, and logic operations, such as OR and AND. The memory stores the program's instructions and data.

Ans to the qus no 2 c

In numeral system, we know octodecimal is base-18 and binary is base-2. To convert octodecimal 5G.AB216 to binary, you follow these steps:

To do this, first convert octodecimal into decimal, then the resulting decimal into binary

1. Start from one's place in octodecimal : multiply ones place with 18^0, tens place with 18^1, hundreds place with 18^2 and so on from right to left
2. Add all the products we got from step 1 to get the decimal equivalent of given octodecimal value.
3. Then, divide decimal value we got from step-2 by 2 keeping notice of the quotient and the remainder.
4. Continue dividing the quotient by 2 until you get a quotient of zero.
5. Then just write out the remainders in the reverse order to get binary equivalent of decimal number.

First, convert 5G.AB21618 into decimal, by using above steps:

= 5G18  
= 5 × 181G × 180A × 18-1B × 18-22 × 18-31 × 18-46 × 18-5  
= 106.5898618096834810

Now, we have to convert 106.5898618096834810 to binary

106 / 2 = 53 with remainder 0  
53 / 2 = 26 with remainder 1  
26 / 2 = 13 with remainder 0  
13 / 2 = 6 with remainder 1  
6 / 2 = 3 with remainder 0  
3 / 2 = 1 with remainder 1  
1 / 2 = 0 with remainder 1

106 = 1101010 ------- (1)

For converting decimal fraction 0.58986180968348 to binary number, follow these steps:

1. Multiply 0.58986180968348 by 2 keeping notice of the resulting integer and fractional part. Continue multiplying by 2 until you get a resulting fractional part equal to zero (we calcuclate upto ten digits).
2. Then just write out the integer parts from the results of each multiplication to get equivalent binary number.

0.58986180968348 × 2 = 1 + 0.17972361936696  
0.17972361936696 × 2 = 0 + 0.35944723873392  
0.35944723873392 × 2 = 0 + 0.71889447746784  
0.71889447746784 × 2 = 1 + 0.43778895493568  
0.43778895493568 × 2 = 0 + 0.87557790987136  
0.87557790987136 × 2 = 1 + 0.75115581974272  
0.75115581974272 × 2 = 1 + 0.50231163948544  
0.50231163948544 × 2 = 1 + 0.0046232789708824  
0.0046232789708824 × 2 = 0 + 0.0092465579417649  
0.0092465579417649 × 2 = 0 + 0.01849311588353

0.58986180968348 = 0.1001011100 ------- (2)

106.5898618096834810 = 1101010.100101110000002

Therefore, octodecimal number 5G.AB216 converted to binary is equals:  
  
**1101010.10010111000000**

Ans to the qus no 3 a

Today, it is typical to use small amounts of assembly language code within larger systems implemented in a higher-level language, for performance reasons or to interact directly with hardware in ways unsupported by the higher-level language.

Assembly Usage #1 – To Execute a Breakpoint Instruction

Assembly Usage #2 – Transition from Bootloader to Application

Assembly Usage #3 – Code Optimization in a Control Loop

Assembly Usage #4 – To Teach Microcontroller Fundamentals

Assembly language is used to directly manipulate hardware, access specialized processor instructions, or evaluate critical performance issues. These languages are also used to leverage their speed advantage over high level languages for time-sensitive activities such as high frequency trading.

Ans to the qus no 3 b

Input and output, or I/O is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it.

Each microprocessor provides instructions for I/O with the devices that are attached to it, e.g. the keyboard and screen.

The 8086 provides the instructions in for input and out for output. These instructions are quite complicated to use, so we usually use the operating system to do I/O for us instead. The operating system provides a range of I/O subprograms, in much the same way as there is an extensive library of subprograms available to the C programmer. In C, to perform an I/O operation, we call a subprogram using its name to indicate its operations, e.g. putchar(), printf(), getchar(). In addition we may pass a parameter to the subprogram, for example the character to be displayed by putchar() is passed as a parameter e.g. putchar(c). In assembly language we must have a mechanism to call the operating system to carry out I/O. In addition we must be able to tell the operating system what kind of I/O operation we wish to carry out, e.g. to read a character from the keyboard, to display a character or string on the screen or to do

disk I/O.

Finally, we must have a means of passing parameters to the

operating subprogram.

Ans to the qus no 3 c

In Computer Organisation, the register is utilized to acknowledge, store, move information and directions that are being utilized quickly by the CPU. There are different kinds of registers utilized for different reasons. Some of the commonly used registers are:

AC ( accumulator )

DR ( Data registers )

AR ( Address registers )

PC ( Program counter )

MDR ( Memory data registers )

IR ( index registers )

MBR ( Memory buffer registers )

These registers are utilized for playing out the different operations. When we perform some operations, the CPU utilizes these registers to perform the operations. When we provide input to the system for a certain operation, the provided information or the input gets stored in the registers. Once the ALU arithmetic and logical unit process the output, the processed data is again provided to us by the registers.

The sole reason for having a register is the quick recovery of information that the CPU will later process. The CPU can use RAM over the hard disk to retrieve the memory, which is comparatively a much faster option, but the speed retrieved from RAM is still not enough. Therefore, we have catch memory, which is faster than registers. These registers work with CPU memory like catch and RAM to complete the task quickly.

Types of Register in Computer Organization

Here are the following types of registers in computer organization, such as:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **NAME** | **SYMBOL** | **FUNCTIONING** |
| 1 | Accumulator | AC | An accumulator is the most often utilized register, and it is used to store information taken from memory. |
| 2 | Memory address registers | MAR | Address location of memory is stored in this register to be accessed later. It is called by both MAR and MDR together |
| 3 | Memory data registers | MDR | All the information that is supposed to be written or the information that is supposed to be read from a certain memory address is stored here |
| 4 | General-purpose register | GPR | Consist of a series of registers generally starting from R0 and running till Rn - 1. These registers tend to store any form of temporary data that is sent to a register during any undertaking process. More GPR enables the register to register addressing, which increases processing speed. |
| 5 | Program counter | PC | These registers are utilized in keeping the record of a program that is being executed or under execution. These registers consist of the memory address of the next instruction to be fetched. PC points to the address of the next instruction to be fetched from the main memory when the previous instruction has been completed successfully. Program Counter (PC) also functions to count the number of instructions. The incrementation of PC depends on the type of architecture being used. If we use a 32-bit architecture, the PC gets incremented by 4 every time to fetch the next instruction. |
| 6 | Instructions registers | IR | Instruction registers hold the information about to be executed. The immediate instructions received from the system are fetched and stored in these registers. Once the instructions are stored in registers, the processor starts executing the set instructions, and the PC will point to the next instructions to be executed |
| 7 | Condition code registers |  | These have different flags that depict the status of operations. These registers set the flags accordingly if the result of operation caused zero or negative |
| 8 | Temporary registers | TR | Holds temporary data |
| 9 | Input registers | INPR | Carries input character |
| 10 | Output registers | OUTR | Carries output character |
| 11 | Index registers | BX | We use this register to store values and numbers included in the address information and transform them into effective addresses. These are also called base registers. These are used to change operand address at the time of execution, also stated as BX |
| 12 | Memory buffer register | MBR | MBR - Memory buffer registers are used to store data content or memory commands used to write on the disk. The basic functionality of these is to save called data from memory. MBR is very similar to MDR |
| 13 | Stack control registers | SCR | Stack is a set of location memory where data is stored and retrieved in a certain order. Also called last in first out ( LIFO ), we can only retrieve a stack at the second position only after retrieving out the first one, and stack control registers are mainly used to manage the stacks in the computer. SP - BP is stack control registers. Also, we can use DI, SI, SP, and BP as 2 byte or 4-byte registers. EDI, ESI, ESP, and EBP are 4 - byte registers |
| 14 | Flag register | FR | Flag registers are used to indicate a particular condition. The size of the registered flag is 1 - 2 bytes, and each registered flag is furthermore compounded into 8 bits. Each registered flag defines a condition or a flag. The data that is stored is split into 8 separate bits. Basic flag registers - Zero flags Carry flag Parity flag Sign flag Overflow flag. |
| 15 | Segment register | SR | Hold address for memory |
| 16 | Data register | DX | Hold memory operand |