Chapter 3

INSTRUCTION SET

Overview

- Instruction Set and Instruction Formats
- Instruction Categories
- Types of Addressing Modes

3.1 Preamble

Programs consist of meaningful sequences of instructions, each of which specifies some specific action to be done. The operation code or opcode, which is part of the instruction, indicates the particular action to be performed. The locations of data used in instructions are also specified in certain instructions. Addressing specifies where the operands or their addresses are located and how to access them. Let us now go deep into the topics.

Instruction:
Instruction, in a computer, specifies an operation to be carried out and the set of operands or data to be used for that purpose. Operands include the input data or arguments of the operation and the results that are produced.

Instruction Set:
Instruction set is the set of instructions that a machine is able to execute. Each particular machine has its own set of instructions i.e. an instruction set, which consists of all the instructions used in that particular computer, varies from computer to computer depending on the specific organization and architecture of the computer. Design of an instruction set for a particular machine however depends on the designer of that machine.

3.2 Instruction Format

Instruction format deals with the looks of a basic instruction. Each instruction has three parts. The 'opcode' part, the 'addressing mode' part and the operands or the 'address' (i.e. operand address) part. The operation field is called the 'opcode' or the 'operation code'. The operand/address fields contain either the operands themselves or the addresses of storage locations of the data or arguments (i.e. addresses of operands) in main memory or in the processor depending on the various addressing modes (I) as specified in a particular instruction.
So, a basic computer instruction format will be like:

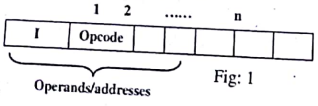


Fig: 1

Format of an instruction can have an opcode, an addressing mode but a set of operands or operand addresses (1, 2...., n).

Opcode:
Opcode is the operation code that indicates the specific operation to be carried out by the instruction. The operation field of an instruction is called the opcode.
Operands:
Operands are the set of data to be used to carry out the operations. Operand fields contain the addresses of storage locations of the arguments to be used in the operation.
Operand Address:
Each operation specified by computer instructions are executed on some data stored in memory or processor registers. These data or operands are specified either by memory address or by register address. So operand addresses are the addresses of storage locations of the operands (i.e. set of arguments or data) to be used in the operation to be performed. Generally the operands are stored in different memory locations or in different CPU registers.
If suppose there are 'i' number of memory locations (i = 1, 2, ...), then an operand residing in memory location 4 (say) will be specified by an address 4 (100). Similarly for a processor, a register address is a binary number of 'k' bits that defines one of the 2^k registers in the CPU.
[***Note:*** Operand addresses are always specified in binary]

Example:
Suppose a CPU has 16 processor registers, R0 through R15. Register address fields are of 4 bits. So, an operand in register R5 will be specified by an address of 0101. So to retrieve or access the operands stored in the specific locations the addresses of those locations must be given.

3.3 Types of Instruction Code Formats

The basic computer has three instruction code formats:
(i) Memory-reference instruction:
Instructions that need reference to a memory location (i.e. reference to a memory register is needed).

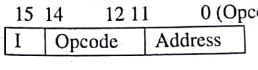


Fig: 2

So here 'Address' field = 12 bits (0-11); 'Opcode' field = 3 bits (12-14); 'Addressing Mode' field = I = 1 bit (15); for direct address, I = 0 and for indirect address, I = 1 (i.e. if the addressing mode field = 0, it means a direct addressing mode and if it is 1, it means an indirect addressing mode). The 'Opcode' field will vary from 000 through 110.

Example:
LOAD 1001 ⇒ The content of memory location 1001 is loaded to the accumulator

(ii) Register-reference instruction:
Instructions, which reference a CPU register for the purpose of addressing. An operand from memory (i.e. no memory reference) is not needed.

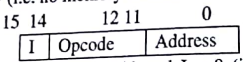


Fig: 3

Here the opcode is always 111 and I = 0 (i.e. the addressing mode field will always indicate a 0). So, the format will look like:

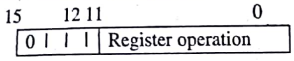


Fig: 4

The 12 bits (0-11) are used to specify the required operand addresses in the CPU registers and the 3 bits (12 – 14) specifies the required operation to be performed.

Example:
ADD_{Direct} B ⇒ The content of CPU register B is added to the accumulator.

(iii) Input-output instruction:
Instructions, which reference an I/O register for the purpose of addressing. Here also no memory references are needed.
Just like the previous type of instruction, opcode of such instructions are always 111 and I = 1. So, the format will look like:

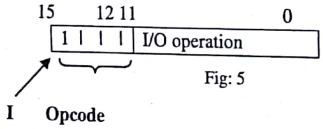


Fig: 5

The 12 bits (0-11) specifies the required operand address and the 3 bits (12–14) specifies the type of input-output operation to be performed.
Example:
IN A, 2010 ⇒ The I/O unit may have numerous registers in it. The content of register 2010 is to be loaded inside the accumulator.

3.4 Differentiation of the Basic Instruction Format

An instruction type can be recognized by the computer from the four bits in position 12 through 15 of the instruction.

(i) Memory-reference instruction:
Opcode field (12-14 bits) → 000 through 110.
I = 0 indicates a direct addressing mode
I = 1 indicates an indirect addressing mode
(ii) Register-reference instruction:
Opcode field (12-14 bits) → 111
I = 0 always.
(iii) Input-output instruction:
Opcode field (12-14 bits) → 111
I = 1 always.
So only in case of memory-reference instructions, the opcode field is not equal to 111, which is equal to 111 for the other two.
If I = 0, it is a register-reference instruction and if I = 1, it is a input-output instruction provided that the opcode field is always 111.

3.5 Different Categories of Instructions

The different instruction categories are the following:
(i) Arithmetic, logical and shift instructions:
The instructions of this category deal with all sorts of computational capabilities for arithmetic, logical and shifting operations.
Example: MUL B

(ii) Instructions for moving information (data) to and from memory and processor registers:
As most computer information is stored in memory, they must be fetched from there and brought to the processor registers, where all the computations are done. So the above type of instructions move the information between the memory and the processor registers.
Example: $ADD_{Memory\ Indirect}$ B

(iii) Program control and status checking instructions:
Program control instructions (like branch instruction) change the program execution sequences.
Example: JMP NZ (Jump if not zero)
Status checking instructions monitors the status of the flag for a condition and checks whether the condition is true or false. If true, some operation is performed and if false, some other operations are performed.

(iv) Input-output instructions:

Such instructions are needed for communication between the computer and the user through input-output devices.

Example:

INP (input) and **OUT** (output) instructions transfer information between the computer and external devices.

3.6 Three, Two, One and Zero Address Instructions

An instruction may have multiple operands and hence multiple operand addresses. These addresses can be explicit as well as implicit (i.e. implied). So based on the number of explicit operand addresses (i.e. operands as well as their addresses are to be defined specifically within an instruction), computer instructions are classified accordingly as:

(i) Three-address instructions:

In these type of instructions, all operand addresses are explicitly defined. Here the instruction format has three different address fields specifying a memory or a processor register operand.

Example:

explicit operands

ADD R1, C, B ⇒ R1 ← M[C] + M[B] i.e. the operands at memory addresses C and B are added and the resultant sum is stored at a third processor register R1.

Advantages:

(i) It results in short programs when evaluating arithmetic expressions.
(ii) Less execution time.

Disadvantages:

(i) Too many bits are required for binary- coded instructions to specify three addresses.
(ii) More complex decoding and processing circuits are needed.

Uses:

Cyber 170 is a commercial computer using three-address instructions.

(ii) Two-address instructions:

Here the instruction format has two different address fields, each specifying either a memory or a processor register operand.

Example:

ADD X, Y ⇒ AC ← M[X] + M[Y]
(Explicit operands)
or
M[X] ← M[X] + M[Y]
or
M[Y] ← M[X] + M[Y]

i.e. the operands at memory addresses X and Y are added and the resultant sum may be stored in either the accumulator (AC), or either of the first M[X] or the second operand location M[Y].

So, in this example, two explicit operand address fields (X and Y) are specified whereas the third operand address field is implicit and it may be any one of AC or M[X] or M[Y]. The choice of the implicit operand depends on the designer of the computer.

Advantages:

(i) Less execution time compare to one-address instructions.
(ii) Number of instructions needed to evaluate a arithmetic expression is much less compared to one-address instructions.

Disadvantages:

(i) It results in comparatively longer programs, than three-address instructions, when evaluating arithmetic expressions.
(ii) Comparatively more execution time than two- address instructions.

Uses:

Two-address instructions are used in all commercial computers.

(iii) One-address instructions:

Such instruction format has a single explicit address field and uses an implied accumulator (AC) register for all data manipulation.

Example:

ADD X ⇒ AC ← AC + M[X]

i.e. the operand at memory location X is added to the accumulator content and the result is stored in the accumulator itself. So there are only one explicit operand address field (i.e. X) and one implicit operand address.

Advantages:

(i) Much less number of bits is required to specify the single operand address.
(ii) Less complicated decoding and processing circuit is needed.

Disadvantages:

(i) Number of instructions needed to evaluate an arithmetic expression is much more compared to two and three address instructions.
(ii) More execution time needed compare to two and three address instructions.
(iii) Limited range of functions each instruction can perform.

Uses:

In Intel 8085 machines.

(iv) Zero-address Instructions:

Such instructions do not contain any explicit addresses (except for PUSH and POP instructions). As the operands are stored in a pushdown stack (the operands required must be there in the top positions in the stack), hence no addresses are required.

Example:

ADD ⇒ TOS ← X + Y

Here the top two operands X and Y in the stack are removed from the stack and added. The result is then again placed at the top of the stack (TOS).

Advantages:

(i) Do not contain any explicit addresses. So instructions are simple.

Disadvantages:

(i) To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expressions into Reverse Polish Notation (RPN).

Uses:

In all stack-type computers.

(***Note:*** Fewer the addresses, shorter the instruction, but also limited range of functions each instruction can perform).

3.7 Addressing Modes

Each operation field of an instruction specifies the operation to be performed. The operand field specifies the data (or operand) on which the operations must be executed. These data are stored in the registers or memory locations. So while executing an instruction, the processor needs the specific data from their storage locations.

The way, the data is specified or the way the operands are chosen by the processor (as instructed by the programmer) during program execution gives the ***addressing mode*** of the instruction.

Effective Address:

Effective address means the actual address of the operand.

It is defined to be the memory address obtained from the computation dictated by the given addressing mode. Irrespective of the addressing mode, address of the exact location (generally in the memory) where the actual operand is located, is the effective address of the operand.

Example:

In figure 10, the effective address is X (as X is the address of the location where the actual operand 99 is stored in the memory).

3.8 Different Types of Addressing Modes

The different addressing modes are the following:

(i) Implied Mode:

In this mode the operands are specified 'implicitly' in the definition of the instruction i.e. the operands are implied instruction definition. Operands need not be specified explicitly.

Example:

Complement Accumulator: The above instruction is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. The operand need not be specified explicitly.

Uses:

All register reference instructions using an accumulator and zero-address instructions in a stack-organized computer are implied-mode instructions.

(ii) Immediate Addressing Mode:

In this mode the operand is itself specified in the instruction operand field. Here the operand value is a constant.

Example:

LOADI 99 or $LOAD_{Immediate}$ 99

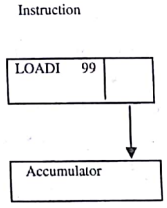


Fig: 6

This means that 99 (i.e. the data or the operand) are to be loaded in the Accumulator as has been shown in the figure. In this mode the instruction has an operand field rather than an address field.

Uses:

Immediate addressing mode instructions are useful for initializing registers to a constant value.

The Implied and the Immediate addressing modes do not need any operand address fields in them. Here the ***operands are themselves used*** (i.e. operands, not their addresses, are used).

The remaining addressing modes need the ***operand addresses*** (and not the operands themselves). Now as the operands may be stored either in the memory or in the processor register, the address field of an instruction may specify either a memory word or a processor register.

In the following addressing modes, the address field specifies a processor register and hence the instruction is said to be in the register mode.

(iii) Register Mode or Register Direct Mode:

In this mode the operands reside in registers that reside within the CPU i.e. the operands reside directly in the CPU registers.

Example:

$ADD_{Register\ direct}$ B

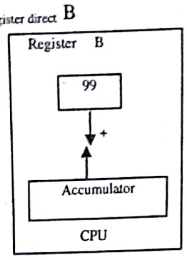


Fig: 7

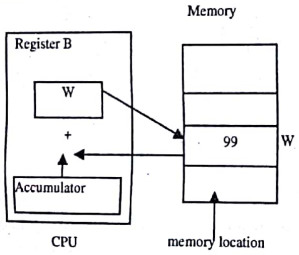
Here B is a CPU register and the content of which is 99 i.e. this content reside directly within the CPU register B (means the address of this operand is the CPU register B). Hence, this content must get added to the content of the accumulator.

(iv) Register Indirect Mode:

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory i.e. the content of the CPU register is the address of the operand's location in the memory.

Example:

$ADD_{Register\ Indirect}$ B



Here the content of the CPU register B (i.e. W) is the memory location that contains the operand 99. So the register B contains the address of the operand 99. This operand (99) is to be added to the accumulator content.

Fig: 8

Uses and Advantages:

(A) The advantages of such modes is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

(B) Also as such addressing modes either do not require or require very less number of memory references (in compared to memory direct and memory indirect addressing modes) hence the instruction executions are comparatively faster.

So for these two reasons, register addressing modes are convenient to use as the execution time needed is much less.

(v) Autoincrement Mode (vi) Autodecrement Mode:

These two addressing modes are similar to the register indirect mode except that the register is incremented or decremented after or before its value is used to access memory i.e. the content of the register (given in the instruction) is incremented or decremented accordingly and then that memory location (i.e. the resultant memory location) is looked upon for the actual operand.

When the address stored in the register refers to table of data in memory, it is necessary to increment or decrement the register after every access to the table. The increment or decrement instruction helps to achieve this.

Example:

$ADD_{Autoincrement\ /\ Autodecrement}$ B

Suppose the CPU register B has in it the memory location address 2010. Let the 1st memory location has this address 2010 containing the operand 99. So this operand will be added to the accumulator content. Again if there is a table of memory locations starting from address 2010 (and increasing downwards), then just incrementing the register content by 1(2010+1=2011), the operand in 2011 location can be accessed and added to new accumulator content. This may continue until the operands in all the memory locations in the table are added to the previous accumulator content.

If however memory is increasing in descending order, then the register content needs to be decremented by 1 each time.

Uses:

Useful in accessing a table of data in memory.

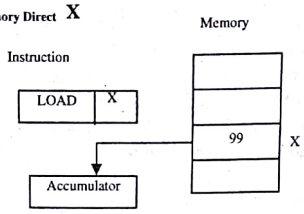
In the following addressing modes, the address field specifies a memory location and hence the instruction is said to be in the memory mode.

(vi) Direct Address Mode:

In this mode, the effective address of the operand is equal to the address part of the instruction i.e. the address part of the instruction indicates the memory location containing the operand.

Example:

LOAD$_{\text{Memory Direct}}$ **X**



In the example, the memory location X contains the operand 99 to be loaded in the accumulator.

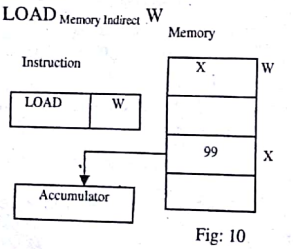
Fig: 9

(vii) Indirect Address Mode:

In this mode the address field of the instruction gives the address where the effective address is stored in memory i.e. the address part of the instruction indicates the memory location whose content is the address of the memory location containing the actual operand.

Example:

LOAD $_{\text{Memory Indirect}}$ W



In the example, the memory location W contains X, which is the memory location whose content is the actual operand 99. This operand is to be ultimately loaded in the accumulator.

Fig: 10

The remaining addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The actual operand address (i.e. the effective address) in these modes is obtained from the following computation: effective address = address part of instruction + content of CPU register. The CPU register used in the computation may be the program counter, an index register, or a base register. Depending on type of CPU register, the addressing modes are also different. Such addressing modes are called:

(viii) Relative Address Mode:

In such modes the content of the CPU register is added to the address part of the instruction to obtain the actual address of the operand (i.e. the effective address). The address part of the instruction, which is usually a signed number (either positive or

negative) on addition to the CPU register content, gives the effective address whose position in memory is relative to the address of the next instruction.

Uses:

(A) Relative addressing mode is often used with branch-type instructions.

(B) This mode also results in shorter address field in the instruction format as the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

Relative addressing modes may be of the following three types:

(a) PC (i.e. Program Counter) Relative Addressing Mode:

In this mode the content of the program counter is added to the address part of the instruction to obtain the actual address of the operand (i.e. the effective address).

Example:

ADD$_{\text{PC-Relative}}$ 25

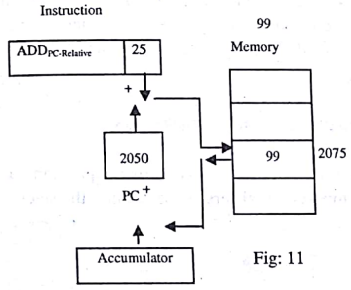


Fig: 11

In this example, the address part of the instruction i.e. 25 is added to the content of the PC (i.e. 2050) and the memory location (i.e. 2050 + 25 = 2075) is looked upon and its content (i.e. 99) is added to the content of the accumulator and the result after addition is stored in the accumulator itself.

Uses:

Same as in relative address mode.

(b) Indexed Addressing Mode or Index Register Relative Addressing Mode:

In this mode the content of the index register is added to the address part of the instruction to obtain the actual address of the operand (i.e. the effective address).

Example:

ADD$_{\text{Index Relative}}$ 25

Diagram and explanation of the example is same as the PC-Relative Mode only that the CPU register is now an index register instead of the PC.

Uses:

The index register can be used to access consecutive operands. This can be done by incrementing the register contents.

(c) 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 actual address of the operand (i.e. the effective address).

Example:

$ADD_{Base\ Register\ Relative}$ 25

Diagram and explanation of the example is same as the PC-Relative Mode only that the CPU register is now base register instead of the PC.

Uses:

Such modes are used in computers to facilitate the relocation of programs in memory.

The last addressing mode is the ***stack-addressing*** mode, which is described as follows:

(ix) Stack Addressing Mode:

Here the address of the operand is specified by the stack pointer (SP). The length of instruction is the shortest as it does not include any address of the memory location or mention any register (just like implied mode of addressing). After each stack operation, the contents of SP are automatically incremented or decremented. PUSH and POP are the two commonly used instructions of this type.

Example:

PUSH A ⇒ To push the content of accumulator to the top of stack (TOS).

Uses:

(a) Useful when PUSH and POP instructions are used in a program by the programmer.

(b) When interrupt occurs the contents of important registers are saved into the stack. For this stack addressing is used.

Related Questions & Answers

Question 1

Which factors help in deciding the length of instruction formats?

Answer:

The factors are the number of explicit operands and hence the number of explicit operand addresses, range of functions each instruction can perform, the nature of processing circuits and storage requirements of each instruction.

Generally the fewer the addresses, the shorter the instruction, less storage requirements in the memory. But fewer addresses mean longer programs are required to perform any given task and thus longer execution times, whereas long instructions with multiple addresses require more complex decoding and processing circuits.

Question 2

Describe briefly different addressing modes with suitable examples. **[WBUT 2003, 2004, 2005]**

Answer: Refer to sections 3.7 and 3.8.

Question 3

Explain the difference between one-address instruction & zero address instructions with suitable examples. **[WBUT 2003]**

Answer: Refer to section 3.6.

Question 4

What are implicit and explicit operand addresses?

Answer:

Operand addresses may be implicit (i.e. implied) or explicit (i.e. not implied). Generally operations performed with an accumulator register has implicit operand addresses i.e. all operations that have reference to the accumulator register have implicit operands as well as implicit operand addresses.

Example:

MOV B ⇒ the content of register B is to be moved in the accumulator. So here the accumulator (A) is implied. This instruction may also be written as MOV A, B. But generally it is not written like this.

All operations that do not have references to the accumulator register generally have ***explicit*** operands as well as explicit operand addresses.

Example:

MUL X, C, B ⇒ to multiply the contents of registers B and C and to store the result in the register X.

Question 5

Show the execution of X = A * B + C * D in three, two and one address machines.

Answer:

Three-address machine:

```
MULTIPLY  T, A, B     ⇒   T ← A * B
MULTIPLY  X, C, D     ⇒   X ← C * D
ADD       X, X, T     ⇒   X ← X + T
```

Two-address machine:

```
MOVE T, A          ⇒   T ← A
MULTIPLY T, B      ⇒   T ← T * B
MOVE X, C          ⇒   X ← C
MULTIPLY X, D      ⇒   X ← X * D
ADD X, T           ⇒   X ← X + T
```

One-address machine:

```
LOAD A       ⇒ transfer certain memory content to accumulator
MULTIPLY B   ⇒ AC ← AC * B
STORE T      ⇒ transfer AC content to memory location T
LOAD C       ⇒ transfer C to accumulator
MULTIPLY D   ⇒ AC ← AC * D
ADD T        ⇒ AC ← AC + T
STORE X      ⇒ transfer result to memory location X
```


Question 6

Give the instruction code format and define opcode. **[WBUT 2004]**

Answer: Refer to section 3.2.

Question 7

What are the requirements to be satisfied by an instruction set?

Answer:

These are the following:

(a) It should be ***complete*** i.e. machine language program required to evaluate any computable function can be constructed using the instructions in the set.

(b) It should be ***efficient*** i.e. frequently required functions can be performed using few instructions.

(c) It should be ***regular*** i.e. all expected opcodes and addressing modes should be there.

(d) It should be ***compatible*** with the hardware and software of the existing machines in order to reduce the design costs.

Question 8

Differentiate direct and indirect instructions and also differentiate between register reference and input-output reference instructions. **[WBUT 2004]**

Answer: Refer to section 3.8.

Question 9

What is an offset or displacement?

Answer:

Consider figure 11. There the PC contains address of the memory location 2050 and the address part of the instruction contains 25. So the actual operand 99 is located in the memory location 2075. Hence in this example 25 is the ***offset*** or ***displacement***, which indicates the difference between the ultimate memory location containing the actual operand (i.e. 2075) and the address of the memory location as stored in any register (which is not the ultimate address). So adding the offset or displacement value to the address contained in the register gives the address of the ultimate memory location containing the actual operand.

Question 10

How many references to memory are needed for direct and indirect addressing modes to bring an operand into processor register?

Answer:

In ***direct addressing mode***, only one reference to memory is needed to bring in an operand into the processor register. This is because the address of the memory location provided in the instruction is the actual operand address. Consider figure 9.

Two references to memory are needed to fetch the required operand in case of ***indirect addressing mode***. Consider figure 10. Here the address of the memory location provided in the instruction acts as a pointer to the ultimate memory location address, which contains the actual operand. So as shown in the figure, in the first memory reference X is to be fetched and in the second memory reference 99 can be fetched.

Question 11

Why does adding different type of addressing modes to an ISA tend to improve performance of a computer? **[WBUT 2005]**

Answer:

The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instruction.

Question 12

What are the general rules that should be followed by a computer designer while designing effective instruction sets?

Answer:

A computer designer should always aim of designing a machine that will prove effective in terms of both cost and performance. Hence, designing effective instruction sets is a major factor. The general rules of designing an effective performance-friendly instruction set are as follows:

a. The length of all instructions must be the same so that the interpretive logic can be very simple and the entire instruction can be easily fetched in a single clock cycle.
b. All instructions should execute in a single and short clock cycle.

c. Instructions must not have any 'side effects', i.e. either the instructions should execute completely performing the entire operation they are meant to do or the instructions should do nothing and should not change the status of registers and flags. In other words, there should not be any instructions, which may 'fail to complete' the task assigned to it.
d. Addressing formats and modes should be simple.

Question 13

Explain the difference between three-address, two-address, one-address instructions & zero-address instruction with suitable examples. **[WBUT 2007]**

Answer: Refer to section 3.6.

Question 14

Show the execution of X = (A–B) / (C+D*E) in three, two and one address machines.

Answer:

Three-address machine:

SUBTRACT:	SUB X, A, B	⇒ X ← A - B
MULTIPLY:	MUL Z, C, E	⇒ Z ← C * E
ADD:	ADD Z, Z, C	⇒ Z ← Z + C
DIVIDE:	DIV X, X, Z	⇒ X ← X / Z

Two-address machine:

MOVE X, A	⇒	X ← A
SUB X, B	⇒	X ← X – B
MOVE Z, D	⇒	Z ← D
MUL Z, E	⇒	Z ← Z * E
ADD Z, C	⇒	Z ← Z + C
DIV X, Z	⇒	X ← X / Z

One-address machine:

LOAD D	⇒	AC ← D
MUL E	⇒	AC ← AC * E
ADD C	⇒	AC ← AC + C
STOR X	⇒	X ← AC
LOAD A	⇒	AC ← A

SUB B ⇒ AC ← AC - B
DIV X ⇒ AC ← AC / X
STOR X ⇒ X ← AC

Question 15

Given an example and explain Base-Index Addressing. [WBUT 2007]

Answer: Refer to sections 3.8.

Question 16

"An ADD instruction in a stack-based instruction set architecture requires less number of bits to get encoded (i.e. smaller) in comparison to a ADD instruction in a general-purpose instruction set architecture." Do you agree with this statement? Explain showing reasons.

Answer:

Yes, I agree with the statement. The ADD instruction in the stack-based instruction set architecture does not require any type of explicit operands. Hence, here, the machine code does not include any bit that may represent the operands to add.

On the other hand, in a general-purpose instruction set architecture, it is required that in each instruction, all register operands should be specified clearly. Hence, for 16 possible registers, in such kind of instruction set, to clearly specify each register, the ADD instruction should include four bits. So, it can be said that the ADD instruction in a stack-based instruction set architecture is smaller than that in a general-purpose instruction set architecture.

Question 17

Explain, why instructions with short lengths are always more preferable than long-length instructions.

Answer:

a. Shorter instructions generally result in faster execution of programs. This is because, rate of delivery of instructions to the central processing unit is a very basic but important limitation on the instruction execution speed. So, lengthier the instruction, lengthier is the instruction execution speed i.e. more time taken to execute the instructions and thus the program.
b. Smaller instructions would require less memory storage space for programs and would thus lower the overall hardware cost, as a small memory would do.

c. Instructions between the memory and the central processing unit are transferred by means of buses having a fixed simultaneous data transfer capacity. Hence, it is always needed that the instruction length should be shorter than the maximum carrying capacity of the bus.

Question 18

Most modern day computers use a stack data structure to track the way different procedures are activated. In such a stack data structure describe the responsibilities of the calling function and the called function.

Answer:

Responsibilities of the calling function or the caller:

a. Stores the different argument onto the stack.
b. Makes or allocates adequate space for output values.
c. Calls or activates the required procedure.

Responsibilities of the called function or the callee:

a. Stores the stack pointer and the return address.
b. Retrieves the arguments passed.
c. Successfully saves the result.
d. On completion, again retrieves the stack pointer and passes the control to the caller.

Question 19

Write short notes on 'Addressing modes'. [WBUT 2007, 2008]

Answer: Refer to sections 3.7 and 3.8.

Question 20

What are advantages and disadvantages of a variable length instruction set over a fixed length instruction set?

Answer:

Advantages:

1. Compared to a fixed length instruction set, a variable length instruction set offers better cache utilization/throughput.
2. Because of more flexible operand fields, it is easily possible to include longer immediate fields.

Disadvantages:

1. Decoding an instruction is harder.
2. As chances of instructions crossing the cache-line boundaries are there, it is quite possible that some part of the instruction produces a hit and remaining produce misses.

Question 21

Differentiate between the two instructions: ADD 42 and ADD-C 42.

Answer:

Both the instructions instruct the CPU to add a number to the value, which is already stored in the accumulator and then put the resulting sum back into the accumulator. However, for the ADD 42 instruction, the number that is to be added is stored in the memory location number 42. For the ADD-C 42 instruction, the number to be added is the number 42 itself.

Question 22

Classify Instruction Set Architectures.

Answer:

An Instruction Set Architecture (ISA), which is an interface between the hardware and software in a computer, can be of four different types depending on the different types of internal storage in a processor. These are: Stack, Accumulator, Register-Memory and Register-Register. The table below provides a comparison of the Stack, Accumulator and Register-oriented machines.

Type of Machine	Pros	Cons
Stack	1. Effective Address is simple 2. Code density is fairly good 3. Instruction decoding is simple 4. Instructions are short	1. Difficult to generate efficient codes 2. Lack of random access features
Accumulator	1. Instructions are short 2. Instruction decoding is simple 3. Context switching is fast 4. Internal state is minimal	1. Memory traffic is substantially high

Type of Machine	Pros	Cons
Register	1. Code and code generations are efficient 2. Multiple different code generation options.	1. Instructions are longer 2. Multiple options exist for size and structure of register sets, which makes the whole thing a bit complicated to handle 3. Effective address generation is complex

Question 23

Sketch the instruction format of a two-address instruction that uses immediate, register direct and indexed addressing mode if size of the memory is 1 MB and size of instruction word is limited to 16 bits with 3-bit opcode field.

Answer:

The desired instruction format has to be designed under the following conditions:

1. The instruction must be 16 bits long.
2. The instruction should be two-address instruction where the second operand will be added to / subtracted from the first operand which will also store the result.
3. The instruction should have 3 addressing modes, namely, immediate, reg-direct and indexed. Thus a 2-bit addressing field should follow the 3-bit op-code field.
4. Since reg-direct addressing is needed but memory-direct addressing is not needed, there must be a set of registers some of which will be special purpose (SP) in nature and others general purpose.

   Since 11 bits are remaining, we assume that the machine has 16 byte registers, which may also be used as 8 register pairs (RP). Thus the instruction format has 5 fields, namely, op-code (3), addressing mode (2), RP (3), R1 (4) and R2 (4).

Question 24

Evaluate the arithmetic statement $X=(A*B)/(C+D)$ ***in one, two and three address machines.*** **[WBUT 2008]**

Answer:

Three-address machine:

```
MULTIPLY  T, A, B   ⇒   T ← A * B
ADD       X, C, D   ⇒   X ← C + D
DIV       X, T, X   ⇒   X ← T / X
```

Two-address machine:

```
MOVE      T, A   ⇒   T ← A
MULTIPLY  T, B   ⇒   T ← T * B
MOVE      X, C   ⇒   X ← C
ADD       X, D   ⇒   X ← X + D
DIV       T, X   ⇒   X ← T / X
```

One-address machine:

```
LOAD A      ⇒ transfer certain memory content to accumulator
MULTIPLY B  ⇒ AC ← AC * B
STORE T     ⇒ transfer AC content to memory location T
LOAD C      ⇒ transfer C to accumulator
ADD D       ⇒ AC ← AC + D
DIV T       ⇒ AC ← AC * T
STORE X     ⇒ transfer result to memory location
```

Question 25

Define instruction set and instruction set architecture. **[WBUT 2005]**

Answer:

Instruction set:

Refer to sections 3.1.

Instruction Set Architecture:

Refer to question number 22.

Question 26

Evaluate the arithmetic statement F = (A + B) * (C + D) in one, two and three address machines.

Answer:

Three-address machine:

```
ADD       T, A, B   ⇒   T ← A + B
ADD       X, C, D   ⇒   X ← C + D
MULTIPLY  X, X, T   ⇒   X ← X * T
```

Two-address machine:

```
MOVE      T, A   ⇒   T ← A
ADD       T, B   ⇒   T ← T + B
MOVE      X, C   ⇒   X ← C
ADD       X, D   ⇒   X ← X + D
MULTIPLY  X, T   ⇒   X ← X * T
```

One-address machine:

```
LOAD A      ⇒ transfer certain memory content to accumulator
ADD B       ⇒ AC ← AC + B
STORE T     ⇒ transfer AC content to memory location T
LOAD C      ⇒ transfer C to accumulator
ADD D       ⇒ AC ← AC + D
MULTIPLY T  ⇒ AC ← AC * T
STORE X     ⇒ transfer result to memory location X
```

Question 27

Compare and contrast RISC and CISC architecture. **[WBUT 2009]**

Answer:

RISC	CISC
i) Multiple register sets, often consisting of more than 256 registers.	i) Single register set, typically 6 to 16 registers total.
ii) Three register operands allowed per instruction (e.g. → add R_1, R_2, R_3)	ii) One or two register operands allowed per instruction (e.g. → add R_1, R_2)
iii) Parameter passing through efficient on chip register windows.	iii) Parameter passing through inefficient off-chip memory.
iv) Single cycle instructions (except for load and store).	iv) Multiple cycle instruction.
v) Hardwired control.	v) Micro-programmed control.
vi) Highly pipelined.	vi) Less pipelined.
vii) Simple instructions that are few in	vii) Many complex instructions.

RISC	CISC
number.	
viii) Fixed length instructions.	viii) Variable length instructions.
ix) Complexity in compiler.	ix) Complexity in microcode.
x) Only load and store instructions can access memory.	x) Many instructions can access memory.
xi) Few addressing modes.	xi) Many addressing modes.

Question 28

Evaluate the arithmetic statement X = (A + B) * C in one, two and three address machines.

Answer:

Three-address machine:

ADD T, A, B ⇒ T← A + B
MULTIPLY X, C, T ⇒ X ← C * T

Two-address machine:

MOVE T, A ⇒ T ← A
ADD T, B ⇒ T← T + B
MULTIPLY C, T ⇒ X ← C * T

One-address machine:

LOAD A ⇒ transfer certain memory content to accumulator
ADD B ⇒ AC ← AC + B
STORE T ⇒ transfer AC content to memory location T
LOAD C ⇒ transfer C to accumulator
MULTIPLY T ⇒ AC← AC * T
STORE X ⇒ transfer result to memory location X

Question 29

Explain the instruction format of a two-address instruction that uses immediate, register direct and indexed addressing mode if size of the memory is 1 MB and size of instruction word is limited to 16 bits with 3-bit opcode field.

Answer:

The desired instruction format has to be designed under the following conditions:

1. The instruction must be 16 bits long.
2. The instruction should be two-address instruction where the second operand will be added to / subtracted from the first operand which will also store the result.

3. The instruction should have 3 addressing modes, namely, immediate, reg-direct and indexed. Thus a 2-bit addressing field should follow the 3-bit op-code field.
4. Since reg-direct addressing is needed but memory-direct addressing is not needed, there must be a set of registers some of which will be special purpose (SP) in nature and others general purpose.
   Since 11 bits are remaining, we assume that the machine has 16 byte registers, which may also be used as 8 register pairs (RP). Thus the instruction format has 5 fields, namely, op-code (3), addressing mode (2), RP (3), R1 (4) and R2 (4).

Question 30

Explain base index addressing with example. **[WBUT 2011]**

Answer:

In Base-Index addressing mode, the effective address is the sum of the contents of the specific base register and that of the specific index register. For example, such an addressing mode could be useful in accessing elements of an array. In this case, the base register would hold the starting location of the specified array and the index register would contain the location of the offset.

Question 31

What are the advantages of relative addressing mode over direct address mode? **[WBUT 2011]**

Answer:

Unlike in the direct addressing mode, it is possible to write "position-independent code" using relative addressing modes. "Position-independent code" implies programs that are properly executed by the processor irrespective of their locations in the memory. The location of the whole program and its related data may easily be tracked and moved from its current position to a new position (may be in a complete different memory location without any adverse effect. Such a location-independent program may not refer to an specific location by address; however, all references to memory must be made throug the use of relative addressing.

Question 32

Classify MRI and non-MRI instructions **[WBUT 2011]**

Answer:
MRI → Memory-Reference Instructions and non-MRI → any non memory-reference instructions like register-reference instructions or input-output instructions. Refer to Sections 3.3 and 3.4 for details.

Question 33

Evaluate the following arithmetic expression using 0, 1, 2, 3 address instruction:
$X = (A + B) / (C * D)$

Answer:
Three-address machine:

ADD	T, A, B	⇒	T ← A + B
MULTIPLY	X, C, D	⇒	X ← C * D
DIV	X, T, X	⇒	X ← T / X

Two-address machine:

MOVE	T, A	⇒	T ← A
ADD	T, B	⇒	T ← T + B
MOVE	X, C	⇒	X ← C
MULTIPLY	X, D	⇒	X ← X * D
DIV	T, X	⇒	X ← T / X

One-address machine:

LOAD A	⇒	transfer certain memory content to accumulator
ADD B	⇒	AC ← AC + B
STORE T	⇒	transfer AC content to memory location T
LOAD C	⇒	transfer C to accumulator
MULTIPLY D	⇒	AC ← AC * D
DIV T	⇒	AC ← AC / T
STORE X	⇒	transfer result to memory location X

Short Questions & Answers

1. What is a pointer?
Answer:
Pointer is the register or memory location that contains the address of an operand. For example in figure 8, register B contains the address W of the operand 99 in the memory. So register B in the figure acts as a pointer.

2. What are orthogonal architectures?
Answer:
Orthogonal architectures are such architectures that allow any instruction referencing memory to choose and use any addressing mode independently without depending on the choice of instruction.

3. What is the implied operand in an one address instruction set design when performing an operation like add or multiply or compare that requires two operands?
Answer:
It is the accumulator.

4. In a zero address instruction set, where do the operands come from?
Answer:
The operands come from the stack.

5. What is an Instruction Set Architecture?
Answer:
This is an interface between the hardware and software of a computer.

Multiple Choice Type Questions

1. *Instruction cycle is* [WBUT 2006, 2007]
   a) *fetch-decode-execution* b) *decode-fetch-execution*
   c) *fetch-execution-decode* d) *none of these*

Answer: (a)

2. *A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (opcode) an address part (allowing for only one address). Each instruction is stored in one word of memory. Bits are needed for opcode.* [WBUT 2007]
   a) 6 b) 7 c) 8 d) 9

Answer: (c)

3. *Micro instructions are kept in* [WBUT 2007]
   a) *Main memory* b) *Control memory*
   c) *Cache memory* d) *None of these*

Answer: (b)

4. *Which of the following addressing modes is used in the instruction PUSH B?*
   a) *Immediate* b) *Register*
   c) *Direct* d) *Register Indirect* [WBUT 2008]

Answer: (b)

Exercise

1. What is an instruction format?
2. What are 0, 1, 2 and 3 address machines?
3. What do you mean by addressing modes?
4. What are the different addressing modes? Explain each of them with suitable example.
5. Give a brief comparison among the different addressing modes.
6. Give the basic features of memory-reference instructions and register-reference instructions.
7. What factors help in deciding the length of instruction format?
8. What is direct and indirect address? Explain with examples.
9. How an effective address is calculated?
10. Explain the difference between one-address and zero-address instructions with suitable examples.
11. Give the instruction code format and define opcode.
12. Differentiate direct and indirect instructions.
13. Differentiate between register reference and input – output reference instructions.