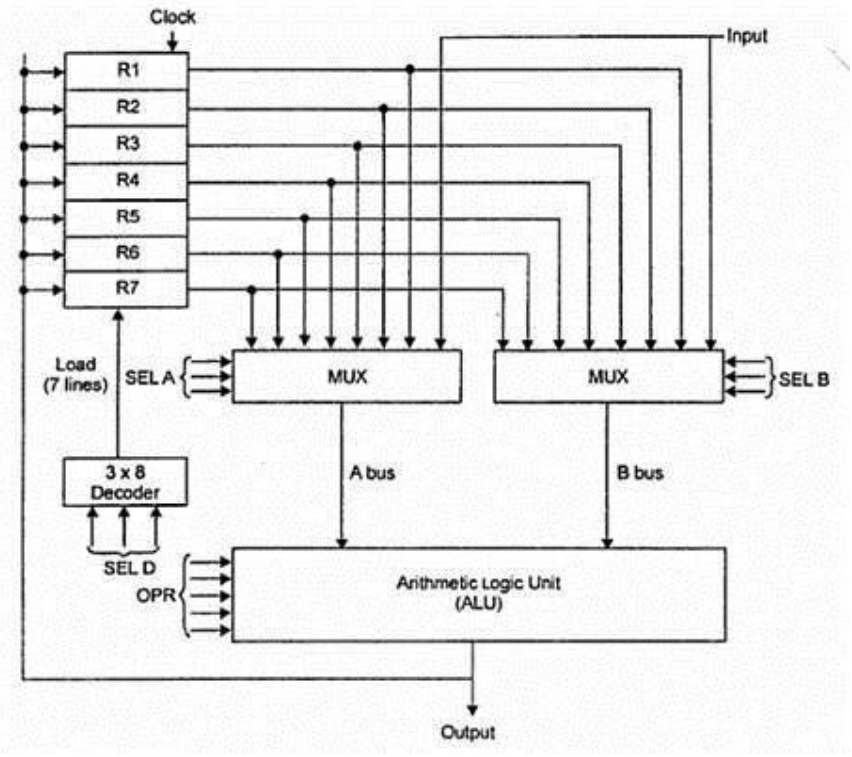
31.	Explain about General Register Organization.	8	Section-4	4

Ans.

General Register organization

- Generally CPU has seven general registers. Register organization show how registers are selected and how data flow between register and ALU. A decoder is used to select a 3 particular register. The output of each register is connected to two multiplexers to form the two buses A and B. The selection lines in each multiplexer select the input data for the particular bus.
- The A and B buses form the two inputs of an ALU. The operation select lines decide the micro operation to be performed by ALU. The result of the micro operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.



A bus organization for seven CPU registers

EXAMPLE:

To perform the operation **R3 = R1+R2** we have to provide following binary selection variable to the select inputs.

1. **SEL A: 001** -To place the contents of R1 into bus A.
2. **SEL B: 010** - to place the contents of R2 into bus B
3. **SEL OPR: 10010** – to perform the arithmetic addition A+B
4. **SEL REG or SEL D: 011** – to place the result available on output bus in R3.

Binary code	SELA	SELB	SELD or SELREG
000	Input	Input	---
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Register and multiplexer input selection code

Operation selection code	Operation	symbol
0000	Transfer A	TSFA
0001	Increment A	INC A
0010	A+B	ADD
0011	A-B	SUB
0100	Decrement A	DEC
0101	A AND B	AND
0110	A OR B	OR
0111	A XOR B	XOR
1000	Complement A	COMA
1001	Shift right A	SHR
1010	Shift left A	SHL

Operation with symbol

What is CONTROL WORD?

- The combined value of a binary selection inputs specifies the control word.
- It consists of four fields SELA, SELB, and SELD or SELREG contains three bit each and SELOPR field contains four bits thus the total bits in the control word are 13-bits.

SEL A	SELB	SELREG OR SELD	SELOPR

Format of control word

1. The three bit of SELA select a source registers of the input of the ALU.
2. The three bits of SELB select a source registers of the b input of the ALU.
3. The three bits of SELED or SELREG select a destination register using the decoder.
4. The four bits of SELOPR select the operation to be performed by ALU

CONTROL WORD FOR OPERATION R2 = R1+R3

SEL A	SEL B	SEL D OR SELREG	SELOPR
001	011	010	0010

Note: Control words for all micro operation are stored in the control memory

Example:

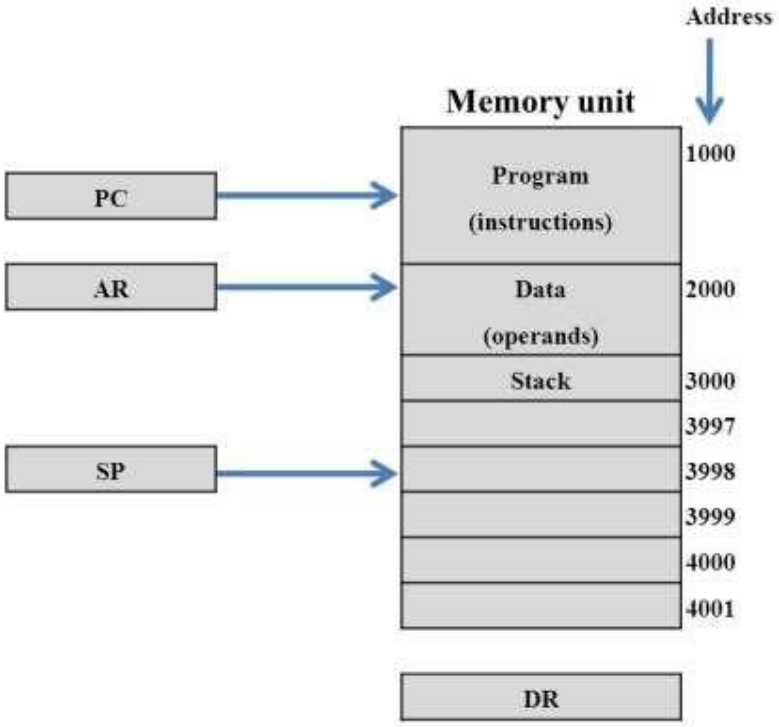
MICROOPERATION	SEL A	SEL B	SEL D OR SELREG	SELOPR	CONTROL WORD			
					00	01	01	001
R2 = R1+R3	R1	R3	R2	ADD	1	1	0	0

32.	Demonstrate memory stack organization.	8	Section-4	4

Ans.

Memory Stack

- In the above discussion a stack can exist as a stand-alone unit. But in the CPU implementation of a stack is done by assigning a portion of memory to a stack operation and using a processor register as stack pointer.
- The below figure shows a portion computer memory partitioned into three segments: program, data, and stack.



- The program counter PC points at the address of the next instruction in program.
- The address register AR points at an array of data.
- The stack pointer SP points at the top of the stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory. o PC is used during the fetch phase to read an instruction. o AR is used during the exec phase to read an operand. o SP is used to push or pop items into or from stack.
- As shown in Fig, the initial value of SP is 4001 and the stack grows with decreasing addresses.
- Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000.
- No provisions are available for stack limit checks.
- The items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows:

  SP← SP-1

  M [SP] ← DR

- The stack pointer is decremented so that it points at the address of the next word. ¬ A memory write operation inserts the word from DR into the top of stack. A new item is deleted with a pop operation as follows:

  DR ← M [SP]
  SP← SP+1

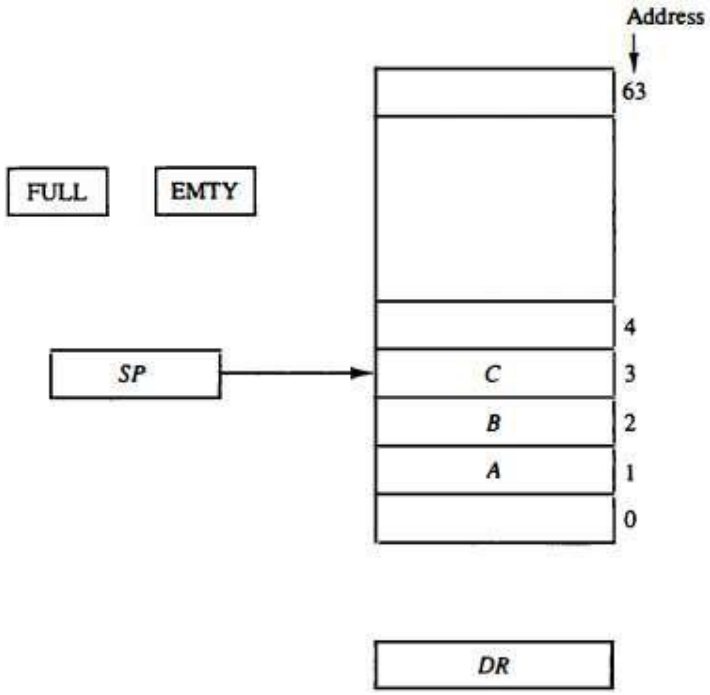
- The top item is read from the stack into DR. The stack pointer is then decremented to point at the next item in the stack.
- Most computers do not provide hardware to check for stack overflow (full stack) or underflow (empty stack).
- The stack limits can be checked by using processor registers: o one to hold the upper limit (3000 in this case) o Other to hold the lower limit (4001 in this case).
- After a push operation, SP compared with the upper-limit register and after a pop operation, SP is a compared with the lower-limit register.
- The two microoperations needed for either the push or pop are (1) An access to memory through SP (2) Updating SP.
- The advantage of a memory stack is that the CPU can refer to it without having specify an address, since the address is always available and automatically updated in the stack pointer.

33.	Demonstrate register stack organization.	8	Section-4	4

Ans.

Register Stack

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.
- The below figure shows the organization of a 64-word register stack.



- The stack pointer register SP contains a binary number whose value is equal to the address of the word is currently on top of the stack. Three items are placed in the stack: A, B, C, in that order.
- In above figure C is on top of the stack so that the content of SP is 3.
- For removing the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of stack SP.
- Now the top of the stack is B, so that the content of SP is 2.
- Similarly for inserting the new item, the stack is pushed by incrementing SP and writing a word in the next higher location in the stack.
- In a 64-word stack, the stack pointer contains 6 bits because 26 = 64.
- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary).
- When 63 is incremented by 1, the result is 0 since 111111 + 1. = 1000000 in binary, but SP can accommodate only the six least significant bits.
- Then the one-bit register FULL is set to 1, when the stack is full.
- Similarly when 000000 is decremented by 1, the result is 111111, and then the one-bit register EMTY is set 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.

Push

- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full.
- If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

- The push operation is implemented with the following sequence of microoperations:

$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack
If $(SP = 0)$ then $(FULL \leftarrow 1)$	Check if stack is full
$EMTY \leftarrow 0$	Mark the stack not empty

- The stack pointer is incremented so that it points to the address of next-higher word.
- A memory write operation inserts the word from DR the top of the stack.
- The first item stored in the stack is at address 1.
- The last item is stored at address 0.
- If SP reaches 0, the stack is full of items, so FULL is to 1.
- This condition is reached if the top item prior to the last push way location 63 and, after incrementing SP, the last item is stored in location 0.
- Once an item is stored in location 0, there are no more empty registers in the stack, so the EMTY is cleared to 0.

Pop

- A new item is deleted from the stack if the stack is not empty (if EMTY = 0).
- The pop operation consists of the following sequence of min operations:

$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

- The top item is read from the stack into DR.
- The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set 1.
- This condition is reached if the item read was in location 1. Once this it is read out, SP is decremented and reaches the value 0, which is the initial value of SP.
- If a pop operation reads the item from location 0 and then is decremented, SP changes to 111111, which is equivalent to decimal 63 in above configuration, the word in address 0 receives the last item in the stack.

34.	Define several instruction formats with examples.	8	Section-4	4

Ans.

Instruction Format

- The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.
- The bits of the instruction are divided into groups called fields.
- The most common fields found in instruction formats are:
  1. An operation code field that specifies the operation to be perform
  2. An address field that designates a memory address or a processor register.
  3. A mode field that specifies the way the operand or the effective address is determined.
- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruct format of a computer depends on the internal organization of its registers.
- Most computers fall into one of three types of CPU organizations:
  1. Single accumulator organization.
  2. General register organization.
  3. Stack organization.

Single Accumulator Organization

- In an accumulator type organization all the operations are performed with an implied accumulator register.
- The instruction format in this type of computer uses one address field.
- For example, the instruction that specifies an arithmetic addition defined by an assembly language instruction as

  ADD X
- Where X is the address of the operand. The ADD instruction in this case results in the operation AC ← AC +M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

General register organization

- The instruction format in this type of computer needs three register address fields.
- Thus the instruction for an arithmetic addition may be written in an assembly language as ADD R1, R2, R3 to denote the operation R1←R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.
- Thus the instruction ADD R1, R2 would denote the operation R1⇓ R1 + R2. Only register addresses for R1 and R2 need be specified in this instruction.
- General register-type computers employ two or three address fields in their instruction format.
- Each address field may specify a processor register or a memory word.
- An instruction symbolized by ADD R1, X would specify the operation R1←R1 + M[X].
- It has two address fields, one for register R1 and the other for the memory address X.

Stack organization

  - The stack-organized CPU has PUSH and POP instructions which require an address field.
  - Thus the instruction PUSH X will push the word at address X to the top of the stack.
  - The stack pointer is updated automatically.
  - Operation-type instructions do not need an address field in stack-organized computers.
  - This is because the operation is performed on the two items that are on top of the stack.
  - The instruction ADD in a stack computer consists of an operation code only with no address field.
  - This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack.
  - There is no need to specify operands with an address field since all operands are implied to be in the stack.

- Most computers fall into one of the three types of organizations.
- Some computers combine features from more than one organizational structure.
- The influence of the number of addresses on computer programs, we will evaluate the arithmetic statement

  X= (A+B) * (C+D)
- Using zero, one, two, or three address instructions and using the symbols ADD, SUB, MUL and DIV for four arithmetic operations; MOV for the transfer type operations; and LOAD and STORE for transfer to and from memory and AC register.
- Assuming that the operands are in memory addresses A, B, C, and D and the result must be stored in memory ar address X and also the CPU has general purpose registers R1, R2, R3 and R4.

Three Address Instructions

- Three-address instruction formats can use each address field to specify either a processor register or a memory operand.

- The program assembly language that evaluates **X = (A+B) * (C+D)** is shown below, together with comments that explain the register transfer operation of each instruction.

```
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
```

- The symbol M [A] denotes the operand at memory address symbolized by A.
- The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.
- The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

Two Address Instructions

- Two-address instructions formats use each address field can specify either a processor register or memory word.
- The program to evaluate **X = (A+B) * (C+D)** is as follows

```
MOV   R1, A     R1←M[A]
ADD   R1, B     R1←R1 + M[B]
MOV   R2, C     R2←M[C]
ADD   R2, D     R2←R2 + M[D]
MUL   R1,R2     R1←R1*R2
MOV   X, R1     M[X]←R1
```

- The MOV instruction moves or transfers the operands to and from memory and processor registers.
- The first symbol listed in an instruction is assumed be both a source and the destination where the result of the operation transferred.

One Address Instructions

- One-address instructions use an implied accumulator (AC) register for all data manipulation.
- For multiplication and division there is a need for a second register. But for the basic discussion we will neglect the second register and assume that the AC contains the result of all operations.
- The program to evaluate X=(A+B) * (C+D) is

```
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
```

- All operations are done between the AC register and a memory operand.
- T is the address of a temporary memory location required for storing the intermediate result.

Zero Address Instructions

- A stack-organized computer does not use an address field for the instructions ADD and MUL.
- The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
- The following program shows how X = (A+B) * (C+D) will be written for a stack-organized computer. (TOS stands for top of stack).

```
PUSH    A    TOS←A
PUSH    B    TOS←B
ADD          TOS←(A + B)
PUSH    C    TOS←C
PUSH    D    TOS←D
ADD          TOS←(C + D)
MUL          TOS←(C + D)*(A + B)
POP     X    M[X]←TOS
```

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation.
- The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

RISC Instructions

- The instruction set of a typical RISC processor is use only load and store instructions for communicating between memory and CPU.
- All other instructions are executed within the registers of CPU without referring to memory.
- LOAD and STORE instructions that have one memory and one register address, and computational type instructions that have three addresses with all three specifying processor registers.
- The following is a program to evaluate X=(A+B)*(C+D)

```
LOAD    R1, A        R1←M[A]
LOAD    R2, B        R2←M[B]
LOAD    R3, C        R3←M[C]
LOAD    R4, D        R4←M[D]
ADD     R1, R1, R2   R1←R1 + R2
ADD     R3, R3, R2   R3←R3 + R4
MUL     R1, R1, R3   R1←R1*R3
STORE   X, R1        M[X]←R1
```

- The load instructions transfer the operands from memory to CPU register.
- The add and multiply operations are executed with data in the register without accessing memory.
- The result of the computations is then stored memory with a store in instruction.

35.	What are the types of addressing modes? Explain.	8	Section-4	4

Ans.

Addressing Modes

- The way the operands are chosen during program execution is dependent on the addressing mode of the instruction.
- Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:
    - To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
    - To reduce the number of bits in the addressing field of the instruction.
- Most addressing modes modify the address field of the instruction; there are two modes that need no address field at all. These are implied and immediate modes.
- **Implied Mode:**
    - In this mode the operands are specified implicitly in the definition of the instruction.
    - For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
    - All register reference instructions that use an accumulator are implied mode instructions.
    - Zero address in a stack organization computer is implied mode instructions.
- **Immediate Mode:**
    - In this mode the operand is specified in the instruction itself.
    - In other words an immediate-mode instruction has an operand rather than an address field.

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

- The address field of an instruction may specify either a memory word or a processor register.
- When the address specifies a processor register, the instruction is said to be in the register mode.
- **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.
- **Register Indirect Mode:**
  - In this mode the instruction specifies a register in CPU whose contents give the address of the operand in memory.
  - In other words, the selected register contains the address of the operand rather than the operand itself.
  - The advantage of a register indirect mode instruction is that the address field of the instruction uses few bits to select a register than would have been required to specify a memory address directly.
- **Auto-increment or Auto-Decrement 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.
- The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated.
- The basic two mode of addressing used in CPU are direct and indirect address mode.
- **Direct Address Mode:**
  - In this mode the effective address is equal to the address part of the instruction.
  - The operand resides in memory and its address is given directly by the address field of the instruction.
  - In a branch-type instruction the address field specifies the actual branch address.
- Indirect Address Mode:
  - In this mode the address field of the instruction gives the address where the effective address is stored in memory.
  - Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.
- A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU.
- 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.
- We have a different addressing mode which is used for a different application.
- **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.
- **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.
  - An index register is a special CPU register that contains an index value.
- **Base Register Addressing Mode:**
  - In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
  - This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.

36.	What are the types of notations? Explain about reverse polish notation.	8	Section-4	4

Ans.

Reverse Polish Notation

- A stack organization is very effective for evaluating arithmetic expressions.
- The common arithmetic expressions are written in infix notation, with each operator written between the operands.
- Consider the simple arithmetic expression.

  A*B+C*D

- For evaluating the above expression it is necessary to compute the product A*B, store this product result while computing C*D, and then sum the two products.
- For doing this type of infix notation, it is necessary to scan back and forth along the expression to determine the next operation to be performed.
- The Polish mathematician Lukasiewicz showed that arithmetic expression can be represented in prefix notation.
- This representation, often referred to as Polish notation, places the operator before the operands. So it is also called as prefix notation.
- The Postfix notation, referred to as reverse Polish notation (RPN), places the operator after the operands.
- The following examples demonstrate the three representations

  Eg: A+B-----> Infix notation
  +AB------> Prefix or Polish notation
  AB+------ > Post or reverse Polish notation
- The reverse Polish notation is in a form suitable for stack manipulation. The expression

  A*B+C*D

  Is written in reverse polish notation as

  AB* CD* +

  And it is evaluated as follows
  - Scan the expression from left to right.
  - When operator is reached, perform the operation with the two operands found on the left side of the operator.
  - Remove the two operands and the operator and replace them by the number obtained from the result of the operation.
  - Continue to scan the expression and repeat the procedure for every operation encountered until there are no more operators.
- For the expression above it find the operator * after A and B. So it perform the operation A*B and replace A, B and * with the result.
- The next operator is a * and it previous two operands are C and D, so it perform the operation C*D and places the result in places C, D and *.
- The next operator is + and the two operands to be added are the two products, so we add the two quantities to obtain the result.
- The conversion from infix notation to reverse Polish notation must take into consideration the operational hierarchy adopted for infix notation.
- This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtraction operations.

Evaluation of Arithmetic Expressions

- Reverse Polish notation, combined with a stack arrangement of registers, is the most efficient way known for evaluating arithmetic expressions.
- This procedure is employed in some electronic calculators and also in some computer.
- The following numerical example may clarify this procedure. Consider the arithmetic expression

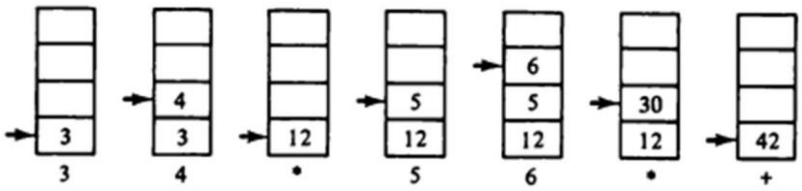
  (3*4) + (5*6)

In reverse polish notation, it is expressed as

34 * 56* +

- Now consider the stack operations shown in Fig. 8-5.

Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



- Each box represents one stack operation and the arrow always points to the top of the stack.
- Scanning the expression from left to right, we encounter two operands.
- First the number 3 is pushed into the stack, then the number 4.
- The next symbol is the multiplication operator *.
- This causes a multiplication of the two top most items the stack.
- The stack is then popped and the product is placed on top of the stack, replacing the two original operands.
- Next we encounter the two operands 5 and 6, so they are pushed into the stack.
- The stack operation results from the next * replaces these two numbers by their product.
- The last operation causes an arithmetic addition of the two topmost numbers in the stack to produce the final result of 42.

37.	Give one numerical example for all addressing modes.	8	Section-4	4

Ans.

- **Numerical Example:**
  - To show the differences between the various modes, we will show the effect of the addressing modes on the instruction defined in Fig. 8-7.

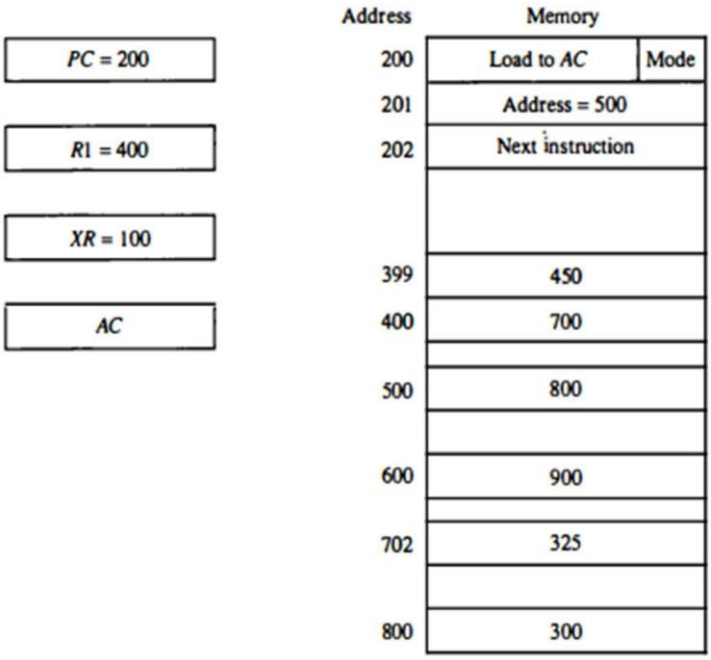


Figure 8-7 Numerical example for addressing modes.

  - The two-word instruction at address 200 and 201 is a "load to AC" instruction with an address field equal to 500.
  - The first word of the instruction specifies the operation code and mode, and the second word specifies the address part.

- PC has the value 200 for fetching this instruction. The content of processor register R1 is 400, and the content of an index register XR is 100.
- AC receives the operand after the instruction is executed.
- In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 500.
- In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC.
- In the indirect mode the effective address is stored in memory at address 500. Therefore, the effective address is 800 and the operand is 300.
- In the relative mode the effective address is 500 + 202 =702 and the operand is 325. (the value in PC after the fetch phase and during the execute phase is 202.)
- In the index mode the effective address is XR+ 500 = 100 + 500 = 600 and the operand is 900.
- In the register mode the operand is in R1 and 400 is loaded into AC.
- In the register indirect mode the effective address is 400, equal to the content of R1 and the operand loaded into AC is 700.
- The auto-increment mode is the same as the register indirect mode except that R1 is incremented to 401 after the execution of the instruction.
- The auto-decrement mode decrements R1 to 399 prior to the execution of the instruction. The operand loaded into AC is now 450.
- Table 8-4 lists the values of the effective address and the operand loaded into AC for the nine addressing modes

TABLE 8-4 Tabular List of Numerical Example

Addressing Mode	Effective Address	Content of *AC*
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

38.	Explain about data transfer instructions.	8	Section-4	4

Ans.

Data Transfer Instructions

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- Table 8-5 gives a list of eight data transfer instructions used in many computers.

TABLE 8-5 Typical Data Transfer Instructions

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

- The load instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The store instruction designates a transfer from a processor register into memory.
- The move instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another and also between CPU registers and memory or between two memory words.
- The exchange instruction swaps information between two registers or a register and a memory word.
- The input and output instructions transfer data among processor registers and input or output terminals.
- The push and pop instructions transfer data between processor registers and a memory stack.
- Different computers use different mnemonics symbols for differentiate the addressing modes.
- As an example, consider the load to accumulator instruction when used with eight different addressing modes.
- Table 8-6 shows the recommended assembly language convention and actual transfer accomplished in each case.

TABLE 8-6 Eight Addressing Modes for the Load Instruction

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

- ADR stands for an address.
- NBA a number or operand.
- X is an index register.
- R1 is a processor register.
- AC is the accumulator register.
- The @ character symbolizes an indirect addressing.
- The $ character before an address makes the address relative to the program counter PC.
- The # character precedes the operand in an immediate-mode instruction.
- An indexed mode instruction is recognized by a register that placed in parentheses after the symbolic address.
- The register mode is symbolized by giving the name of a processor register.
- In the register indirect mode, the name of the register that holds the memory address is enclosed in parentheses.
- The auto-increment mode is distinguished from the register indirect mode by placing a plus after the parenthesized register. The auto-decrement mode would use a minus instead.

39.	Explain about data manipulation instructions.	8	Section-4	4

Ans.

Data Manipulation Instructions

- Data manipulation instructions perform operations on data and provide the computational capabilities for the computer.
- The data manipulation instructions in a typical computer are usually divided into three basic types:
  1. Arithmetic instructions
  2. Logical and bit manipulation instructions
  3. Shift instructions

1. Arithmetic instructions

- The four basic arithmetic operations are addition, subtraction, multiplication and division.
- Most computers provide instructions for all four operations.
- Some small computers have only addition and possibly subtraction instructions. The multiplication and division must then be generated by mean software subroutines.
- A list of typical arithmetic instructions is given in Table 8-7.

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

- The increment instruction adds 1 to the value stored in a register or memory word.
- A number with all 1's, when incremented, produces a number with all 0's.
- The decrement instruction subtracts 1 from a value stored in a register or memory word.
- A number with all 0's, when decremented, produces number with all 1's.
- The add, subtract, multiply, and divide instructions may be use different types of data.
- The data type assumed to be in processor register during the execution of these arithmetic operations is defined by an operation code.
- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision or double-precision data.
- The mnemonics for three add instructions that specify different data types are shown below.

  ADDI Add two binary integer numbers
  ADDF Add two floating-point numbers
  ADDD Add two decimal numbers in BCD
- A special carry flip-flop is used to store the carry from an operation.
- The instruction "add carry" performs the addition on two operands plus the value of the carry the previous computation.
- Similarly, the "subtract with borrow" instruction subtracts two words and borrow which may have resulted from a previous subtract operation.
- The negate instruction forms the 2's complement number, effectively reversing the sign of an integer when represented it signed-2's complement form.

2. Logical and bit manipulation instructions

- Logical instructions perform binary operations on strings of bits store, registers.
- They are useful for manipulating individual bits or a group of that represent binary-coded information.

- The logical instructions consider each bit of the operand separately and treat it as a Boolean variable.
- By proper application of the logical instructions it is possible to change bit values, to clear a group of bits, or to insert new bit values into operands stored in register memory words.
- Some typical logical and bit manipulation instructions are listed in Table 8-8.

TABLE 8-8 Typical Logical and Bit Manipulation Instructions

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

- The clear instruction causes the specified operand to be replaced by 0's.
- The complement instruction produces the 1's complement by inverting all bits of the operand.
- The AND, OR, and XOR instructions produce the corresponding logical operations on individual bits of the operands.
- The logical instructions can also be used to performing bit manipulation operations.
- There are three bit manipulation operations possible: a selected bit can cleared to 0, or can be set to 1, or can be complemented.
  - ✓ The AND instruction is used to clear a bit or a selected group of bits of an operand.
  - ✓ The OR instruction is used to set a bit or a selected group of bits of an operand.
  - ✓ Similarly, the XOR instruction is used to selectively complement bits of an operand.
- Other bit manipulations instructions are included in above table perform the operations on individual bits such as a carry can be cleared, set, or complemented.
- Another example is a flip-flop that controls the interrupt facility and is either enabled or disabled by means of bit manipulation instructions

3. Shift Instructions

- Shifts are operations in which the bits of a word are moved to the left or right.
- The bit shifted in at the end of the word determines the type of shift used.
- Shift instructions may specify logical shifts, arithmetic shifts, or rotate-type operations.
- In either case the shift may be to the right or to the left.
- Table 8-9 lists four types of shift instructions.

TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- The logical shift inset to the end bit position.
- The end position is the leftmost bit position for shift rights the rightmost bit position for the shift left.
- Arithmetic shifts usually conform to the rules for signed-2's complement numbers.
- The arithmetic shift-right instruction must preserve the sign bit in the leftmost position.

- The sign bit is shifted to the right together with the rest of the number, but the sign bit itself remains unchanged.
- This is a shift-right operation with the end bit remaining the same.
- The arithmetic shift-left instruction inserts 0 to the end position and is identical to the logical shift-instruction.
- The rotate instructions produce a circular shift. Bits shifted out at one of the word are not lost as in a logical shift but are circulated back into the other end.
- The rotate through carry instruction treats a carry bit as an extension of the register whose word is being rotated.
- Thus a rotate-left through carry instruction transfers the carry bit into the rightmost bit position of the register, transfers the leftmost bit position into the carry, and at the same time, shift the entire register to the left.

40.	Explain about Reduced Instruction Set Computer (RISC).	8	Section-4	4

Ans.

Reduced Instruction Set Computer

- A computer with large number instructions is classified as a complex instruction set computer, abbreviated as CISC.
- The computer which having the fewer instructions is classified as a reduced instruction set computer, abbreviated as RISC.

Reduced Instruction Set Architecture (RISC)

The main idea behind this is to simplify hardware by using an instruction set composed of a few basic steps for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.

Characteristics of RISC

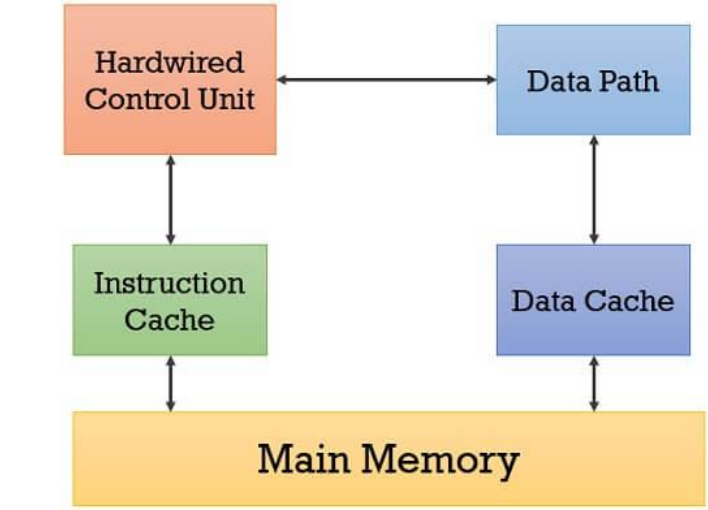
- Simpler instruction, hence simple instruction decoding.
- Instruction comes undersize of one word.
- Instruction takes a single clock cycle to get executed.
- More general-purpose registers.
- Simple Addressing Modes.
- Fewer Data types. A pipeline can be achieved.

Advantages of RISC

- **Simpler instructions**: RISC processors use a smaller set of simple instructions, which makes them easier to decode and execute quickly. This results in faster processing times.
- **Faster execution**: Because RISC processors have a simpler instruction set, they can execute instructions faster than CISC processors.
- **Lower power consumption**: RISC processors consume less power than CISC processors, making them ideal for portable devices.

Disadvantages of RISC

- **More instructions required**: RISC processors require more instructions to perform complex tasks than CISC processors.
- **Increased memory usage**: RISC processors require more memory to store the additional instructions needed to perform complex tasks.
- **Higher cost**: Developing and manufacturing RISC processors can be more expensive than CISC processors.



RISC Architecture