


USN						1	P	E								
	<div>PESIT Bangalore South Campus Hosur road, 1km before Electronic City, Bengaluru -100 Department of Information Science and Engineering</div>															

INTERNAL ASSESSMENT TEST –I

Date: 30/08/2017

Subject & Code: Computer Organization – 15CS34

Name of the faculty: Mrs.Sharmila Banu.A

Max Marks: 40

Semester: III (A & B)

Time: 8.30 am – 10.00 am

Answer any FIVE full questions, choosing one full question from each choice

Q.No	Questions	Marks
1	a) Draw the connection between processor & memory and mention the functions of each component in the connection and explain how the following instructions can be executed with relevant steps: i) Move LOCA, R1 ii) Add R3, LOC2	8

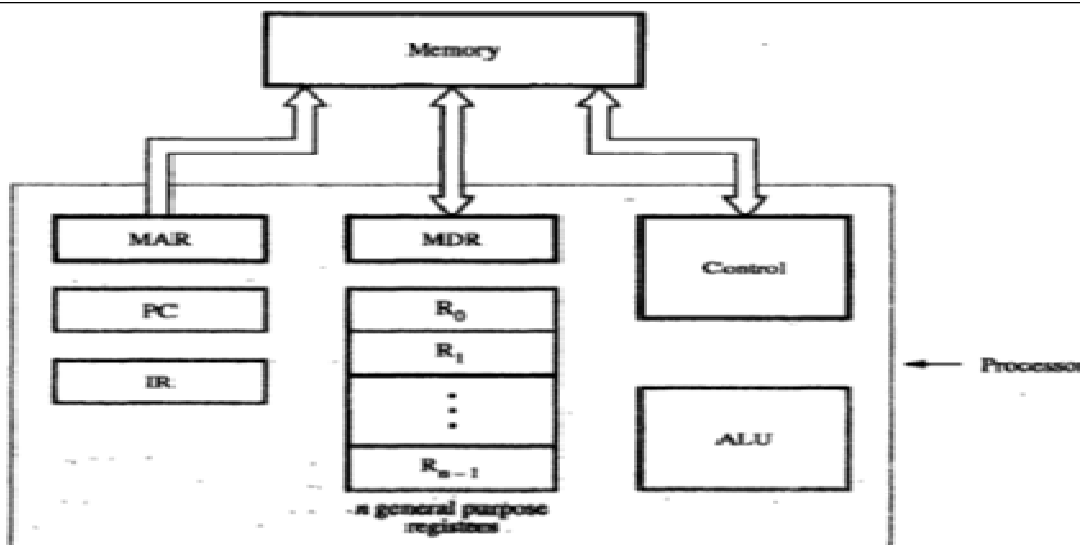


Figure 1.2 Connections between the processor and the memory.

The figure shows how memory & the processor can be connected. In addition to the ALU & the control circuitry, the processor contains a number of registers used for several different purposes.

The instruction register (IR):- Holds the instructions that are currently being executed. Its output is available for the control circuits which generates the timing signals that control the various processing elements in one execution of instruction.

The program counter PC:- This is another specialized register that keeps track of execution of a program. It contains the memory address of the next instruction to be fetched and executed.

Besides IR and PC, there are n-general purpose registers R0 through R_{n-1}.

The other two registers which facilitate communication with memory are: -

1. MAR – (Memory Address Register):- It holds the address of the location to be accessed.

2. MDR – (Memory Data Register):- It contains the data to be written into or read out of the address location.

i) Move LOCA, R1

Operating steps are

- i. Programs reside in the memory & usually get these through the input unit.
- ii. Execution of the program starts when the PC is set to point at the address of MOVE instruction of the program.
- iii. Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
- iv. After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
- v. Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
- vi. If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
- vii. An operand in the memory is fetched by sending its address(LOCA) to MAR & Initiating a read cycle.
- viii. When the operand has been read from the memory to the MDR, it is transferred from MDR to the register R1.
- ix. The contents of PC are incremented so that PC points to the next instruction that is to be executed.

Add R3, LOC2:

- i) Programs reside in the memory & usually get these through the input unit.
- ii) Execution of the program starts when the PC is set to point at the address of ADD instruction of the program.
- iii) Contents of PC are transferred to MAR and a Read Control Signal is sent to the memory.
- iv) After the time required to access the memory elapses, the address word is read out of the memory and loaded into the MDR.
- v) Now contents of MDR are transferred to the IR & now the instruction is ready to be decoded and executed.
- vi) If the instruction involves an operation by the ALU, it is necessary to obtain the required operands.
- vii) An operand in the memory is fetched by sending its address(LOC2) to MAR & Initiating a read cycle.
- viii) When the operand has been read from the memory to the MDR, it is transferred from MDR to the ALU.
- ix) After one or two such repeated cycles, the ALU can perform the ADD operation.
- x) The result of this operation is to be stored in the memory LOC2, the result is sent to MDR.
- xi) Address of location(LOC2) where the result is stored is sent to MAR & a write cycle is initiated.
- xii) The contents of PC are incremented so that PC points to the next instruction that is to be executed.

2	a)	What is performance measurement? Explain overall SPEC rating for computer.	4
----------	-----------	---	----------


The performance measure is the time taken by the computer to execute a given bench mark.

A non-profit organization called SPEC- System Performance Evaluation Corporation selects and publishes bench marks. The program selected range from game playing, compiler, and data base applications to numerically intensive programs in astrophysics and quantum chemistry. In each case, the program is compiled under test, and the running time on a real computer is measured. The same program is also compiled and run on one computer selected as reference.

The ‘SPEC’ rating is computed as follows.

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

Let SPEC_i be the rating for program ‘i’ in the suite. The overall SPEC rating for the computer is given by

USN						1	P	E										
		PESIT Bangalore South Campus Hosur road, 1km before Electronic City, Bengaluru -100 Department of Information Science and Engineering																

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

Where 'n' = number of programs in suite.

Since actual execution time is measured the SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, the OS, the processor, the memory of comp being tested.

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

b)	A program contains 1000 instructions. Out of that 25% instructions requires 4 clock cycles, 40% instructions require 5 clock cycles and remaining require 3 clock cycles for execution. Find the total time required to execute the program running in a 1 GHz machine.	4
----	---	---

N=1000

25% of N= 250 instructions require 4 clock cycles,

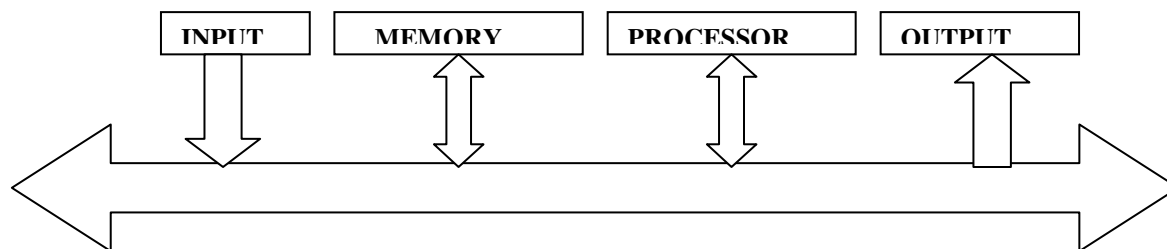
40% of N =400 instructions require 5 clock cycles,

35% of N=350 instructions require 3 clock cycles

$$T = (N*S)/R = 250*4+400*5+350*3/1*10^9 = 1000+2000+1050/1*10^9 = 4.05 \mu s$$

3 a)	What is a bus? Explain the single bus structure used to interconnect functional units in computer system.	4
------	---	---

- The simplest way to interconnect functional units of computer is to use a single bus
- Functional units may be connected by a group of parallel wires.
- The group of parallel wires is called a **bus**.
- Each wire in a bus can transfer one bit of information.
- The number of parallel wires in a bus is equal to the word length of a computer.

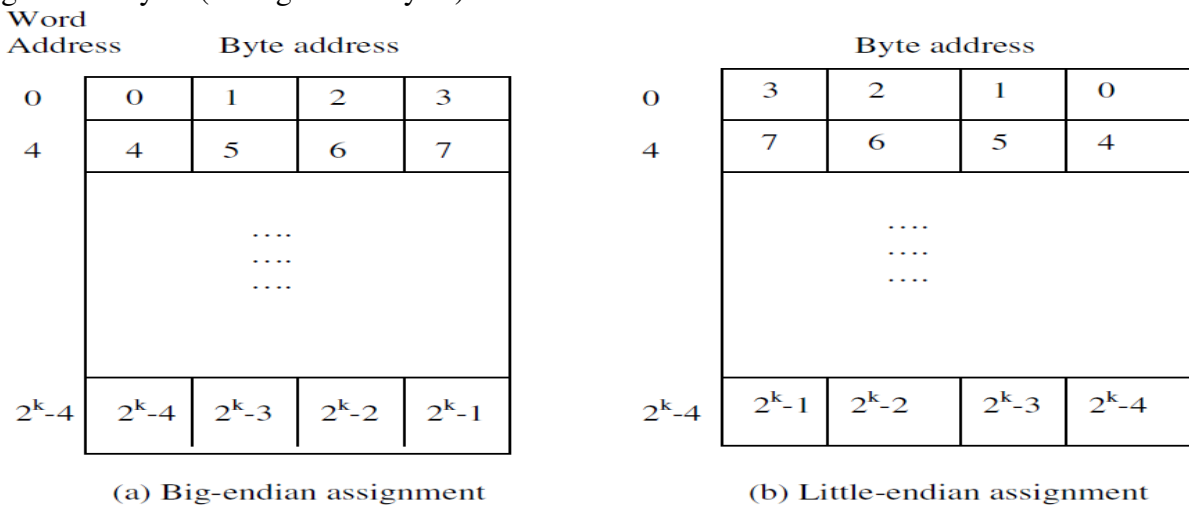


- The devices connected to a bus vary widely in their speed of operation
 - Some devices are relatively slow, such as printer and keyboard

- Some devices are considerably fast, such as optical disks
- Memory and processor units operate are the fastest parts of a computer
- Efficient transfer mechanism thus is needed to cope with this problem
 - A common approach is to include buffer registers with the devices to hold the information during transfers
 - Buffer registers smooth out timing differences among processors, memories and I/O devices.
 - They prevent a high speed processor from being locked to a slow I/O device during a sequence of data transfers.

	b)	Explain BIG-ENDIAN and Little-Endian methods of byte addressing with proper example.	4
--	-----------	--	----------

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




4	a)	Mention Four types of operations to be performed by instructions in a computer. Explain with basic types of instruction formats to carry out $X = (A+B) * (C+D)$	8
----------	-----------	--	----------

1. Data transfers between the memory and the processor registers
2. Arithmetic and logic operations on data
3. Program sequencing and control
4. I/O transfers

Example: Evaluate $(A+B) * (C+D)$

- Three-Address
 1. ADD A, B, R1 ; $R1 \leftarrow M[A] + M[B]$
 2. ADD C, D, R2 ; $R2 \leftarrow M[C] + M[D]$
 3. MUL R1, R2, X ; $M[X] \leftarrow R1 * R2$
- Two-Address
 1. MOV A, R1 ; $R1 \leftarrow M[A]$
 2. ADD B, R1 ; $R1 \leftarrow R1 + M[B]$

USN						1	P	E									
		PESIT Bangalore South Campus Hosur road, 1km before Electronic City, Bengaluru -100 Department of Information Science and Engineering															

3. MOV C, R2 ; $R2 \leftarrow M[C]$
4. ADD D, R2 ; $R2 \leftarrow R2 + M[D]$
5. MUL R1, R2 ; $R2 \leftarrow R1 * R2$
6. MOV R2, X ; $M[X] \leftarrow R2$

- One-Address

1. LOAD A ; $AC \leftarrow M[A]$
2. ADD B ; $AC \leftarrow AC + M[B]$
3. STORE T ; $M[T] \leftarrow AC$
4. LOAD C ; $AC \leftarrow M[C]$
5. ADD D ; $AC \leftarrow AC + M[D]$
6. MUL T ; $AC \leftarrow AC * M[T]$
7. STORE X ; $M[X] \leftarrow AC$

- Zero-Address

1. PUSH A ; $TOS \leftarrow A$
2. PUSH B ; $TOS \leftarrow B$
3. ADD ; $TOS \leftarrow (A + B)$
4. PUSH C ; $TOS \leftarrow C$
5. PUSH D ; $TOS \leftarrow D$
6. ADD ; $TOS \leftarrow (C + D)$
7. MUL ; $TOS \leftarrow (C+D)*(A+B)$
8. POP X ; $M[X] \leftarrow TOS$

5 a)	Define an addressing mode. Explain the following addressing modes, with example for each <ol style="list-style-type: none"> i) Immediate addressing mode ii) Indirect addressing mode iii) Relative addressing mode iv) Auto Increment mode 	8
------	---	---

i) Immediate addressing mode:

The operand is given explicitly in the instruction

Eg: MOVE #200, R0

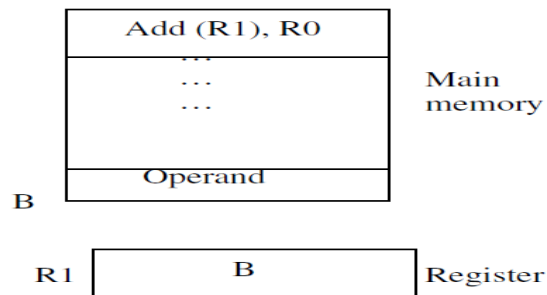
The use of a constant is , i.e. $R0 \leftarrow 200$

$A = B + 6$ MOVE B, R1
 ADD #6, R1
 MOVE R1, A

ii) Indirect addressing mode:

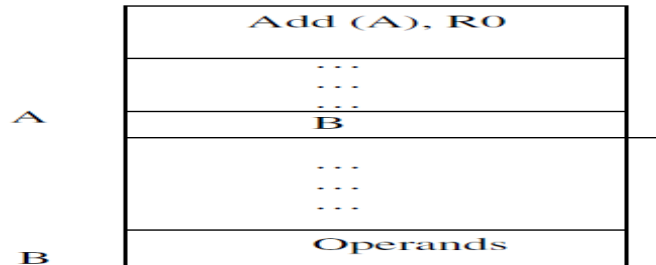
The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

Example1: Through a general-purpose register



To execute the Add instruction in fig (a), the processor uses the value which is in register R1, as the effective address of the operand. It requests a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register R0.

Example2: Indirect addressing through a memory location.



In this case, the processor first reads the contents of memory location A, then requests a second read operation using the value B as an address to obtain the operand.

iii) Relative addressing mode:


The effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri. This mode can be used to access data operands. But, it's most common use is to specify the target address in branch instructions.

- $X(PC)$ – note that X is a signed number
- $EA = X + [PC]$

Example: An instruction such as Branch > 0 LOOP Causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied. This location can be computed by specifying it as an offset from the current value of the program counter. Since the branch target may be either before or after the branch instruction, the offset is given as a signed number.

iv) Auto Increment mode:

- The effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
- Computers that have autoincrement mode automatically increment the contents of a register by a value that corresponds to the size of the accessed operand.
- $(R_i)+$. The increment is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands.

USN						1	P	E								
	<p>PESIT Bangalore South Campus Hosur road, 1km before Electronic City, Bengaluru -100 Department of Information Science and Engineering</p>															

6	a)	Write ALP program to copy 'N' numbers from array 'A' to array 'B' using indirect addresses. (Assume A and B are the starting memory location of an array).	5
----------	-----------	--	----------

Move N, R1
 Move #A, R2
 Move #B, R3
 Clear R0
 LOOP Move (R2), R0
 Move R0, (R3)
 Decrement R1
 Branch > 0 LOOP

	b)	Both the following statements cause the value 300 to be stored in location 1000, but at different times. ORIGIN 1000 DATAWORD 300 And Move #300, 1000 Explain the difference.	3
--	-----------	---	----------

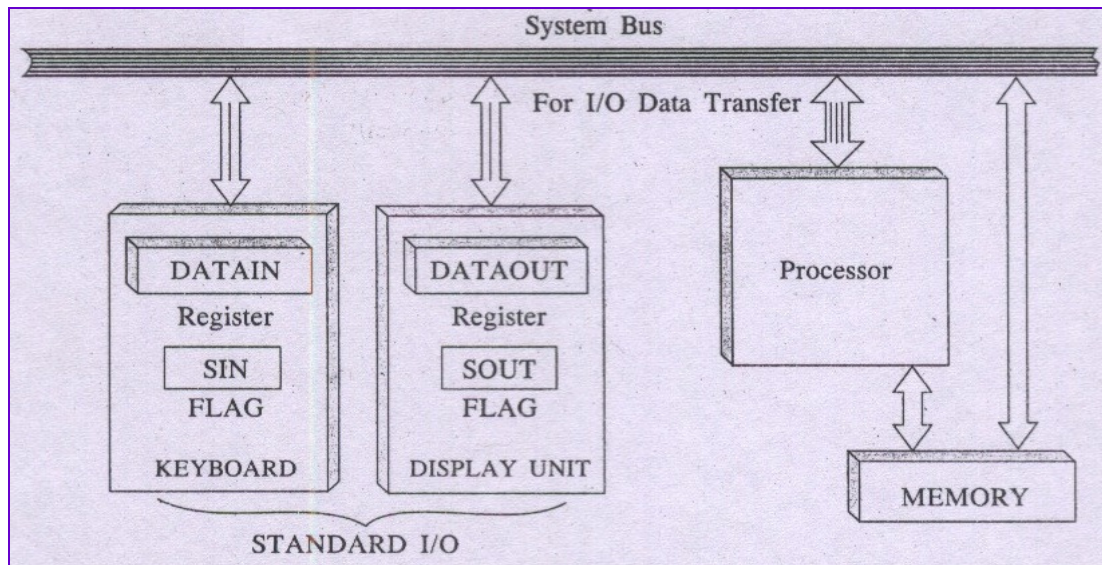
- The assembler directives ORIGIN and DATAWORD cause the object program memory image constructed by the assembler to indicate that 300 is to be placed at memory word location 1000 at the time the program is loaded into memory prior to execution.
- The Move instruction places 300 into memory word location 1000 when the instruction is executed as part of a program.

7	a)	Draw the arrangement of a single bus structure connecting processor and I/O device and brief about memory mapped I/O with a program that reads a line of character from the keyboard and echoes it to the display.	8
----------	-----------	--	----------

Mechanism of I/O transfer between processor and keyboard & video monitor:


- A character key when pressed from the keyboard its scan code is sent to an 8-bit buffer register DATAIN in the keyboard.
- Processor is informed about a valid character data presence in DATAIN register by setting a synchronization flag SIN to 1.
- I/O driver program continuously monitors contents of SIN flag, & when SIN is set to 1, it reads the contents of DATAIN.
- Thus character stored in DATAIN register is transferred to processor over a system bus and SIN content is automatically reset to 0.
- A similar set of events take place while transferring a character data from processor to the display screen.

- Here a **DATAOUT** register that holds a character's code to be displayed when synchronization control flag **SOUT** is set to 1. When **SOUT** equals 1, the display device is ready to receive a character from processor.
- The transfer of a character to **DATAOUT** resets **SOUT** to 0.
- I/O driver program instructions control the status of **SOUT** flag.
- The buffer registers **DATAIN**, **DATAOUT**, and control flags **SIN**, **SOUT** in this hardware setup forms parts of a connectivity circuits commonly known as *device interface or interface hardware*.



- **Memory-Mapped I/O** – some memory address values are used to refer to peripheral device buffer registers.
- Data can be transferred b/w the cpu and these registers using same set of instructions and same status flags.
- No special instructions are needed.
- Ex: MoveByte DATAIN,R0
MoveByte R0,DATAOUT
- Status flags **SIN** and **SOUT** can be included device status register
- Assume that the bit b3 of the status registers **INSTATUS**, **OUTSTATUS** corresponds to **SIN** and **SOUT**.
- **Program to read a line of characters and to display it**

	Move	#LOC,R0	Initialize pointer register R0 to point to the address of the first location in memory where the characters are to be stored.
READ	TestBit	#3,INSTATUS	Wait for a character to be entered in the keyboard buffer DATAIN.
	Branch=0	READ	Transfer the character from DATAIN into the memory (this clears SIN to 0).
	MoveByte	DATAIN,(R0)	Wait for the display to become ready.
ECHO	TestBit	#3,OUTSTATUS	Move the character just read to the display buffer register (this clears SOUT to 0).
	Branch=0	ECHO	Check if the character just read is CR (carriage return). If it is not CR, then branch back and read another character.
	MoveByte	(R0),DATAOUT	Also, increment the pointer to store the next character.
	Compare	#CR,(R0)+	
	Branch≠0	READ	

USN						1	P	E							
		PESIT Bangalore South Campus Hosur road, 1km before Electronic City, Bengaluru -100 Department of Information Science and Engineering													

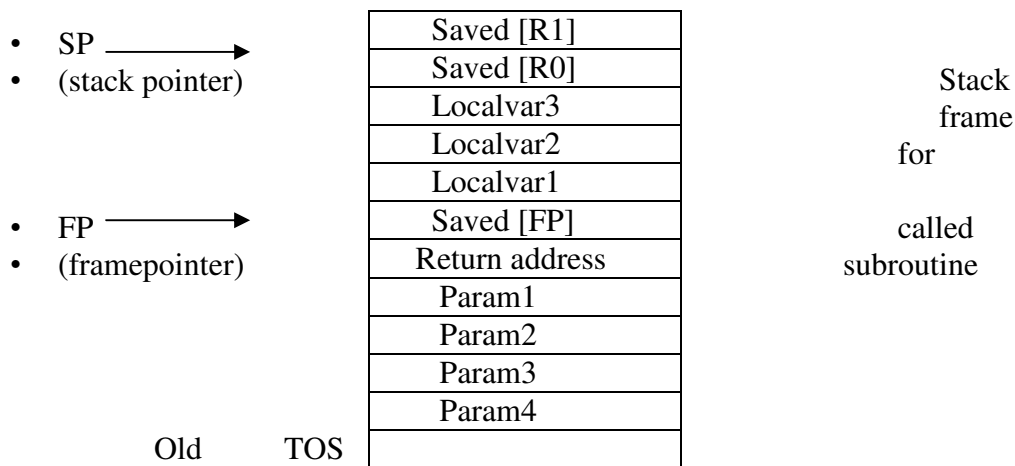
8	a)	Registers R1 and R2 of a computer contain the decimal values 1200 and 4600. What is the effective address of the memory operand in each of the following instructions? (a) LOAD 20(R1), R5 (b) MOVE #3000, R5 (c) STORE R5, 30(R1, R2) (d) ADD - (R2), R5	4
---	----	---	---

- a) 1220
b) part of the instruction
c) 5830
d) 4599

	b)	What is stack frame? How to set the frame pointer while executing the subroutine? Give an example.	4
--	----	--	---

- The consecutive location on stack that constitute a private work space for the subroutine, created at the time the subroutine is entered and freed up when the subroutine returns control to the calling program.
- Such space is called stack frame of the subroutine.
- The stack frame locations get filled as and when each subroutine is called for execution.
- Also, stack frame space is vacated soon after the completion of execution of subroutine
- Another pointer register, called the frame pointer (FP) is used, for convenient access to the parameters passed to the subroutine and to the local memory variables used by the subroutine
- The pointers SP and FP are manipulated as the stack frame is built, used, and dismantled for a particular of the subroutine.
- Thus, the first two instructions executed in the subroutine are
 Move FP, -(SP)
 Move SP, FP
- After these instructions are executed, both SP and FP point to the saved FP contents
- Space for the three local variables is now allocated on the stack by executing the instruction
 Subtract #12, SP
- Finally, the contents of processor registers R0 and R1 are saved by pushing them onto the stack. At this point, the stack frame has been set up as shown in the fig.
- The subroutine now executes its task.
- When the task is completed, the subroutine pops the saved values of R1 and R0 back into those registers, removes the local variables from the stack frame by executing the instruction.
 Add #12, SP
- And pops the saved old value of FP back into FP.
- At this point, SP points to the return address, so the Return instruction can be executed, transferring control back to the calling program.
- Calling program is responsible for removing the parameters from the stack frame, some which may be the results passed back by the subroutine

- The SP now points to the old TOS



9	a)	Explain how to encode the instructions into 32-bit words.	5
---	----	---	---

- OP code: the type of operation to be performed and the type of operands used may be specified using an encoded binary pattern
- Suppose 32-bit word length, 8-bit OP code, 16 registers in total, 3-bit addressing mode indicator.

The instruction

Add R1, R2

Has to specify the registers R1 and R2, in addition to the OP code. If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register addressing mode is used for each operand.

The instruction

Move 24(R0), R5

Requires 16 bits to denote the OP code and the two registers, and some bits to express that the source operand uses the Index addressing mode and that the index value is 24.

The instructions can be encoded in a 32-bit word. Depicts a possible format. There is an 8-bit Op-code field and two 7-bit fields for specifying the source and destination operands. The 7-bit field identifies the addressing mode and the register involved (if any). The “Other info” field allows us to specify the additional information that may be needed, such as an index value or an immediate operand.

But, what happens if we want to specify a memory operand using the Absolute addressing mode?

The instruction


Move R2, LOC

(a) One-word instruction

Opcode	Source	Dest	Other info
--------	--------	------	------------

(b) Two-Word instruction

Opcode	Source	Dest	Other info
Memory address/Immediate operand			

USN						1	P	E									
		PESIT Bangalore South Campus Hosur road, 1km before Electronic City, Bengaluru -100 Department of Information Science and Engineering															

(c) Three-operand instruction

Op code	Ri	Rj	Rk	Other info
---------	----	----	----	------------

Requires 18 bits to denote the OP code, the addressing modes, and the register. This leaves 14 bits to express the address that corresponds to LOC, which is clearly insufficient.

And #FF000000. R2

In which case the second word gives a full 32-bit immediate operand.

If we want to allow an instruction in which two operands can be specified using the Absolute addressing mode, for example

Move LOC1, LOC2

Then it becomes necessary to use two additional words for the 32-bit addresses of the operands.

If the Add instruction only has to specify the two registers, it will need just a portion of a 32-bit word. So, we may provide a more powerful instruction that uses three operands

Add R1, R2, R3

Which performs the operation

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

A possible format for such an instruction is shown in fig c.

	b) Register R has a value 1101 1101. Determine the sequence of binary values in R after a logical shift left followed by a rotate right and followed by arithmetic shift right.	3
--	--	----------

R=1101 1101

Logical Shift Left: R= 1011 1010

Rotate Right: R= 0101 1101

Arithmetic shift right: R=0010 1110

10	a) What is subroutine? Explain the role of stack in subroutine with an example while passing parameters.	8
-----------	---	----------

SUBROUTINES: -

In a given program, it is often necessary to perform a particular subtask many times on different data-values. Such a subtask is usually called a subroutine. For example, a subroutine may evaluate the sine function or sort a list of values into increasing or decreasing order.

It is possible to include the block of instructions that constitute a subroutine at every place where it is needed in the program. However, to save space, only one copy of the instructions that constitute the subroutine is placed in the memory, and any program that requires the use of the subroutine simply branches to its starting location. When a program branches to a subroutine we say that it is calling the subroutine. The instruction that performs this branch operation is named a Call instruction.

After a subroutine has been executed, the calling program must resume execution, continuing immediately after the instruction that called the subroutine. The subroutine is said to return to the program that called it by executing a Return instruction.

SUBROUTINE NESTING AND THE PROCESSOR STACK:-

A common programming practice, called subroutine nesting, is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents. Hence, it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise, the return address of the first subroutine will be lost.

Subroutine nesting can be carried out to any depth. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call sequence. That is, return addresses are generated and used in a last-in-first-out order. This suggests that the return addresses associated with subroutine calls should be pushed onto a stack. A particular register is designated as the stack pointer, SP, to be used in this operation. The stack pointer points to a stack called the processor stack. The Call instruction pushes the contents of the PC onto the processor stack and loads the subroutine address into the PC. The Return instruction pops the return address from the processor stack into the PC.

PARAMETER PASSING:-

When calling a subroutine, a program must provide to the subroutine the parameters, that is, the operands or their addresses, to be used in the computation. Later, the subroutine returns other parameters, in this case, the results of the computation. This exchange of information between a calling program and a subroutine is referred to as parameter passing.

Parameter passing may be accomplished in several ways. The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine. Alternatively, the parameters may be placed on the processor stack used for saving the return address.

- If many parameters are involved, there not may be enough registers available for passing them to subroutine
- A stack can be used in such situations

Assume top of stack is at level 1 below.

Move	#NUM1, -(SP)	Push parameters onto stack.
Move	N, -(SP)	
Call	LISTADD	Call subroutine
		(top of stack at level 2).
Move	4(SP), SUM	Save result.
Add	#8, SP	Restore top of stack
		(top of stack at level 1).
:		

The purpose of the subroutines is to add a list of numbers. Instead of passing the actual list entries, the calling program passes the address of the first number in the list. This technique is called passing by reference. The second parameter is passed by value, that is, the actual number of entries, n, is passed to the subroutine.

SubRoutine

LISTADD	MoveMultiple R0-R2,-(SP)	
	Move 16(SP),R1	
	Move 20(SP),R2	
	Clear R0	
LOOP	Add (R2)+,R0	
	Decrement R1	
	Branch>0 LOOP	
	Move R0,20(SP)	
	MoveMultiple	
(SP)+,R0-R2		
	Return	

Top of stack



level 3

[R2]

[R1]

[R0]



level 2

Return Address

N

NUM1



level 1