18/01/2021

Computer Organization

Different types of computers:-

Since their introduction in 1940s, digital computers are of different types that vary widely in size, cost, computational power and intended use. Modern computers can be divided roughly into the following categories.

* Embedded computers
* Personal computers
* Servers and Enterprise systems
* Super computers and Grid computers

Embedded Computers:-

* These are integrated into a large device or system in order to automatically monitor and control a physical process or environment.
* They are used for a specific purpose rather than for general (tasking) procesing tasks.
* Typically applications includes industrial and home automation, appliances, tale communication products, and vehicles
* Users may not be even aware of the role that computers play in such systems

Personal Computers:-

* Personal computers have achieved widespread use in homes, education and business and engineering office setting primarily for dedicated individual use
* They support a variety of applications such as general computation, document preparation, computer aided design, audio visual entertainment, inter personal communication, internet browsing.
* A number of classifications are used for personal computers

Desktop computers

Workstation computers

Portable and Notebook computers

* Desktop computers serve general needs and fit within a typical personal workspace
* Workstation computers offer higher computational capacity and more powerful graphical display capabilities for engineering and scientific work
* Portable and Notebook computers provide the basic features of a personal computer in a smaller light-weight package.
* They can operate on batteries to provide mobility.

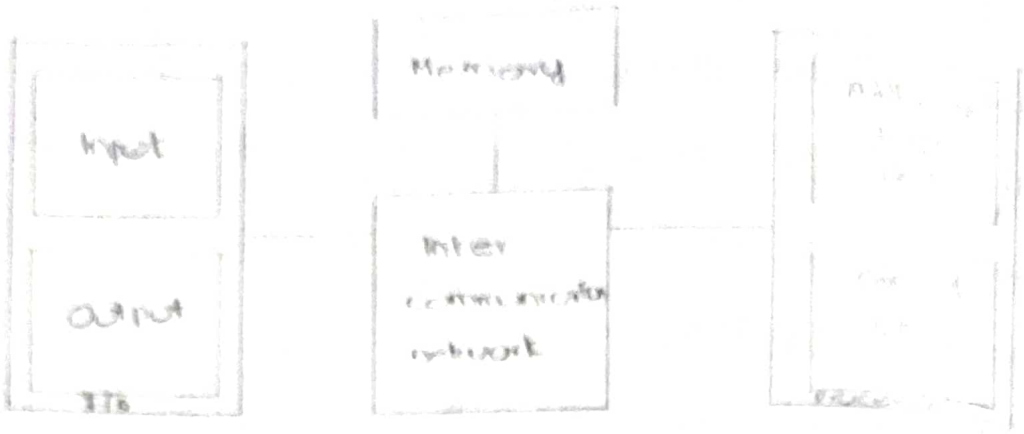
Servers and Enterprise systems:-

* Large computers that are meant to be shared by a potentially large numbers of users who access them from (or) some form of personal computers over a public or private network
* Such computers may host large databases and provide information processing for a government agency or a commercial organization.

Supercomputers and Grid computers:-

* Normally offer the highest perfomance.
* They are the most expensive and physically the largest category of computers.
* Super computers are used for the highly demanding computations needed in weather forecasting, simulation engineering design and scientific work.
* They have high cost.
* Grid computers provide most cost-effective alternative.
* They combine a large number of personal computers and disc storage units in a physically distributed high speed network, called a grid, which is called as a co-ordinating computing resource.
* By evenly distributing the computational workload across the grid, it is possible to achieve high perfomance on large applications ranging from numerical computation to information searching.

Functional units:-



* There must be interconnections for data transmissic

Input unit:-

* Computers accept coded information through input units - the computer keyboard
* Whenever a key is pressed, the corresponding let or digit is automatically translated into its binary code and transmitted to procesor.
* Some of the input units are: Touch pad, mouse, joystick, trackball, microphone, camera, internet

Memory unit:-

* Its function is to store programs and data
* There are two classes of storage

Primary storage :-

* It is called main memory
* Programs must be stored in memory while they are being executed
* Consists of large number of semiconductor storage cells - capable of storing one bit of information

* They are handled in groups of fixed size called words

* The memory is organized so that one word can be stored or retrieved in one basic operation

* To provide easy access to any word in the memory, a distinct address is associated with each word location

A particular word is accessed by specifying its address and issuing a control command to the memory that starts the storage or retrieval process

A memory in which any location can be accessed in a short and fixed amount of time after specifying its address is called a random access memory

The time required to access memory is memory access time.

22/01/24

Memory hierarchy - cache, main memory

Registers → cache → main memory.

Registers

cache

Main memory

secondary memory

Arithmetic & Logic Unit :-

* Most computer operations are executed in ALU of the processor
* Load the operands into memory, bring them to the processor - perform operation in ALU, store the result back to memory or retain in the processor
* Registers
* Fast control of ALU

23/1/2021

* Activity in a computer is governed by instructions
* To perform a task, An appropriate program consisting of a list of instructions is stored in the memory
* Individual instructions are brought from the memory into the processor, which executes the specified operation
* Data to be used as operands and also stored in the memory.

A Typical instruction:-

Add R0, LOCA

Add the operand at memory location LOCA to the operand in a register R0 in the processor.

Every processor haves some special purpose and general purpose registers.

The notation to represent general purpose registers is

$R_0, R_1, R_2 \ldots R_{n-1}$

The special purpose registers are used represent as MAR(memory address register), MDR(memory data register), PC (program counter), IR(instruction register) which are used for specific tasks

2 Jul 2021

Number representations and arithmetic operations:-

Signed Integers:-

There're three major representations

Sign and magnitude

One's complement

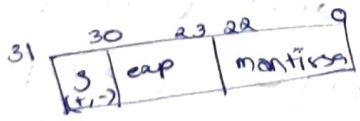
Two's complement

* In all three systems MSB position 0 for positive & 1 for negative.

* Positive values have identical representations in all the systems, but negative values have different representations

Floating values:-

single precision: 32 bit



While storing floating point numbers, you have to normalise and store

In Ex: $1001100101 = 1.\underbrace{001100101}_{\text{mantissa}}$

Sign = 0 (positive)

exponent = 7

mantissa = 001100101

biased exponent = $(7+128)_{10 \to 2}$

Ex: 10011001.01 (64 bits - double precision)

(1, 11, 52)

sign = 0

exp = 7 + 1024

mantissa = 00110101

Ascii :-

a: 97, A: 965

Sign extension:-

```
short int x = 15213;
int ix = (int) x;
```

x = 15213 = 0011 1011 0110 1101

ix = 15213 = 0000 0000 0000 0000 0011 1011 0110 1101

y = -15213 = 1100 0100 1001 0011

iy = -15213 = 1111 1111 1111 1111 1100 0100 1001 0011

2's complement addition and subtractions:-

Addition:

```
(+2)  0010          0100 (+4)
(+3)  0011          1010 (-6)  [2's complement]
(+5)  0101          1110 (-2)  [2's complement]

(-5)  1011          0111 (+7)
(-2)  1110          1101 (-3)
(-7)  1001          0100 (+4)
```

subtraction:-

```
  1101 (-3)
- 1001 (-7)
-------
  1101
  0111
+ 1
-------
  0110 (+4)
```

```
  0010 (+2)
- 0100 (+4)
-------
  0010
  1100
-------
  1110 (-2) (complement)
```

```
  1001 (+9)
- 0011 (+5)
-------
  1001
  0101
+ 1
-------
  0110 (-2)
```

```
  1001 (-7)
- 0001 (+1)
-------
  1001
  1111
+ 1
-------
  1000 (-8)
```

If carry in to high order bit = carry out and ignore

If carry-in differs from carry-out, then overflow

* Now consider a different representation of the mod-16 circle
* We will reinterpret the binary vectors outside the circle to represent the signed integers from -8 through +7 in the 2's complement representation as shown inside the circle

Ex: Add +7 to -3

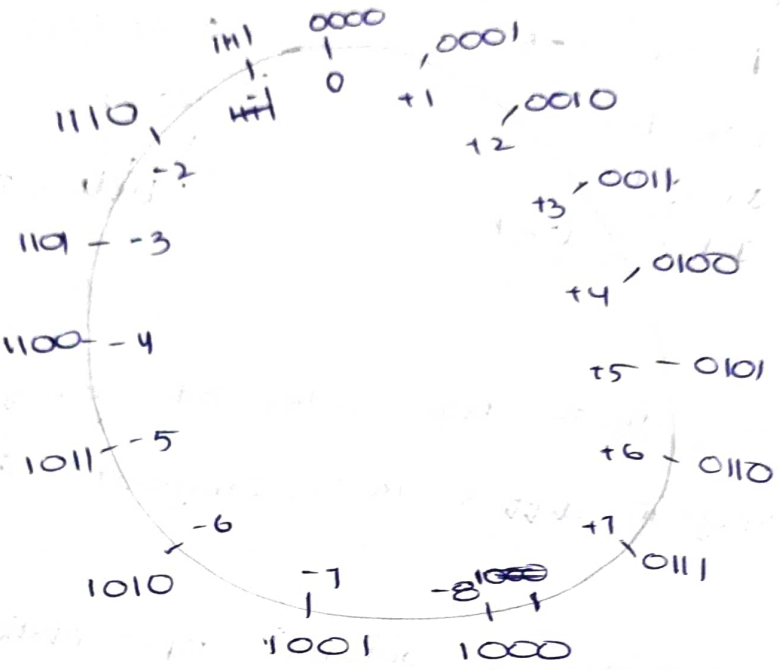
Locate 0111 on circle, move 1101 (13) steps in clockwise direction to arrive 0100 (4) which is correct answer

* To understand 2's complement arithmetic, consider addition modulo N.
* A helpful graphical device for the description of addition of unsigned integers mod N is a circle with the values

0 through N-1 marked along its perimeter.

* the decimal values 0 through 15 are represented by their 4-bit binary values 0000 through 1111 on the outside of circle

* To perform addition, locate a, and move b units clockwise to arrive at (a+b) mod 16



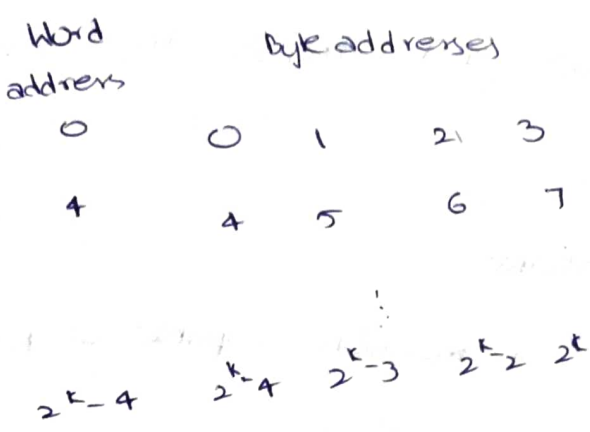
01/02/2021

Memory locations, addresses and operations:-

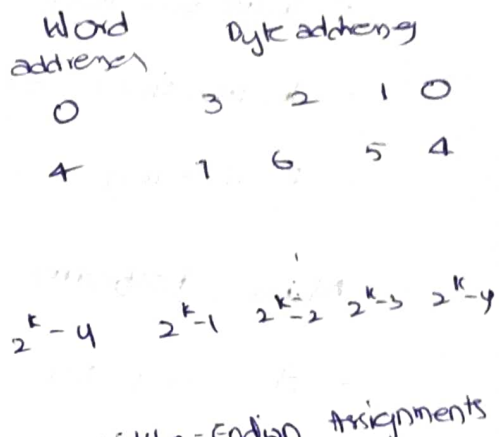
* Memory is organised as a large array of bytes a words
* Word is a group of n bits, 'n' is word length
* Memory is represented as collection of words
* The word length range from 16 to 64 bits
* Accessing the memory requires address for each location.
* So, use numbers from 0 to $2^k - 1$ to address the successive memory locations. have up to 2^k memory locations.

* Memory consits of many millions of storage cells each of which can store 1 bit
* Data is usually accessed in n-bit groups.

Big- Endian and Little- Endian Assignments:-



a) Big- Endian assignment



b) Little-Endian Assignments

Memory Operation:-

- Both program instruction and data are stored in memory.
* To execution and instruction the processor first take the instruction from memory
- the operands and results are also moved between memory and processor
- Two basic operations are involved in memory

→ Read Operation

→ To start the read operation the processor sends the address of the memory and a request that its contents to be read.

Instruction and Instruction Sequencing:-

- Register Transfer notation
- Assembly language notation
- RISC and CISC instruction sets
- Introduction to RISC instruction sets
- Instruction execution and straight line sequencing
- Branching

Must - Perform Operations:-

- A computer have instructions capable of performing 4 types of must operations
- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers.

Register Transfer notations:-

- We will see the transformation from one location in a computer to the other. Possible locations are memory locations, processor registers, registers in I/O sub systems
- Such locations are assigned with convenient names
- Eg: LOC, PLACE, A, Var11 etc

→ Predefined names for processor registers may be R0, R1, etc

→ contents of any location are represented with square brackets

→ Identify a location by a symbolic name standing for its hardware binary address (LOC, A0...)

→ contents of a location are denoted by placing square brackets around the name of the location

(R1 ← [LOC] , R3 ← [R1]+[R2])

→ Instruction specifies an operation to be performed and the operands involved.

→ In assembly language ADD, Move, load etc are called memonics

→ Memonics are the words that describes operations.

Assembly language notation:-

- Represent machine instructions and programs
- Move LOC , R1

  R1 = R1 ← [LOC]
- Add R_1, R_2, R_3

  $R_3 = R_3 \leftarrow [R1]+[R2]$

→ To initiate write

→ The procesor sends the address of desired location to the memory along with the data to be written into the location

→ Memory uses the address and data perform a write

→ store (or Write)

→ Overwrite the content in memory

RISC and CISC Instruction sets:-

- One of the most important characteristics that distinguish different computers in the nature of their instructions.
- There are two fundamentally different approaches in the design of instruction sets for modern computers

Key characteristics of RISC instruction sets:-

- Each instruction fits in a single word
- A load/store architecture is used, in which
→ memory operands are accessed only using load and store instructions
→ All operands involved in an arithmetic or logic operation must either be in processor registers, or one of the operands may be given explicitly within the instruction word.

Eg:- C = A + B

```
Load  R2, A
Load  R3, B
Add   R4, R2, R3
Store R4, C
```

25/08/2021

RISC: Reduced Instruction set Computer.

CISC: Complex Instruction set computer

RISC Characteristics:-

* Relatively few instructions
* Relatively few addressing modes
* More accesses are limited to load and store instructions
* All operations are done with the registers of CPU
* It is designed to reduce the execution time by simplifying the instruction computer.
* Each instr

* Program Counter (PC) always holds the address of the next instruction to be executed
* The address is copied to MAR (buses) through buses
* The Instruction is copied to IR from memory which is called Fetching (Fetch phase)
* Once the instruction is fetched, enters into the execution phase and starts decoding in which the ~~is~~ what the instruction is, operands are etc.
* If any operands are presents, the registers load the operands by get operands and loaded into MDR
* Then the execution starts, one after the other sequencing so our instruction exection is straight-line sequencing

* Execution of a given instruction is a two-phase procedure
* In the first phase, called instruction fetch, the instruction is fetched from the memory location whose address is in PC
* This instruction is placed in instructio Register (IR), the processor.
* At the start of the second phase, called instruct execute, the instruction in IR is examined to determine which operation is to be performed.
* The specified operation is then performed by the processor.
* This involves a small number of steps such as fetching operands from the memory or from proces registers, performing an arithmetic or logic operation, and storing the result in the destination location
* Let us consider how this program is executed
* The processor contains a register called program counter which holds the address of the next instruction to be executed
* To begin executing a program, the address of its first instruction must be placed into the PC
* The processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in order of increasing addresses
* This is called straight-line sequencing. During execution, the PC is incremented by 4 to point to the next instruc

06/10/2021

Register Indirect Addressing mode:-

Register name is specified as part of the instruction That register contains the address of the operand in the memory. i.e, The registers contains the address of the operand other than the operand

Load R2, (R3)

Branch constructions:-

- We now introduce branch instructions This type of instruction loads a new address into the Program counter
- As a result, the processor fetches and executes the instruction at this new address, called the branch target instead of the one in sequence
- A conditional branch instruction causes a branch only if a specified condition is satisfied.
- If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed

Branch_if_[R2] > 0 LOOP

- One way of implementing conditional branch instructions is to compare the contents of two registers and then branch to the target instruction if the comparision meets the specified

Branch_if_[R4] > [R5] LOOP

Branch_greater_than R4, R5 LOOP

BGT R4, R5, LOOP

Addressing modes:-

- The way in which memory operand is addres is called Addreing modes
- The term addressing modes refers to the way which the operand of an instruction is specified

RISC-type addressing modes:-

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute	LOC	Effective Addres = LO
Register indirect	(Ri)	EA = [Ri]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri, Rj)	EA = [Ri] + [Rj]

Immediate mode:-

Actual data is specified as part of the instruction i.e., instruction as an operand field rather than as addres field. The operand field contains the actual operand

ADD R4, R6, #200

Register Mode:-

Register mode is specified as part of the instructio i.e., register contains actual data i.e, operands are in registers

ADD R4, R2, R3

Absolute mode (or) Direct Addressing mode:-

Memory address is specified as a part of the instruction, this memory location contains the actual value. ie; the operand resides in the memory and its address is specified in the address field of the instruction.

Load R2, Num1.

Indirect addressing:-

- The address field of the instruction specifies the address of memory location
- That contains the effective address of the operand.

Two references to the memory are required to fetch the operand

ADD R1, (A)

Register Indirect addressing mode:-

Register name is specified as part of the instruction That register contains the address of the operand in the memory. i.e; The register contains the address of the operand rather than the operand

Load R2, (R3)

Relative Addressing mode:-

In this, effective address of the operand is obtained by adding the content of PC with address part of the instruction

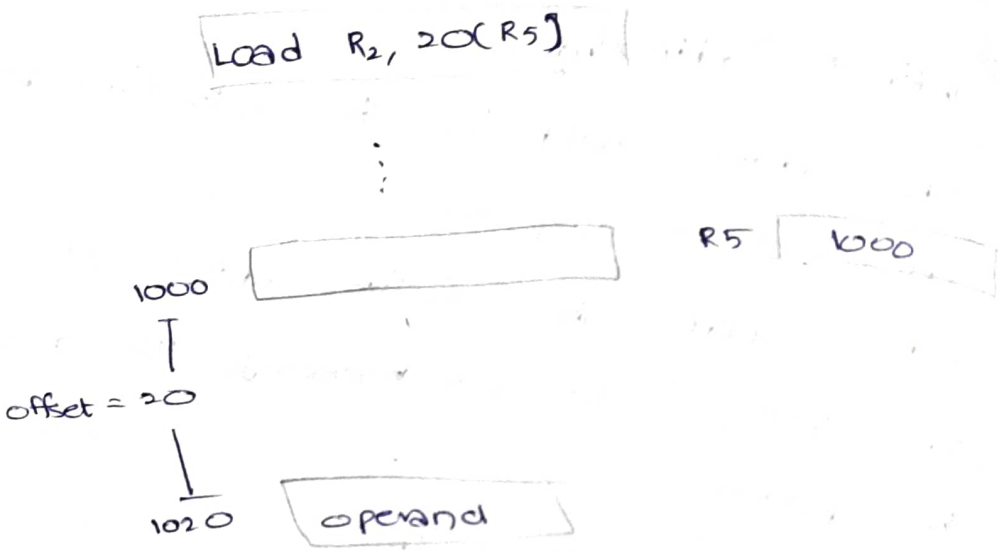
EA = X + (PC)

Indexed Addressing mode:-

- Effective operand is given by.

$$EA = X + [Ri]$$

- In this addressing mode the content of the given register



Base with index:-

- Effective address is the sum of the contents of register Ri and Rj
- second register is called base register

ADD R2, [Ri+Rj]

Auto increment and Auto decrement:

- The EA of operand is the contents of a register specified in the instruction.
- After accessing the operand, contents of this register is automatically incremented by 1

Add R1, (R2)+

Stacks:-

- A stack is a list of data elements or words
- LIFO
- Push, Pop, Top
- It is storage structure that stores information in the format of LIFO
- It is a group of memory locations with the register that holds the address of the top of the element
- The register that holds the address of the top element of the stack is called stack Pointer (SP)
- Stack can be implemented in two ways
  - → memory stack
  - → register stack
- Memory stack: code segment, Data segment:

Push: SP ← SP - 4

M[SP] ← DR (Data register)

subtract SP, SP, #1

store Rj, (SP)

New item to be pushed on the stack is in processor register Rj

Pop: DR ← M[SP]

SP ← SP + 4

Load Rj, (SP)

ADD SP, SP, #4

Load the top value from the stack into register Rj and increment the SP by 4

- Most of the computers do not provide hardware to check stack full or empty
- The stack limits can be checked by using two processor registers one to hold the upper limit and other to hold lower limit
- Before push, SP is compared with upper limit
- After pop, SP is compared with lower limit

10/02

Subroutines:-

-It is a set instructions which are used repeatedly in a program is called subroutine

(or)

- the block of instructions which carries out a specific and well defined task is called subroutine
- Some space is allocated for subroutines
- When a subroutine is called, PC jumped to the starting address of the subroutine
- Before it jumps the present PC value is stored in a link register
- After execution of subroutine, when a return statement appears the link register's value is placed in PC

Subroutine calls and returns:-

- The call instruction is just a special branch instruction that performs the following instructions
  - → store the contents of PC in link register
  - → Branch to the target address specified by the call instruction
- The return instruction is a special branch instruction

that performs the following operation,

→ Load the contents of link register in PC

* the way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method

* the simplest subroutine linkage method is to save the return address in a specific location, which may be a register dedicated to this function, such a register is link register

Subroutine Nesting:-

- the one subroutine calls another
- The nested calls the link register will be overwritten, so stack is used for this purpose

Parameter Passing:-

- When calling a subroutine, the program must provide parameters to the subroutine to be used in the computation
- Later, the subroutine returns other parameters, which are the results of the computation
- This exchange of information between a calling program and a subroutine is referred to as parameter passing
- Parameter passing can be accomplished in several ways
- The parameter may be placed in registers or in memory locations, where they can be accessed by the subroutine
- Alternatively, the memory may be placed in the processor stack

- Passing the parameter through processor registers is straight forward

11/02/2021

Calling program

```
Load    R2, N          Parameter 1 is list size
Move    R1, #NUM1      parameter 2 is list location
Call    LIST ADD       call subroutine
Store   R3, SUM        save result
```

Subroutine:-

```
LISATADD:   Subtract
Subtract    SP, SP, #4      save the contents of
Store       R5, (SP)        R5 on the stack
Clear       R3              Initilize sum to zero

LOOP:
Load        R5, (R4)        Get the next number
Add         R3, R3, R5      Add the number to sum
Add         R4, R4, #4      Increment pointer by 4
Subtract    R2, R2, #1      Decrement the counter
Branch_if_[R2]>0  LOOP
Load        R5, (SP)        Restore contents of R5
Add         SP, SP, #4
return                      Return to calling program
```

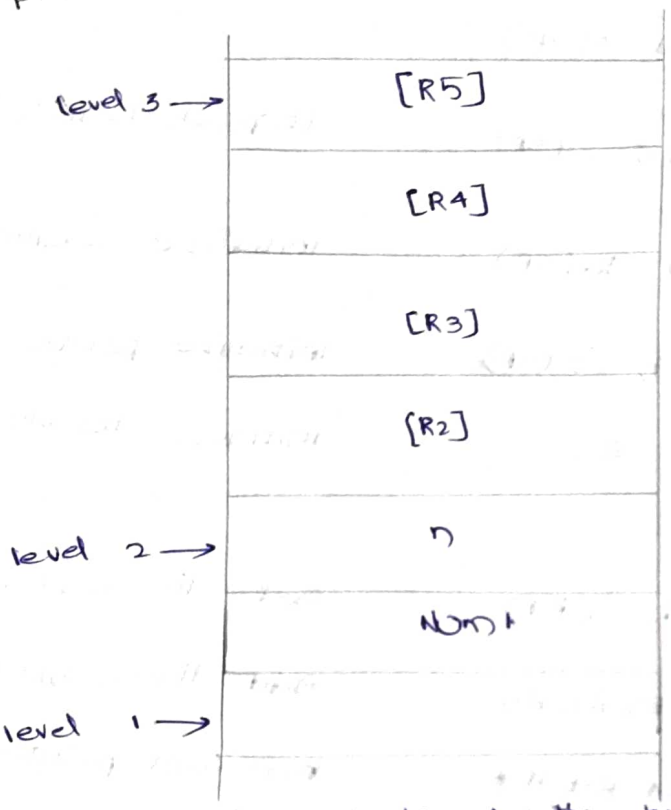
parameter passing through registers

- If many parameters are involved, there may not be enough general purpose registers available for passing them to the subroutine
- The parameter stack provides a convinient and flexible

mechanism for passing an arbitrary number of parameters

- Assume that before the subroutine is called, the top of the stack is at level 1.
- The calling program pushes the address Num1 and the value n onto the stack and call LISTADD
- The top of the stack is now at level 2

parameters passed on stack:-



stack contents for the below program

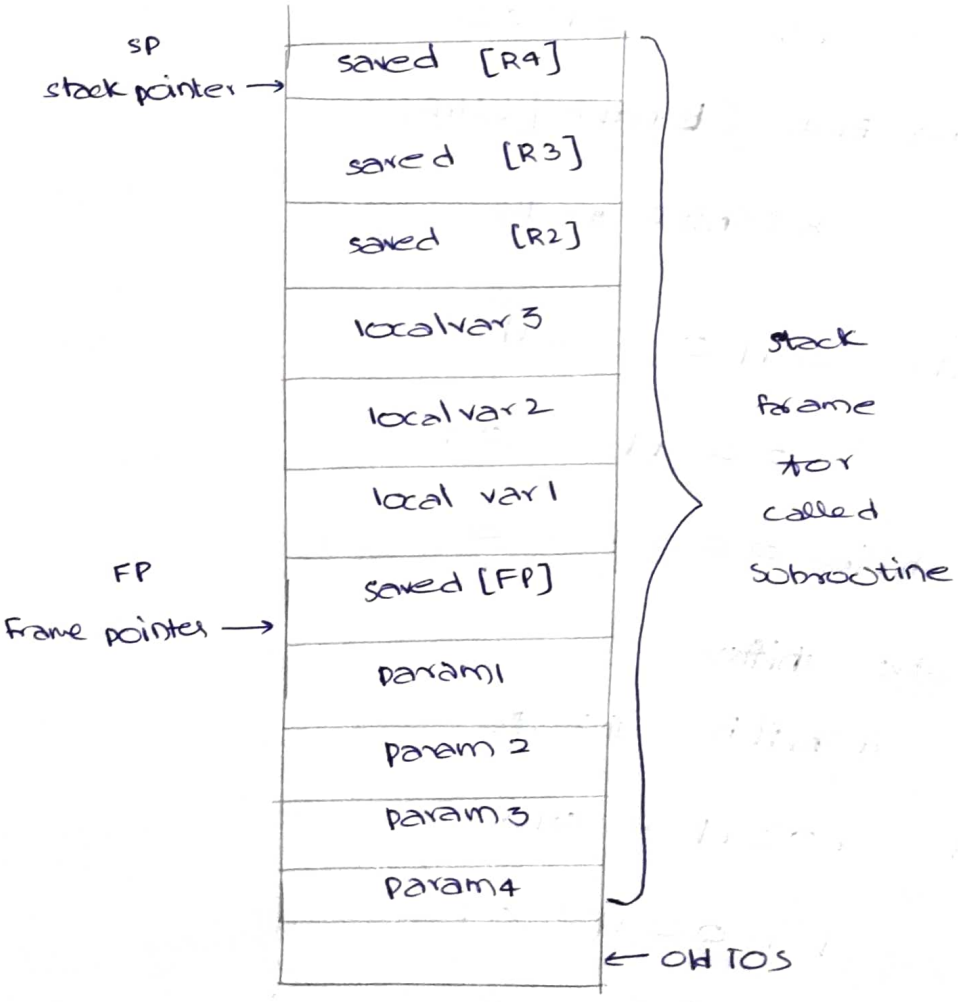
Move	R2, #Num1	Push parameters onto stack
subtract	SP, SP, #4	-
store	R2, (SP)	-
Load	R2, N	-
subtract	SP, SP, #4	-
store	R2, (SP)	-
call	LISTADD	call subroutine
		(top of stack is at level 2)
Load	R2, 4(SP)	Get result from stack, save it in sum

store	R2, SUM	save it in sum
Add	SP, SP, #8	Restore top of stack (top of stack is at 1)
LISTADD:		
Subtract	SP, SP, #16	save registers
store	R2, 12(SP)	-
store	R3, 8(SP)	-
store	R4, 4(SP)	-
store	R5, (SP)	(top of stack is at 2)
Load	R2, 16(SP)	Initialize counter to n
Load	R4, 20(SP)	Initialize pointer to list
clear	R3	Initialize sum to 0
LOOP:-		
Load	R5, (R4)	Get the next number
Add	R3, R3, R5	Add this number to sum
Add	R4, R4, #4	Increment pointer by 4
Subtract	R2, R2, #1	Decrement the counter
Branch_if_[R2]>0	LOOP	-
store	R3, 20(SP)	Put result in stack
Load	R5, (SP)	Restore registers
Load	R4, 4(SP)	-
Load	R3, 8(SP)	-
Load	R2, 12(SP)	-
Add	SP, SP, #16	(top of stack is at 2)
Return	-	Return to calling program

15/02/2021

Stack frame :-

- During execution of the subroutine, six locations at the top of the stack contain entries that are needed by the subroutine
- These locations constitute a private work space for the subroutines, allocated at the time the subroutine is entered and deallocated when the subroutine returns control to the calling program
- Such space is called a stack frame
- If the subroutine requires more space for local memory variables, the space for these variables can also be allocated on the stack.



Subroutine stack frame example

shows an example for a commonly used layout of information in a stack frame

In addition to the stack pointer SP, it is useful to have another pointer register, called the frame pointer (FP), for convinient access to the parameters passed to the subroutine and the local memory variables used by the subroutine

Additional instructions:-

Logical shifts:-

Shifting Left (LShiftL / LSHL):-

LShiftL #2, R0

Actual: 0110...011

0110...0110

After: 10....01100

Shifting Right (BShiftR / LSHR):-

LShiftR #2, R0

Actual: 0110...011

0011...01

After shifting: 0001...0

Arithmetic shifts:-

AShiftR #2, R0

Actual: 10011...010

11001...01

After: 11100...0

Rotate :-

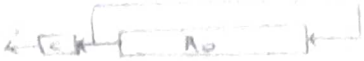
a) Rotate without carry to left

RotateL #2, R0

Actual: 01110...011

1110...0110

After: 110....01101



b) Rotate left with carry

RotateLC #2, R0

Actual: 01110...011

1110...0110

After: 110...01100

c) Rotate right without carry

RotateR #2, R0

Actual: 01110...011 C=0

10111...01 C=1

After: 110111..0

d) Rotate right with carry

RotateR #2, R0

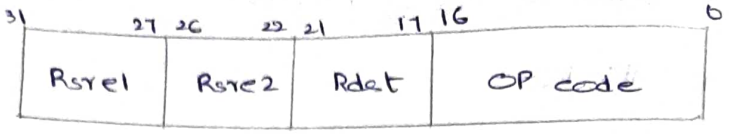
Actual: 01110...011

00111...01

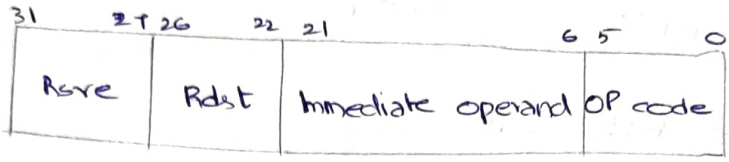
After: 100111...0

There are some of the rotate instructions

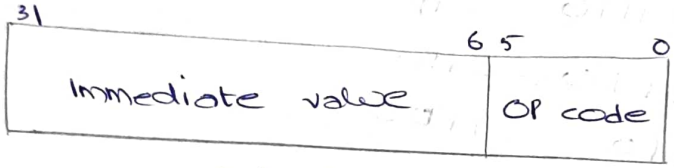
Encoding of machine instructions:-



Register - operand format



Immediate - operand format



Call format

16/02/2021