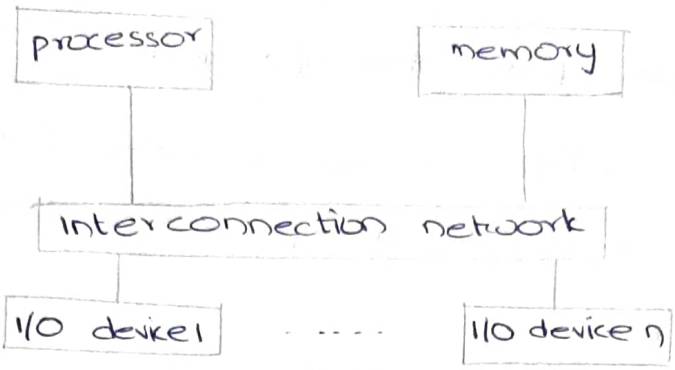
16/02/2021

UNIT-2 : Basic Input/Output :-

Accessing I/O devices:-



- The components of a computer system communicate with each other through an interconnection network
- The interconnection network consists of circuits needed to transfer information between the processor, the memory unit, a number of I/O devices
- Each I/O device is not taken as a separate device, rather an I/O device is as an addressable locations to the processor
- Each I/O device is mapped with one memory location
- Multiple I/O devices may be connected to the processor and the memory via a bus, bus consists of three sets of lines to carry address, data and control signals each I/O device is assigned an unique address
- To access an I/O device, the processor places the address on the address lines, the device recognises the address, and responds to the control signals.
- Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor This creates the need for mechanisms to synchronize

data transfer between them)

* Program controlled I/O

→ Processor repeatedly monitors a status flag to achieve the necessary synchronization

→ Processor polls the I/O devices

* Two other mechanisms used for synchronizing data transfers between the processor and memory:

→ Interrupts

→ Direct memory access

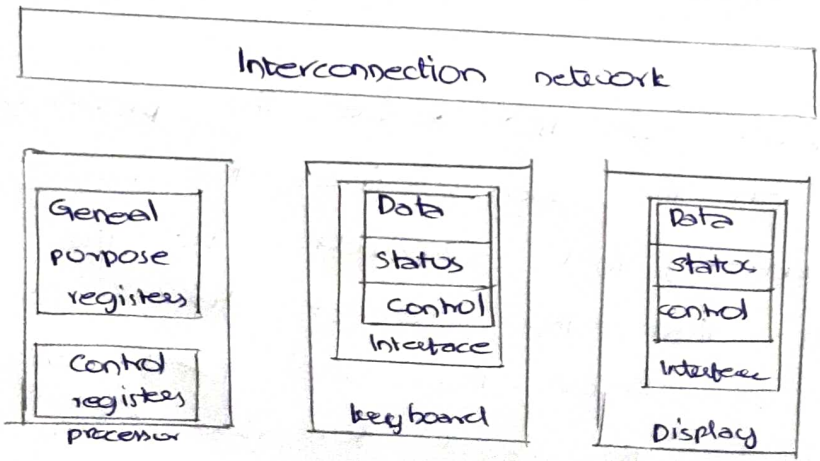
Program - Controlled I/O:

• Read in characters input from a keyboard and produce characters output on a display screen

→ Rate of data transfer (keyboard, display, processor)

→ Difference in speed between processor and I/O device creates the need for mechanisms to synchronize the transfer of data

→ A solution: on output, the processor sends the first character and then waits for a signal from the display that the character has been recieved, it then sends the second character. Input is sent from the keyboard in a similar way



From the processor perspective, an I/O device appears as a set of special purpose registers of 3 types

Status registers:

May hold information about the current status of the device

control registers:-

These are used to control and configure the device

Data registers:-

Used to read the data from or send data to I/O device

→ Each of the I/O registers has an address; so that CPU can read/write the data into specific registers

→ These registers are accessed by program instructions

Memory mapped I/O:-

- It is a way to exchange data and instructions between CPU and I/O devices attached to it.
- It is the one where processor and the I/O devices share the same address space
- Any instruction that can access memory can be used to transfer data to and from and I/O devices

- LOAD R2, DATAIN

DATAIN is the address of a register in an input

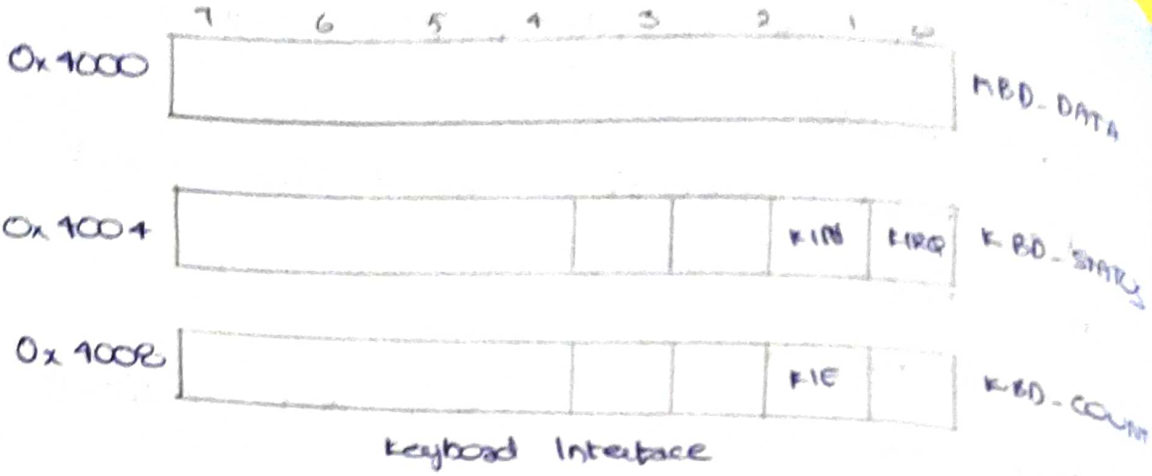
- Store R2, DATAOUT

DATAOUT, which is a register in an output device

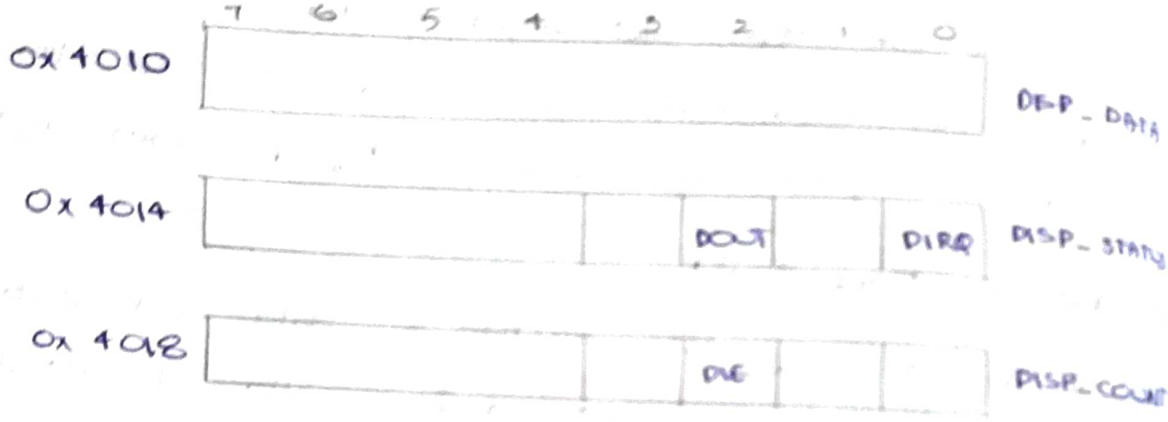
```
READWAIT    Read the KIN Flag
            Branch to READWAIT if KIN = 0
            Transfer data from KBD - DATA TO R5
```

character into processor register R5

Address



Keyboard Interface



Display Interface

I/O device interface:-

- I/O devices are connected to an interconnection network, by using the circuit called the device interface
- It provides the means of data transfer between CPU and I/O devices
- The computer system includes a special hardware between the CPU and I/O to supervise and synchronize all I/O transfers. These components are called Interface

Interrupts :-

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready
  → Do so by sending a hardware signal called an interrupt to the processor
  → Atleast one of the bus control lines, called an interrupt-request line is dedicated for this purpose
- Processor can perform other useful tasks while it is waiting for the device to be ready
- Three types of interrupts

External interrupts :-

Generated by external devices like I/O devices requesting the transfer of data

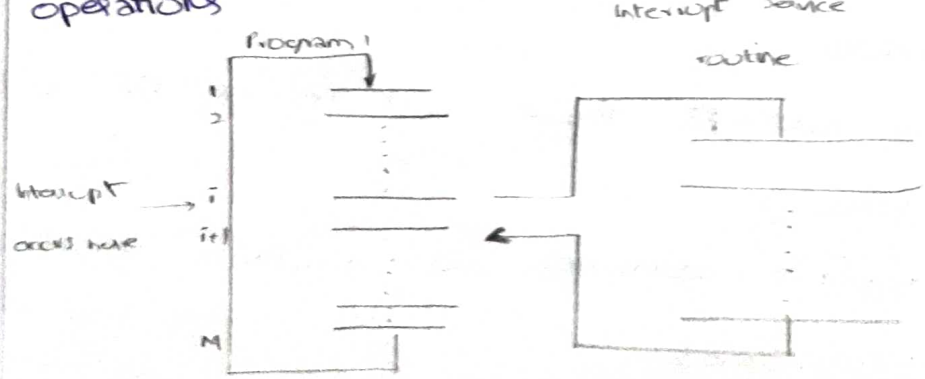
Internal :-

called traps. Raised from illegal use of an instruction/data. These are caused by program.

Eg: Divide by zero, Register overflow, invalid opcode, stack overflow

Software interrupts :-

These are used by the programmers for various operations



17/02/2021

- Treatment of an interrupt service routine is very similar to that of a subroutine
- However there are significant differences:
→ A subroutine performs a task that is required by the calling program
→ Interrupt-service routine may not have anything in common with the program it interrupts
→ Interrupt-service routine and the program that it interrupts may belong to different users.

What happens when an interrupt occurs:-

- Processor is executing the instruction located at address i when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine
- When an interrupt occurs, control must be transferred to the interrupt service routine
- But before transferring control, the current contents of the PC (i+1), must be saved in a known location
- This will enable the return-from-interrupt instruction to resume execution at i+1
- Return address, or the contents of PC are usually stored on the processor stack
- ISR tells the processor, what to do when an interrupt occurs
- For every interrupt, there must be an ISR available in main memory
- All ISR starting addresses are available in IVT (Interrupt vector table)

i) Execute the current instruction, and saves the PC on the stack

ii) It saves the current status of the PC register

iii) It jumps to the memory location of the IVT

iv) The processor gets the address of ISR from IVT and jumps to it. And starts the execution of ISR.

- As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
- This will enable the interrupted program to resume execution upon return from interrupt service routine
- Saving and restoring information can be done automatically by the processor or explicitly by program instructions

Drawbacks:-

→ Increases the total execution time

→ Increases the delay between the time an interrupt request is recieved, and the start of execution of the interrupt-service routine

→ The delay is called interrupt latency

- In order to reduce the interrupt latency, most processors save only the minimal amount of information:

→ This minimal amount of information includes program counter and processor status registers

- Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine

• When a processor recieves an interrupt - request it must branch to the interrupt service routine

• It must also inform the device that it has recognized the interrupt request

• This can be accomplished in two ~~fo~~ ways

→ Some processors have an explicit interrupt - acknowledge control signal for this purpose

→ In other cases, the data transfer that takes place between the device and the processor can be used to inform the device

• Processors generally provide the ability to enable and disable such interruptions are desired

• One simple way is to provide machine instructions such as interrupt - enable and interrupt - disable in PS register.

• To avoid interruption by the same device during the execution of an interopt service routine:

→ First instruction of an interrupt service routine can be intesupt - disable

→ Last instruction of an interrupt service routine can be interrupt - enable

• Processor has special line called Edge - triggered line

Multiple devices:

• Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.

→ Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests

• This means by which these issues are handled vary from one computer to another

• When an interrupt request is recieved it is necessary to identify the particular device that raised the request

• If two devices raise interrupt requests at the same time, it must be possible to break the tie and select one of the two requests for service

• When the processor recieves an interrupt request over the control line, this information is available in the status register of the device requesting an interrupt:

→ The status register of each device has an IRQ bit which it sets to 1 when it requests an interrupt

• Interrupt service routine can poll the I/O devices connected to the bus. The first device with IRQ equal to 1 is the one that is serviced

• Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

• An alternate is vectored interrupts.

Vectored interrupts:-

• The device requesting an interrupt may identify itself directly to the processor

→ Device can do so by sending a special code (4 to 8 bits) the processor over the bus. If processor busy wait until INTA signal.

→ Code supplied by the device may represent a part of the starting address of the interrupt service routine.

→ The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.

• Usually the location pointed to by the interrupting device is used to store the starting address of the interrupt service routine

Interrupt nesting:-

• During the execution of one ISR, if it allows another interrupt then that is known as interrupt nesting.

• This is like our nested subroutine calls.

Priority of devices:-

• I/O devices are organized in a priority structure.

• A priority level is assigned to a processor that can be changed under program control.

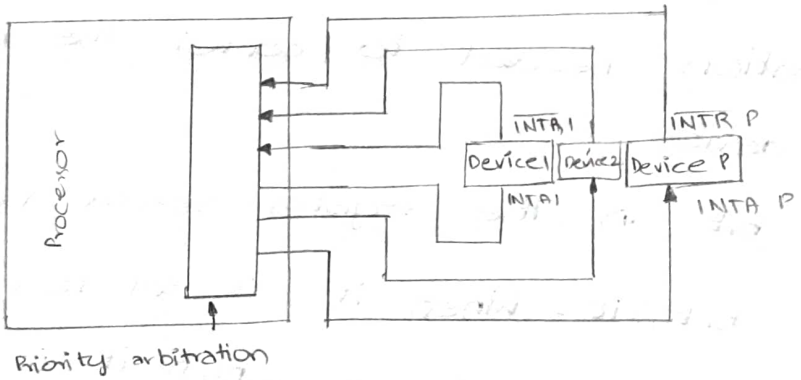
→ Priority level of a processor is the priority of the program that is currently being executed

→ When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.

→ If the device sending an interrupt request has a higher priority than the processor, the processor accepts the interrupt request

Handling simultaneous requests:-

- We also need to consider the problem of simultaneous arrivals of interrupt requests from two or more devices.
- The processor must have some means of which request to service first
- Polling the status registers of the I/O devices is the simplest such mechanism
- In this case, priority is determined by the order in which the devices are polled.
- Each device has a separate interrupt-request and interrupt-acknowledge line. Each interrupt request line is assigned a different priority level.
- Interrupt requests recieved over these lines are sent to a priority arbitration circuit in the processor.
- If the interrupt request has a higher priority level than the processor, then the request is accepted.



Polling scheme:-

If the processor uses a polling (machine) mechanism to poll the status registers of I/O device, it determine which device is requesting an interrupt.

- In this case the priority is determined by the order in which the devices are polled, the first device with status bit set to 1 is the device whose interrupt request is accepted.

Daisy chain scheme:-

- Devices are connected to form a daisy chain
- Services share the interrupt request line, and interrupt - acknowledge line is connected to form a daisy chain
- Then devices raise an interrupt request, the interrupt request line is activated.
- The processor in response activates interrupt-acknowledge.
- Received by device 1, If device1 doesnot need service it passes the signal to give service that is electrically closest to the processor has the highest priority.
- I/O devices vary in complexity from simple to quite complex.
- A commonly used approach is to provide a control registes in the device interface, which holds the information needed to control the behaviour of the device
- One bit in the registes serves as the interrupt-enable bit, IE. When it is set to 1, the device is placed into a mode in which it is allowed to interrupt the processor whenever it is ready for an I/O transfer

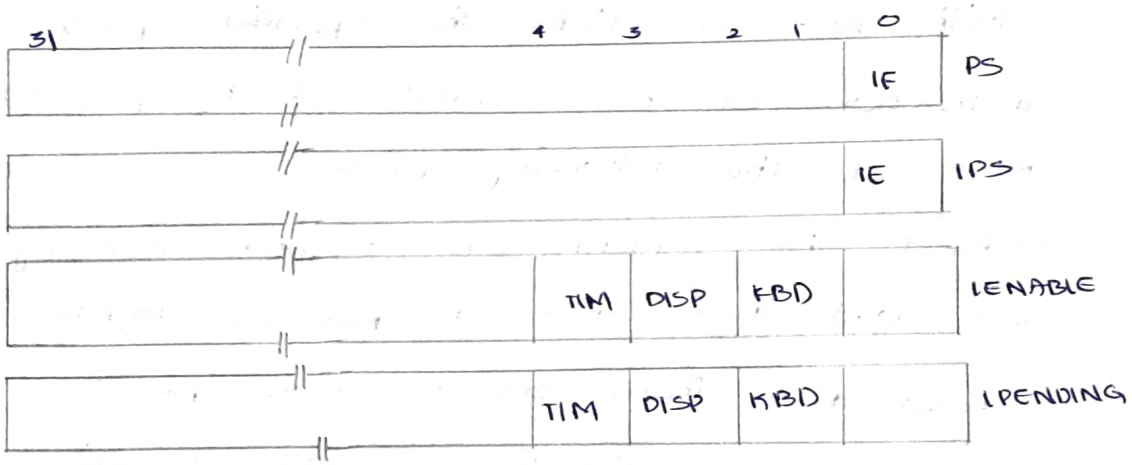
Eg: Keyboard, Display.

Controlling I/O device behaviour:-

- It is important to ensure that interrupt requests are generated only by those I/O devices that the processor is currently willing to recognize.
- Hence we need a mechanism in the interface circuits of individual devices to control whether a device is allowed to interrupt the processor.
- The control needed is usually provided in the form of an interrupt-enable bit in the device's interface circuit.

Processor control registers:-

- To deal with interrupts it is useful to have some other control registers.



control registers in Processor

- It becomes necessary to save the contents of IPS on the stack if nested interrupts are allowed.
- The IENABLE registers allows the processor to selectively respond to individual I/O devices.
- A bit may be assigned for each device, as shown in the figure for the keyboard, display and a timer. A circuit when a bit is set to 1, the processor will accept interrupt requests from the corresponding devic

20/9/24

Basic Processing Unit :-

steps to be performed - Instruction:-

Fetch Phase:

- Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase)

$$IR \leftarrow [[PC]]$$

- Assuming that the memory is byte addressable, increment the contents of the PC by 4

$$PC \leftarrow [PC] + 4$$

- Carry out the actions specified by the instruction in the IR (execution phase)

Instruction execution phase:-

- With few exceptions, the operation specified by an instruction can be carried out by performing one or more of the following actions:

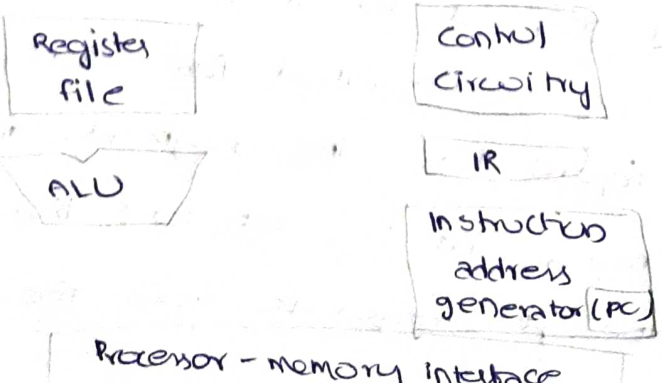
→ Read the contents of a given memory location and load them into a processor register.

→ Read data from one or more processor registers.

→ Perform an arithmetic or logic operation and place the result into a processor register.

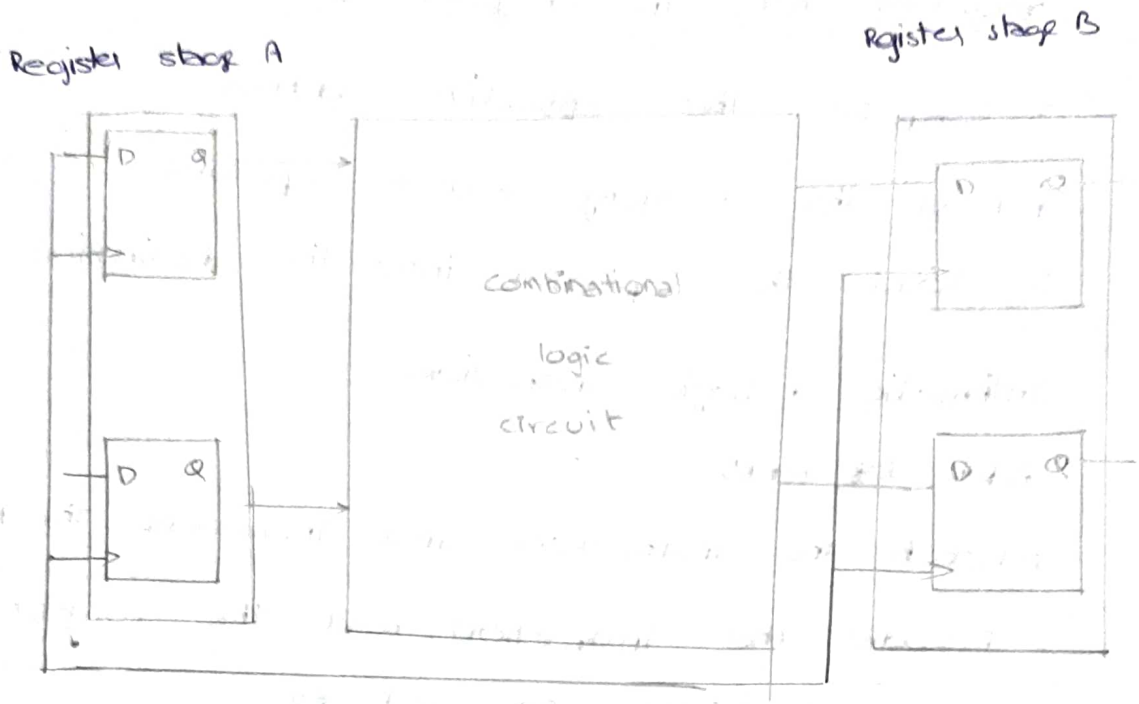
→ Store data from a processor register into a given memory location

Hardware components needed:-



Data Processing hardware:

Processing is a procedure that transforms raw data into some meaningful information



Load instructions:-

Load R5, X(R7)

- which uses the index addressing mode to load a word of data from memory location X + [R7] into register R5

Execution of this instruction involves the following actions:

- Fetch the instruction from the memory
- Increment the program counter
- Decode the instruction to determine the operation to be performed
- Read register R7
- Add the immediate value X to contents of R7
- Use X+[R7] as EA, load data recieved from location and load Instruction execution:- into R5
- Actions involved in fetching and executing instructions
- Illustrate of actions using a few representative RISC style instructions

1. Fetch the instruction and increment the PC

2. Decode the instruction and read the contents of register R7 in register file

3. Compute the effective address

4. Read the memory source operand.

5. Load the operand into the destination register, R5

Arithmetic & Logic instructions:-

Add R9, R4, R5

1. Fetch the instruction and increment the PC

2. Decode the instruction, read the contents of source registers R4 and R5

3. compute the sum [R4] + [R5]

4. No action

5. Load the result into destination register, R9

Add R3, R4, #1000

2. Decode the instruction & read R4

3. compute the sum [R4] + 1000

Store instructions:-

Store R6, X(R8)

1. Fetch the instruction and increment the PC

2. Decode the instruction and read R6 & R8

3. Compute the effective address X + [R8]

4. Store contents of R6 into location X + [R8]

5- No action

Five-step sequence of actions:-

1-Fetch an instruction and increment the PC

2. Decode the instruction and read registers from the register file

3. Perform ALU operation

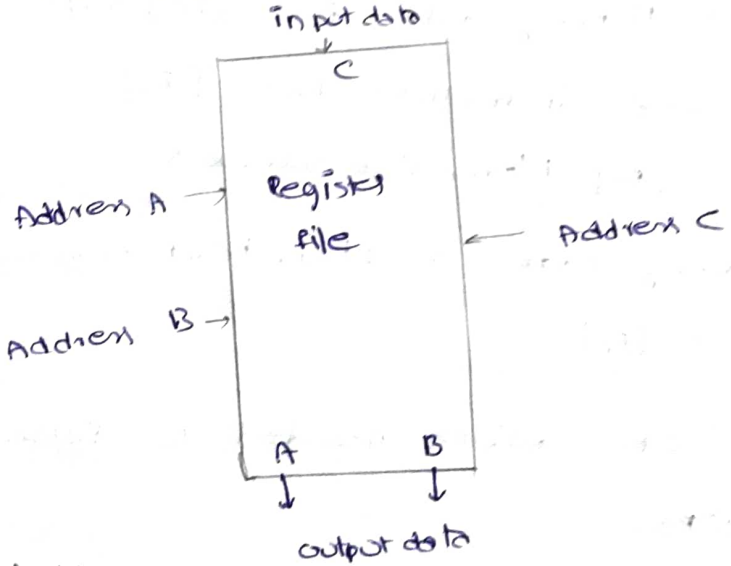
4. Read or write memory data if the instruction involves a memory operand.

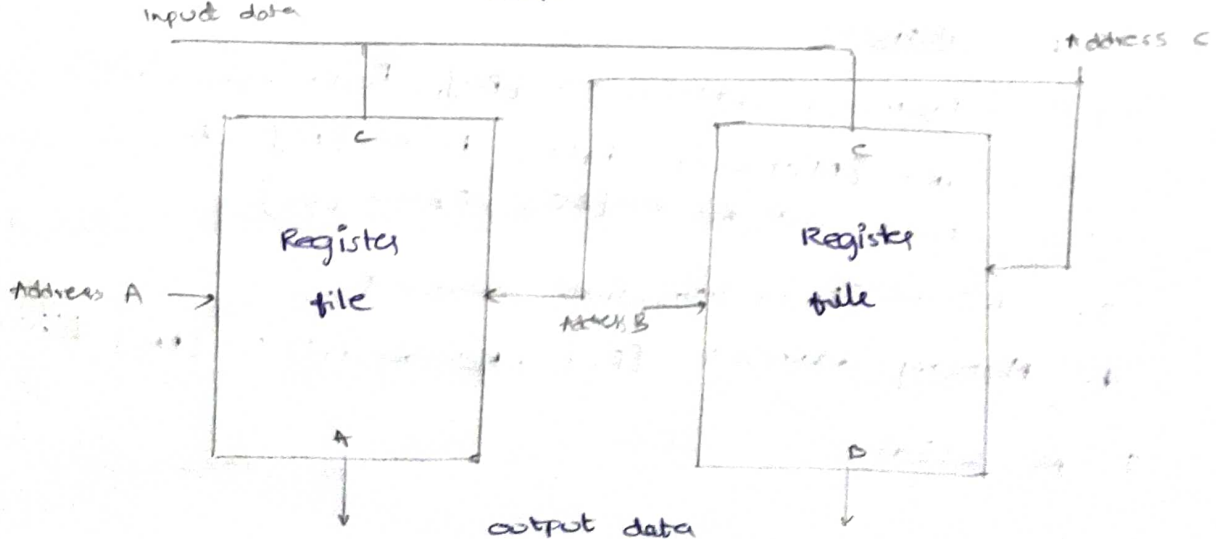
5. Write the result into the destination register, if needed.

Hardware components:-

Register file:-

- A 2-port register file is needed to read the two source registers at the same time.
- It may be implemented using a 2-port memory





18/03/2021

Instruction fetch & Execution steps:-

Sequence of actions needed to perform Add R_3, R_4, R_5

Step	Action
1.	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2.	Decode instruction, RA ← [R_4], RB ← [R_5]
3.	RZ ← [R_A] + [R_B]
4.	R_Y ← [R_2]
5.	R_3 ← [R_Y]

Sequence of actions need to fetch & execute the instruction Load R_5, X(R_7)

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2.	Decode instruction, R_A ← [R_7]
3.	RZ ← [R_A] + Immediate value X
4.	Memory Address ← [RZ], Read memory, R_Y ← Memory data
5.	R_5 ← [R_Y]

Sequence of actions needed to fetch & execute

Store R_6, X(R_8)

Step	Action
1.	Memory address ← [PC], Read memory, IR ← [Memory data, PC ← [PC] + 4
2.	Decode inst, RA ← [R_8], RB ← [R_6]
3.	R2 ← [RA] + Immediate value X
4.	Memory address ← [R2], Memory data ← [RM], Write memory
5.	NO action.

Branching:-

Unconditional:-

- Branch instructions specify the branch target address relative to the PC.
- A branch offset given as an immediate value in the instruction is added to the current contents of the PC.
- Branch offset:- Distance between branch target and memory location following the branch instruction.

Steps for implementing an unconditional branch instruction:-

Step	Action
1	Memory address ← [PC], Read memory, IR ← Memory data, PC ← [PC] + 4
2	Decode instuction
3	PC ← [PC] + Branch offset
4	No Action
5.	No Action.

Conditional:-

- Branch target: Distance between the branch target and memory location following the branch instruction
- Branch instruction specifies a compare-and-test operation that determines the branch condition
- Branch_if_[R5] = [R6] loop
- Results in a branch if the contents of registers R5 & R6 are identical
- Comparision by subtraction.

Steps of actions needed to fetch & execute instruction: Branch_If - [R5] = [R6] LOOP

Step	Action
1.	Memory address ← [PC], IR ← Memory data, PC ← [PC] + 4
2.	Decode instruction, RA ← [R5], RB ← [R6]
3.	Compare [RA] to [RB], If [RA] = [RB]
4.	No action
5.	No action

Subroutine call instructions:-

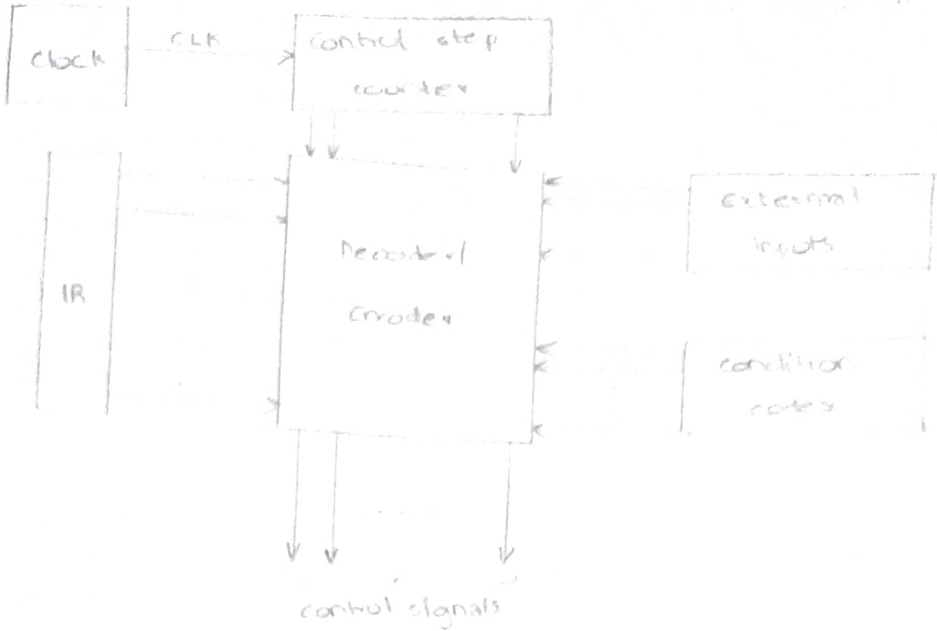
- Subroutine calls & returns are implemented in a similar manner to branch instructions.
- The address of the subroutine may either be computed using

19/03/2014

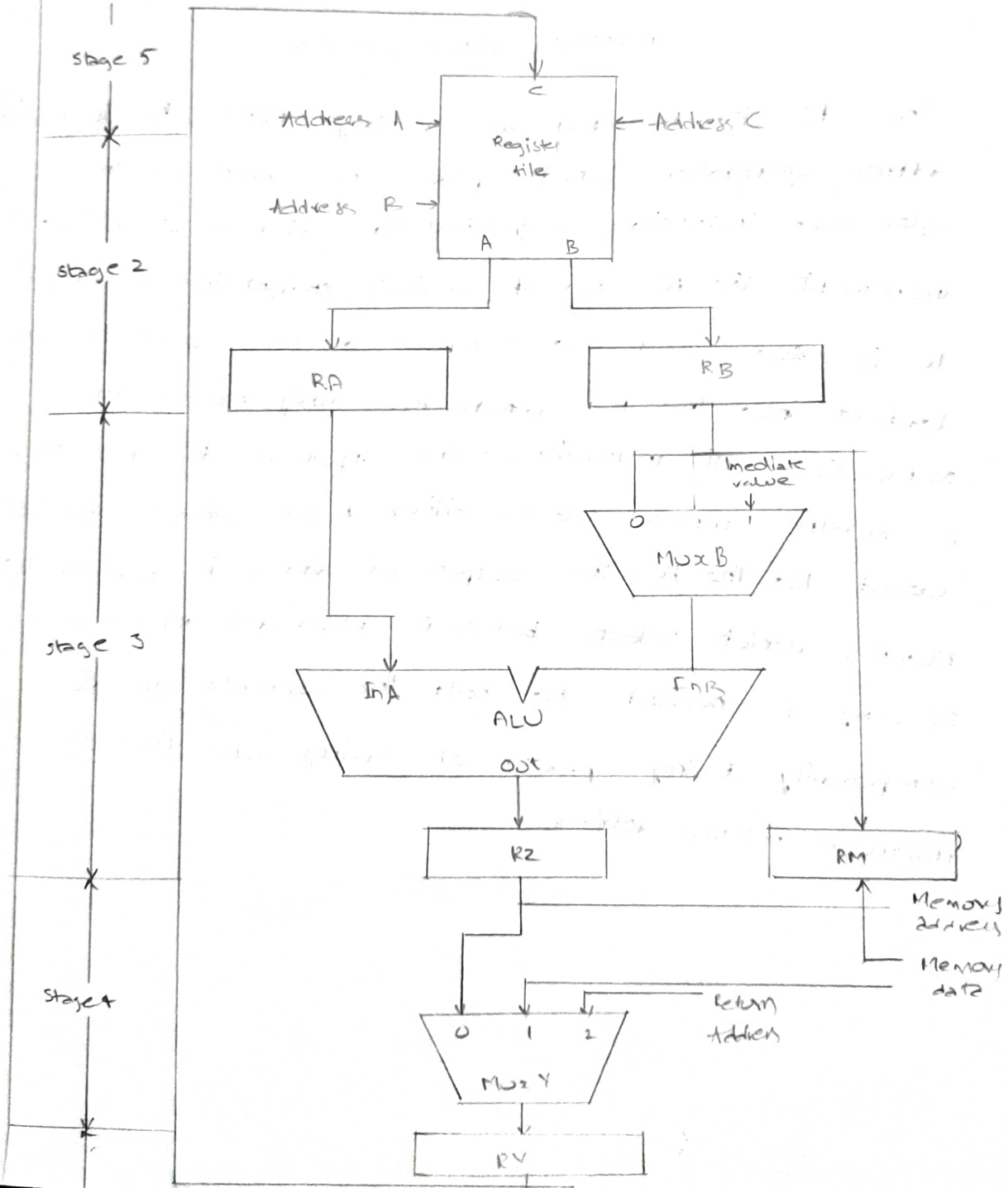
Control unit

To execute instructions the processor must have some means of generating the control signals needed in the proper sequence. The required control signals are determined by the following:-

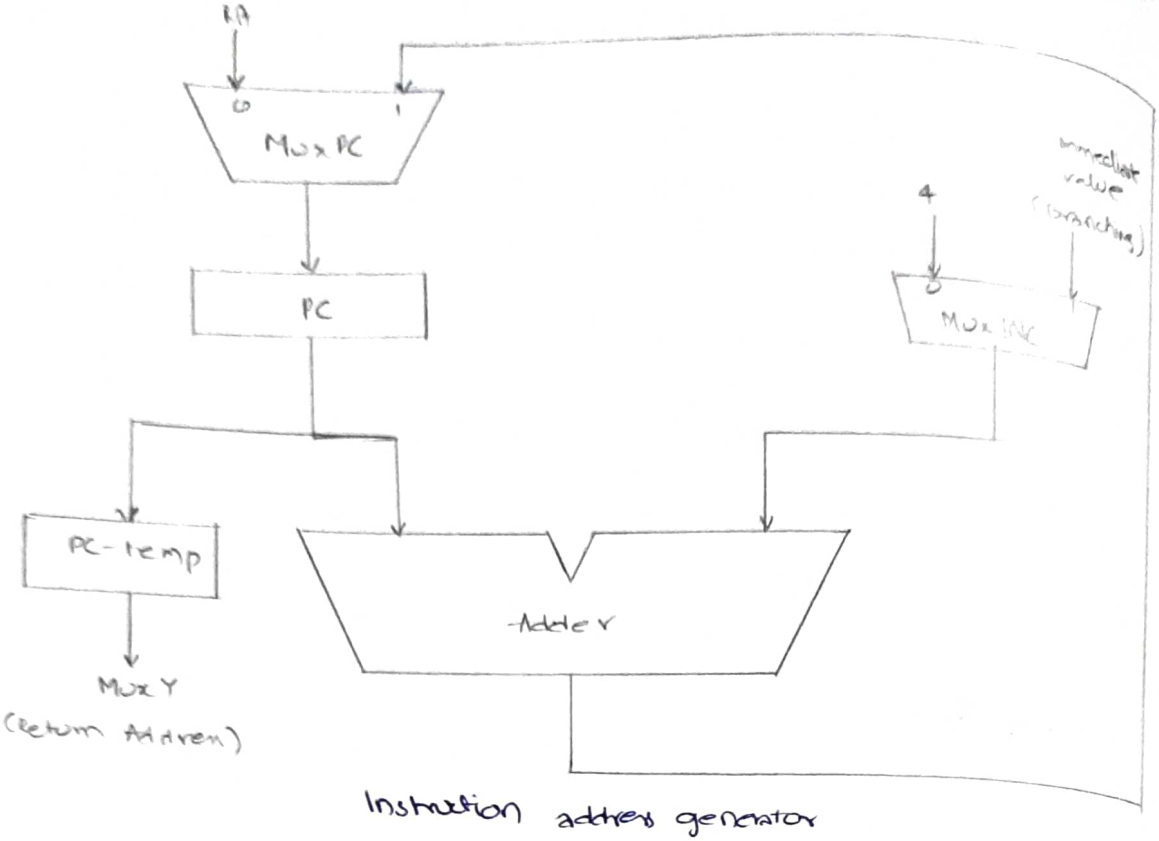
1. Contents of control step counter
2. Contents of instruction register
3. Contents of condition code flags
4. External input signals, MFC & interrupt requests


clock
CLK
control step counter
IR
Decoder / encoder
external inputs
control signals

Datapath:-


stage 1
stage 2
stage 3
stage 4
stage 5
Address A
Address B
Address C
Register file
A
B
RA
RB
Immediate value
0
1
MUX B
InA
InB
ALU
Out
RZ
RM
Memory address
Memory data
Return Address
0
1
2
MUX Y
RY

Instruction Fetch section :-



Instruction address generator

The PC is included in a large block, the instruction address generator, which updates the contents of the PC after each instruction is fetched. An adder is used to increment the PC by 4 during straight-line execution. It is also used to compute a new value to be loaded into the PC when executing branch and subroutine call instructions. One input is PC and other is MUXINC which selects either 4 or offset to be added to the PC. The output of adder is routed to MuxPC, which selects between adder and RA. Register PC-temp is needed to hold the contents of PC temporarily during process of saving subroutine or interrupt return address.

Waiting for memory:-

. Modern computer use fast, on chip cache memories
. Most of the time, the instruction or data referenced in memory Read and write operations are found in cache, in which operation is completed in 1 clock cycle.
. When it is not to be fetched from main memory, several clock cycles may be needed.
. The interface circuit must inform the processor's control circuitary about such situations.
. The processor-memory interface circuit generates a signal called Memory Function Completed (MFC)
. It asserts that signal when a requested memory Read or write has been completed.

Memory address ← [PC], Read memory,
Wait for MFC, IR ← Memory data, PC ← [PC] + 4