EE2016 Microprocessor Theory & Lab
Fall, 2024

Week8: Interfacing Peripherals: Polling, Interrupts & DMA. Part II Interrupts & DMA

Dr. R. Manivasakan. OWSM Lab, IIT, Madras

Organization

- Interfacing CPU with I/O or memory or any peripherals
  - Memory mapped
  - Port mapped
- Memory mapped I/O
- Memory interface
- Peripheral interface
- KB Interfacing
- Polling
- Interrupt
- DMA

Interfacing Peripherals with CPU

- Recap
  - Till now
    - From FF to memory block
      - Memory block representation with data, address and control buses
      - Memory along with ALU
    - Execution Unit
      - Internal registers, status registers, multiplier, shifter etc, along with ALU formed EU. Each sub-engines studied in detail
      - Studied EU along with the first instruction MOV, then LDI
      - Studied many example assembly programs
    - Control Unit
      - PC, Stack, SP, Instruction Decoder, hardwired control unit implementation
      - Studied in detail, each sub-engines
  - Together EU with CU formed microprocessor
    - Studied interfacing peripherals through MMIO (Exp 8)
      - Step 1 Addressing
      - Step 2 Handshake: Polling in KB, as example
- What next?
  - Interfacing peripherals, 2nd step interrupt (KB example) & DMA
- Interrupt
- DMA

Introducing Interrupt

- For polling based key-press detection mechanism, the processor needs to continuously poll every msec to detect any key press
  - This keeps the processor busy all the time – even when the keys are not pressed at all for minutes to hours
- Alternative
  - Processor is **Interrupted** when key is pressed (when any of the Ammeters detect current flowing)
  - The keyboard controller nudges the processor to indicate that a key has been pressed – KB controller is proactive here
  - Once the processor receives this interrupt, it then initiates a scan of all the registers in the keyboard PIC.

What is this Processor Interrupt?

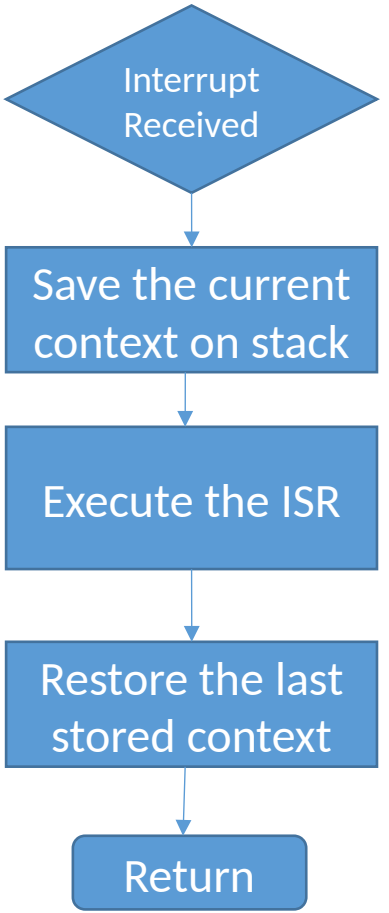
- Processor Execution Unit or Control Unit discussed till now, had till nothing called Interrupt
  - Interrupt will require change in Control Unit (CU)
- An external signal (from PIC) interrupts the Instruction flow-control unit
  - The current Instruction in execution is completed
  - But, then the processor jumps (branch) over to start executing instructions from an “Interrupt” subroutine (instead of currently executing program)
  - Once the task in the Interrupt subroutine is completed, the control unit returns to continue the Interrupted program
    - But this will require the point of interruption to be saved during interrupt and restored during return (in some predefined memory location called STACK)
      - Both the Instruction Pointer as well as all the registers that the subroutine may unwittingly alter, flags are collectively referred to as context of the program

Interrupt: Advantages

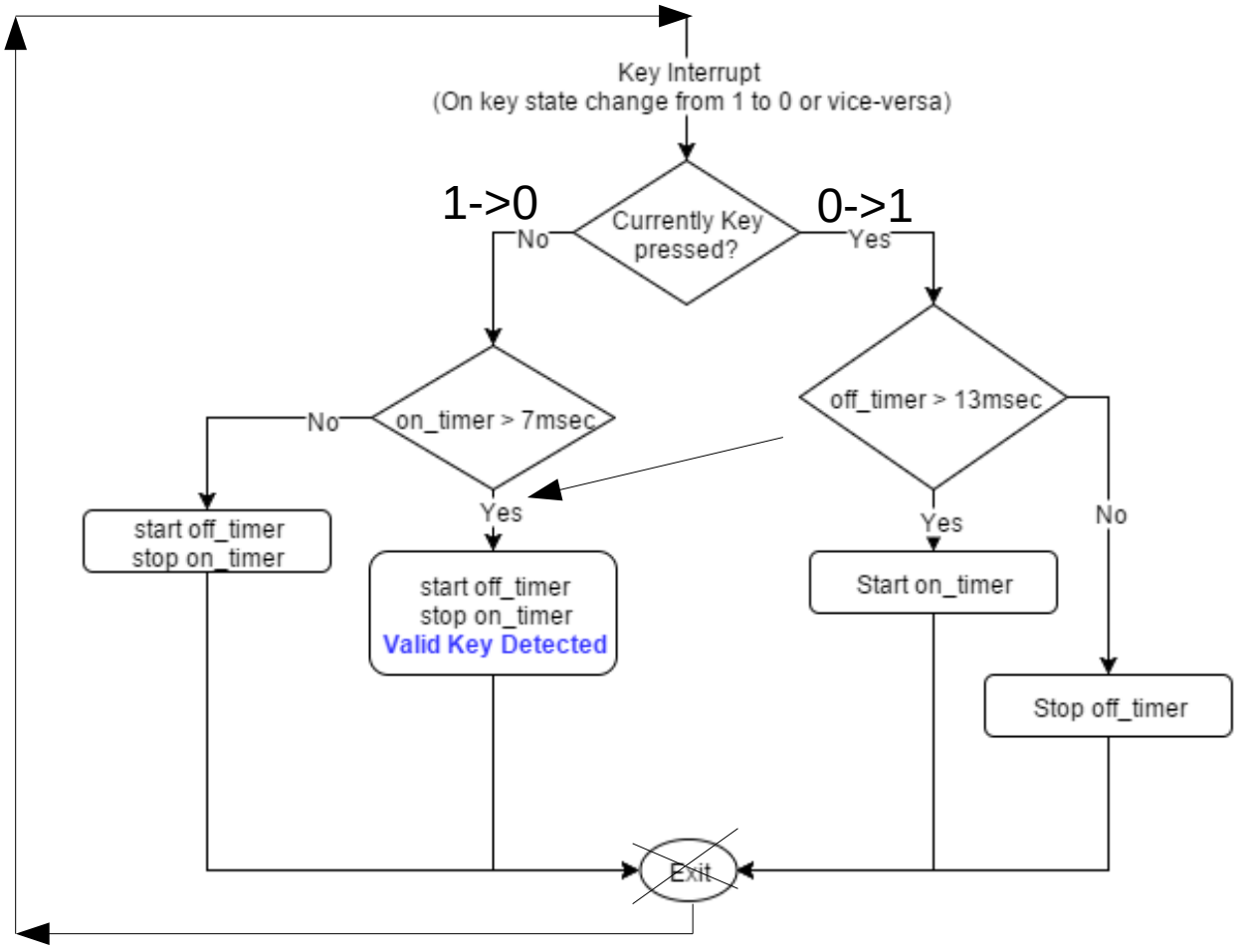
- Advantages of interrupt
  - CPU time is saved as the peripheral is proactive in intimating the CPU that there is an interrupting event
  - CPU energy is saved, since its energy (power) could be spent constructively in executing other programs waiting to be executed
  - CPU throughput (number of instructions executed per sec) is increased.

General Interrupt handling

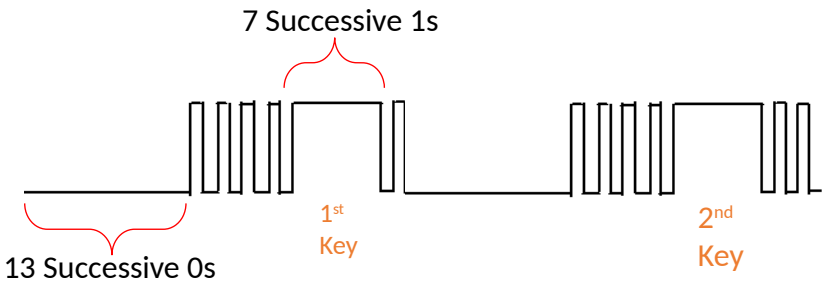
- Interrupt Signal interrupts the processor
  - Can be used by any PIC to draw attention to a task (avoids unnecessary polling)
- Every interrupt has a Interrupt Service Routine (ISR) associated with it.
  - This routine is called whenever an interrupt is received by a processor
- Before executing the ISR, the processor saves the current context: program counter value, Flag, some current register Values
  - This is stored in a stack
- On completion of ISR, the last stored context is restored: populate the PC, flags and saved registers



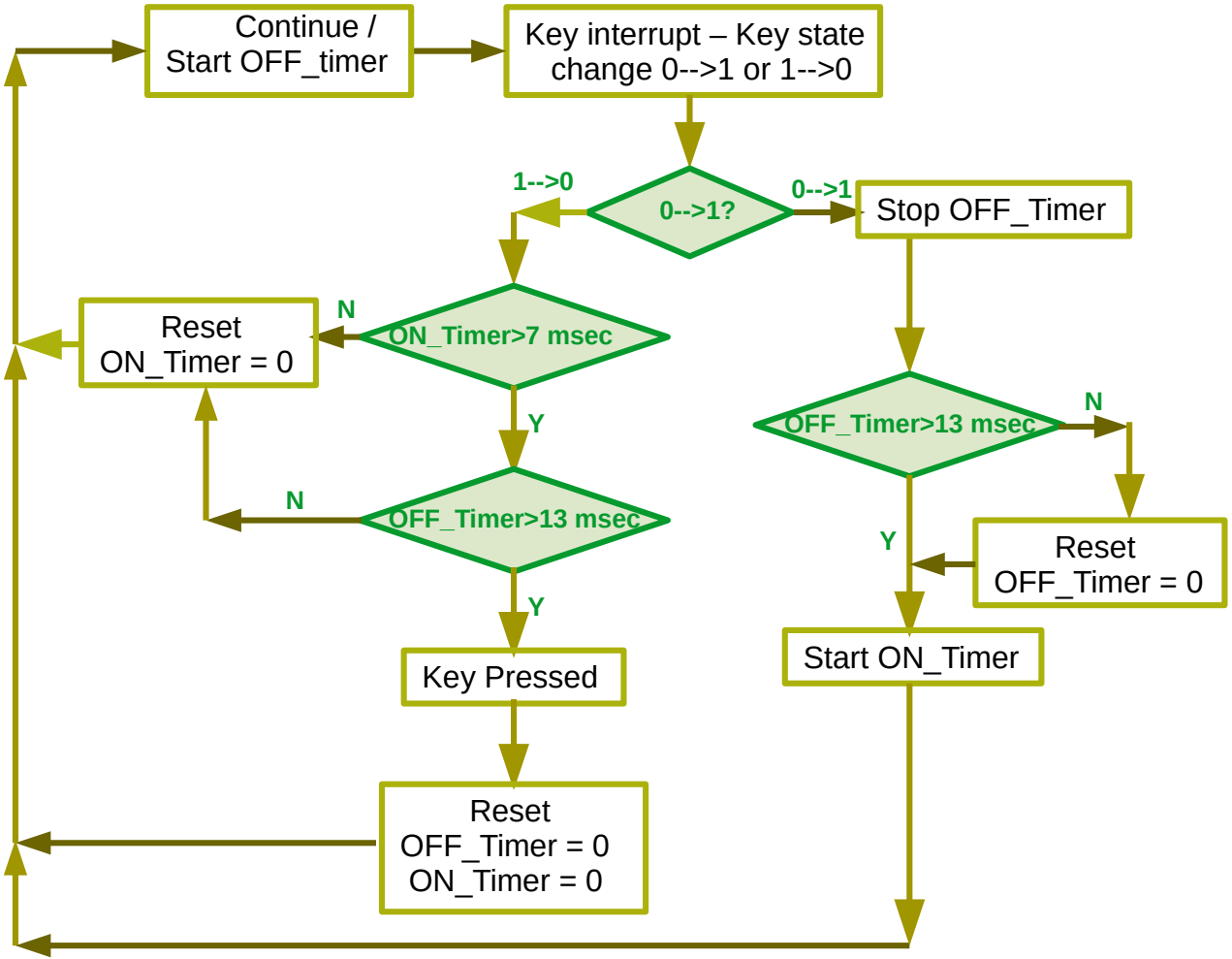
Key Press Detection Using Interrupt



Flow Chart for Interrupt

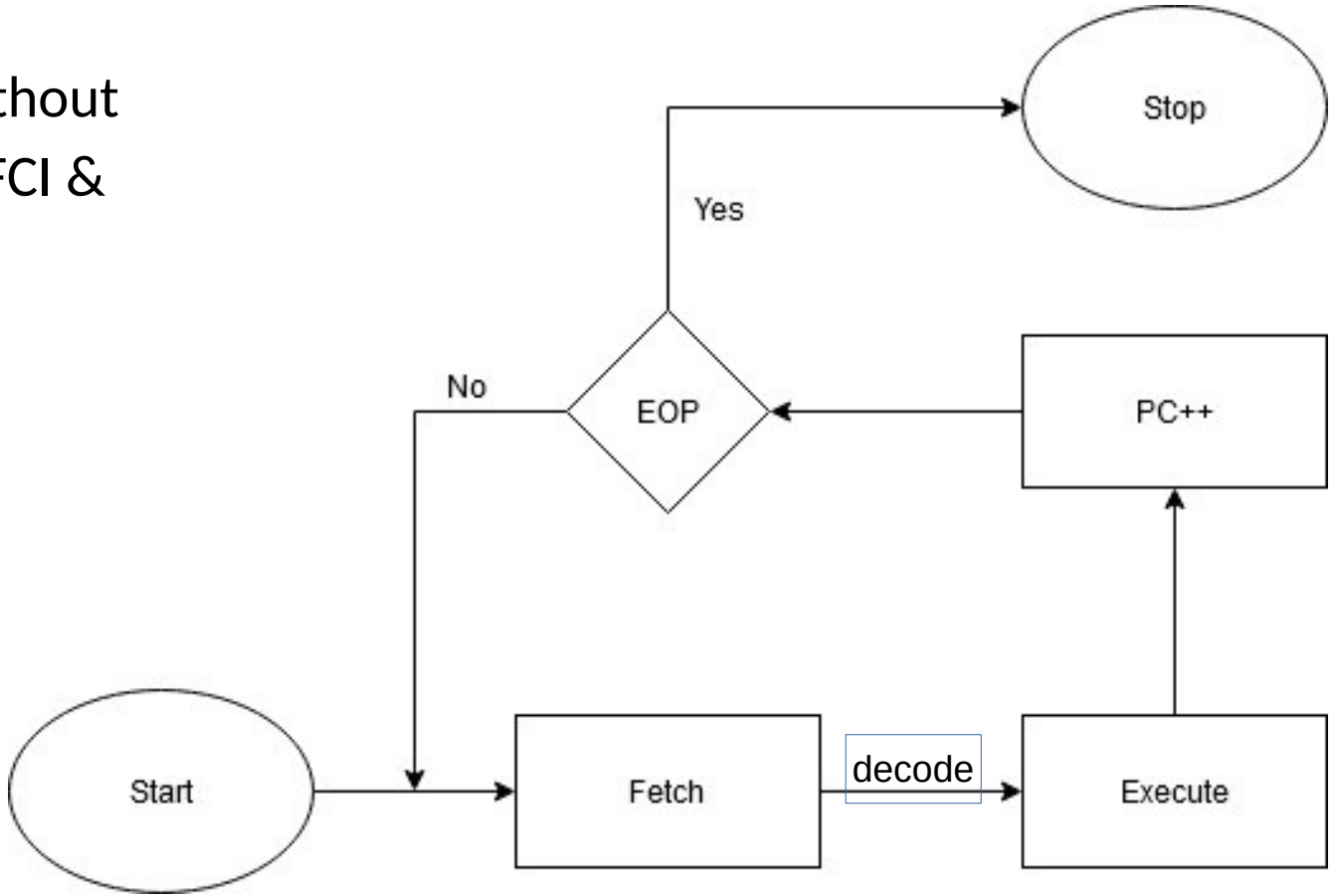


Required Behavior On Consecutive Key Press



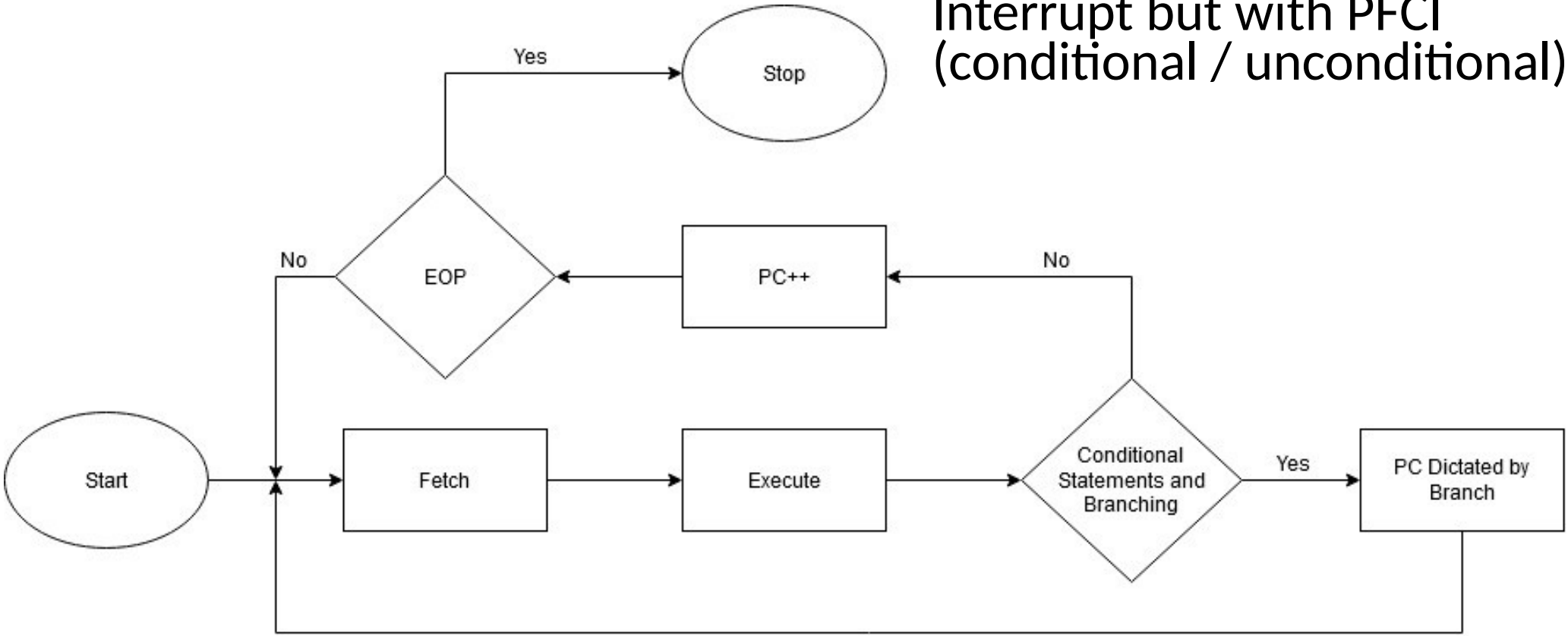
Fetch – Execute Cycle

Fetch – Execute cycle without Interrupt and without PFCI & without ‘Subroutines’



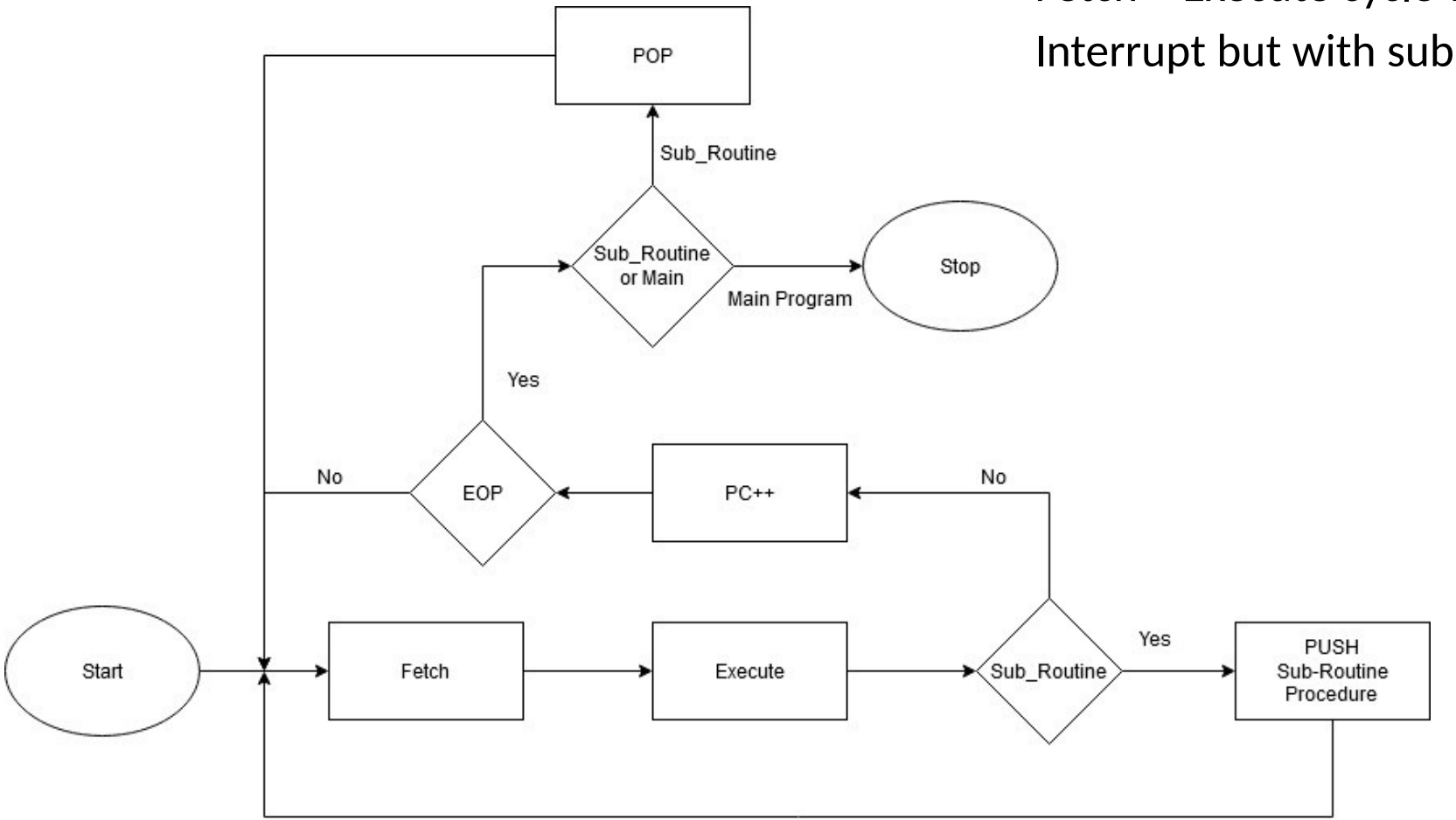
Fetch – Execute Cycle

Fetch – Execute cycle without Interrupt but with PFCI (conditional / unconditional)

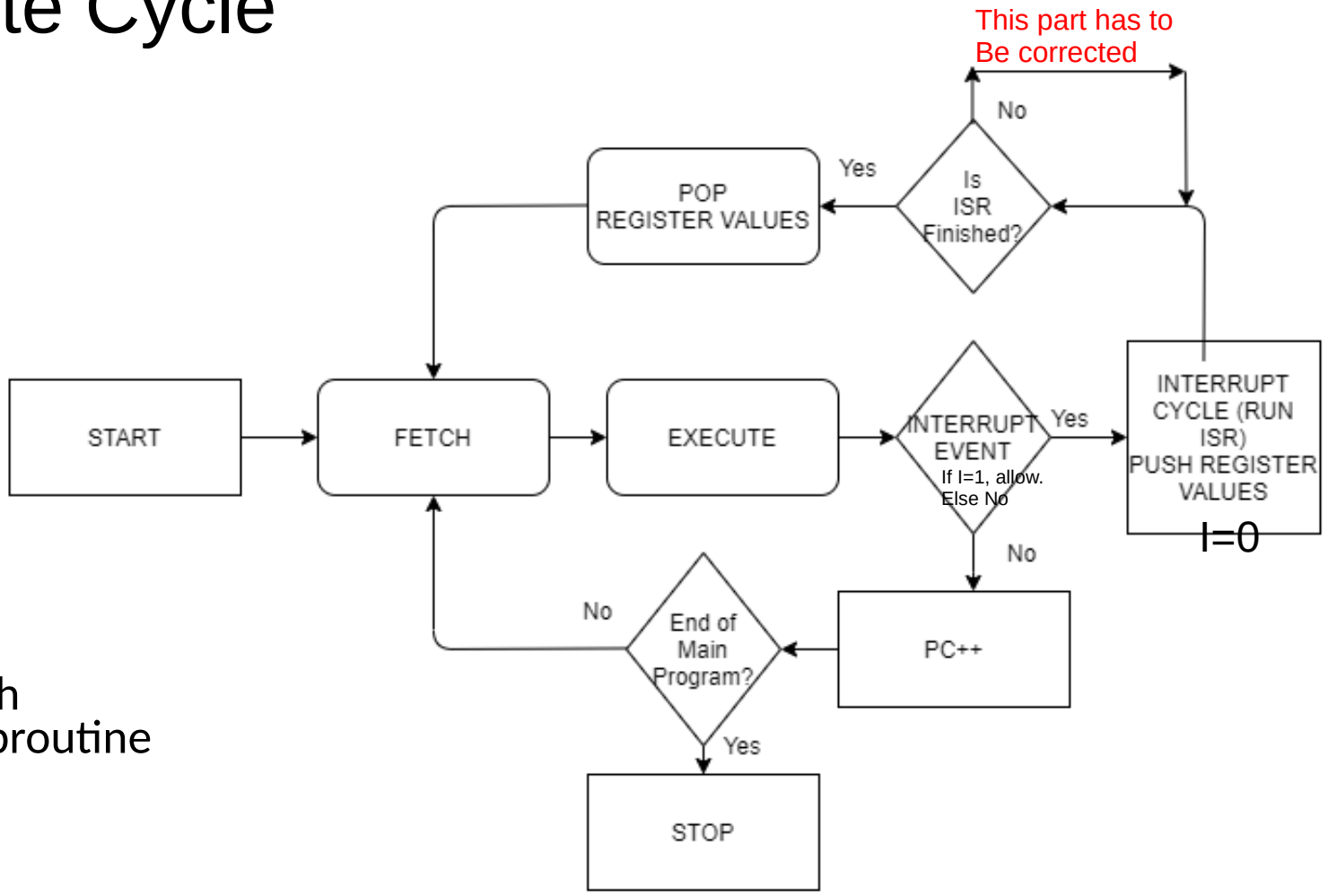


Fetch – Execute Cycle

Fetch – Execute cycle without Interrupt but with subroutine

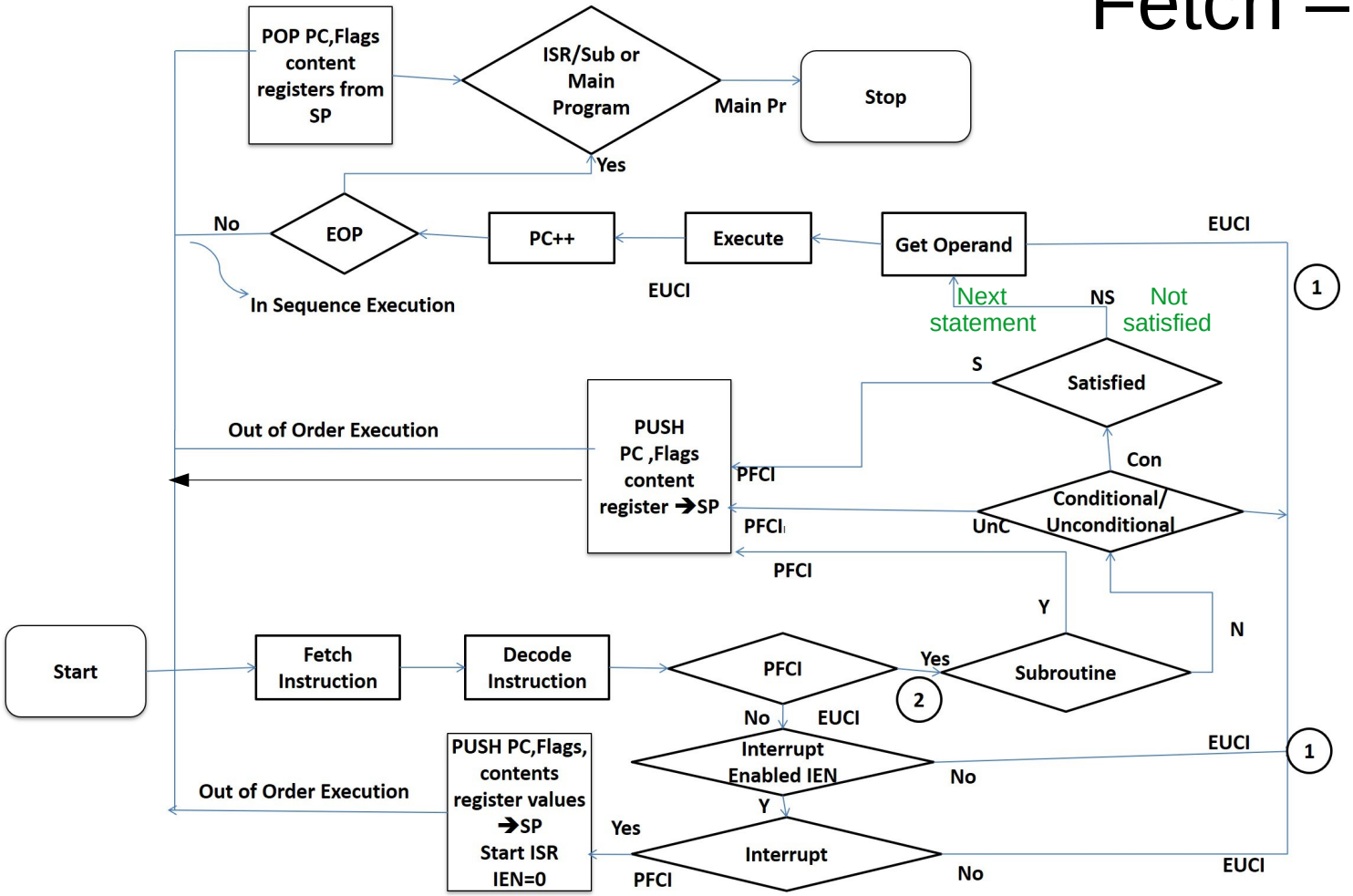


Fetch – Execute Cycle



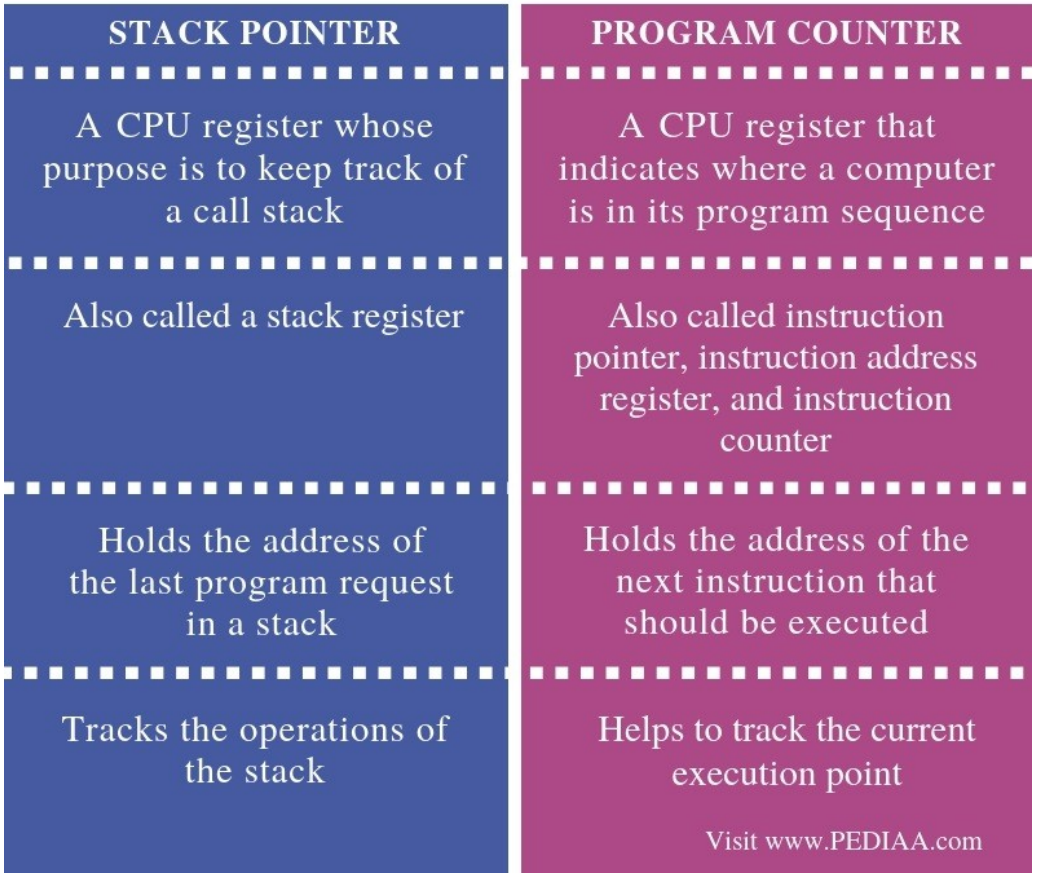
Fetch – Execute cycle with
Interrupt but without subroutine

Fetch – Execute Cycle



- Fetch – Execute cycle with Interrupt & subroutine

STACK POINTER VERSUS PROGRAM COUNTER



Stack and Stack Pointer

- Stack Memory
  - A piece of memory which could be external SRAM (a part of main memory) or could be internal memory, whose access is restricted / different than that of conventional way of accessing
  - Stack memory is a subset of memory, in which () one end (say, TOP – address of its active part) is fixed, the other (say BOTTOM, address of its active part) could be variable.
  - When there is no interrupt being served, then stack pointer points to the top end.
  - As and when the Interrupt arrives, the (a) PC values (b) flag /status register values (c) context registers are PUSHed into the stack memory from top, in the arrival order and correspondingly the stack pointer also decremented as shown in illustration.
  - Interrupts can be nested also
  - When the top most (inner most in the nest) interrupt (which is currently getting served), when the corresponding ISR is served, then the above three values are POPped up while the stack pointer moves up.
- Stack Pointer
  - 2 byte pointer pointing to the variable end of the active part of stack memory
- Assembly syntax
  - PUSH
    - The next instruction to be executed is 'pushed' in
  - POP
    - The stored instructions are "Pop"ped out in the order opposite to the order in which it was stored
    - --> First IN, last OUT queue
  - Used during the interrupt and subroutine part of the program

INT 0

03 – Context register value
02 – Status register value
01 – PC value

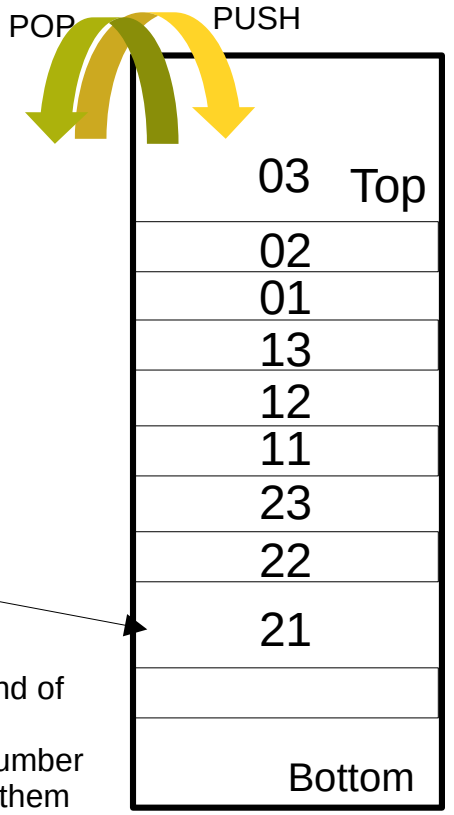
INT 1

13 – Context register value
12 – Status register value
11 – PC value

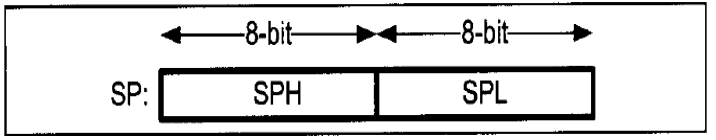
INT 2

23 – Context register value
22 – Status register value
21 – PC value

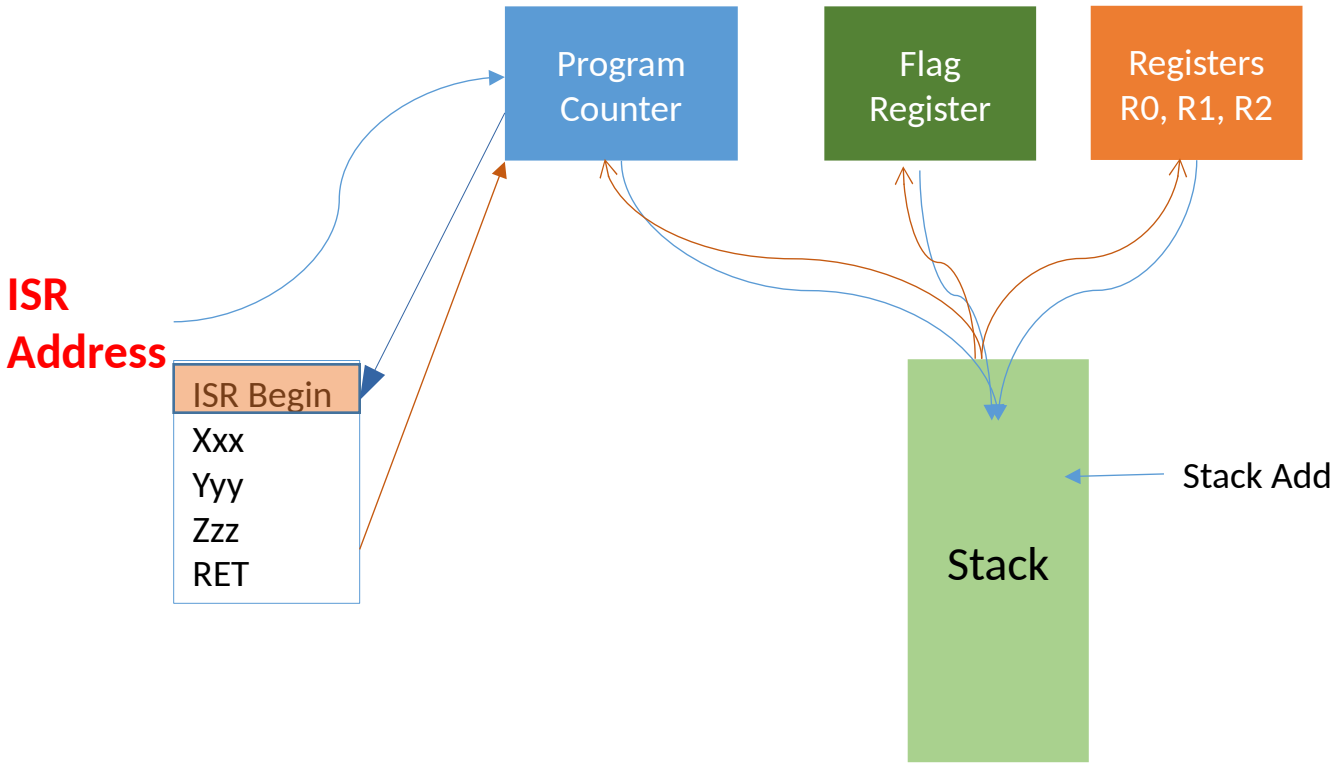
$5C	$3C	OCR0
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG



- Stack pointer ~ address of variable end of stack
- Another byte needed to denote the number of memory locations needed to store them
- (address, 0, 2, 3)



Interrupt Process Flow



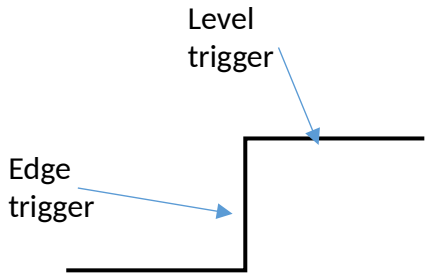
- On Interrupt, the processor PUSHES the current context to Stack

INT

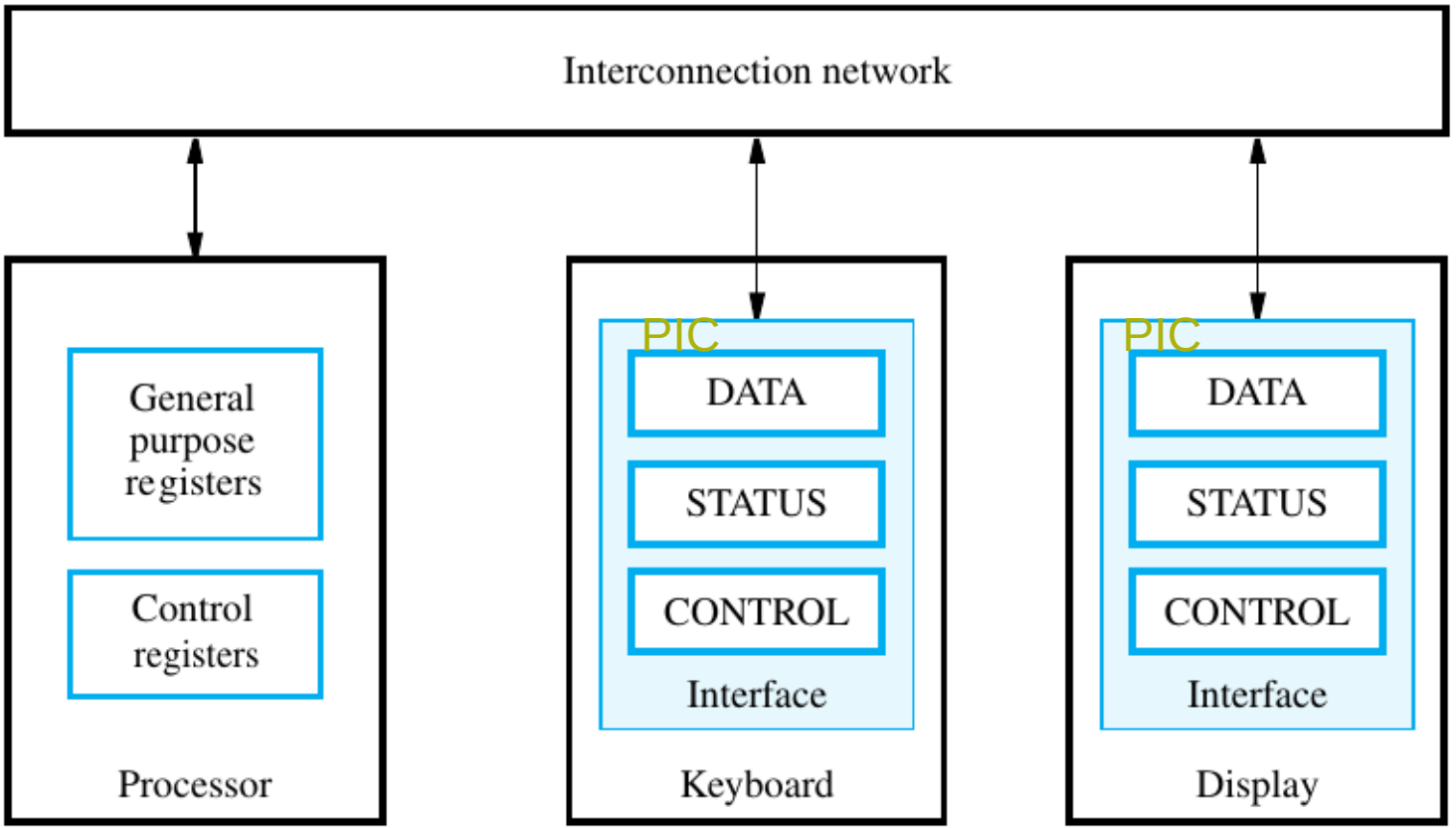
- On Return, the processor POPS the last saved current context out of Stack
- Stack is Last IN First OUT memory
  - Stack Address is incremented on Push and decremented on Pop
- Time taken for interrupt handling: Time for context switch + Time taken to execute the ISR

Nested Interrupts

- A Processor may have multiple Interrupts
  - Associated with multiple ISR tasks, triggered by different external events
  - An Interrupt may interrupt an ISR: Nested Interrupts
    - Ongoing ISR is interrupted with PC and context pushed into stack; interrupting subroutine will be executed and the previous ISR will be continued
    - Stack will grow further to accommodate multiple contexts and shrink when ISR exits
- Different interrupts may have different priorities
  - Only an higher priority interrupt may interrupt an ongoing ISR
    - Equal and lower priority interrupt will be kept pending till earlier ISR exits
- Interrupt may be edge triggered or Level triggered
  - Edge triggering for immediate response



Interconnecting Processor, Keyboard & Display

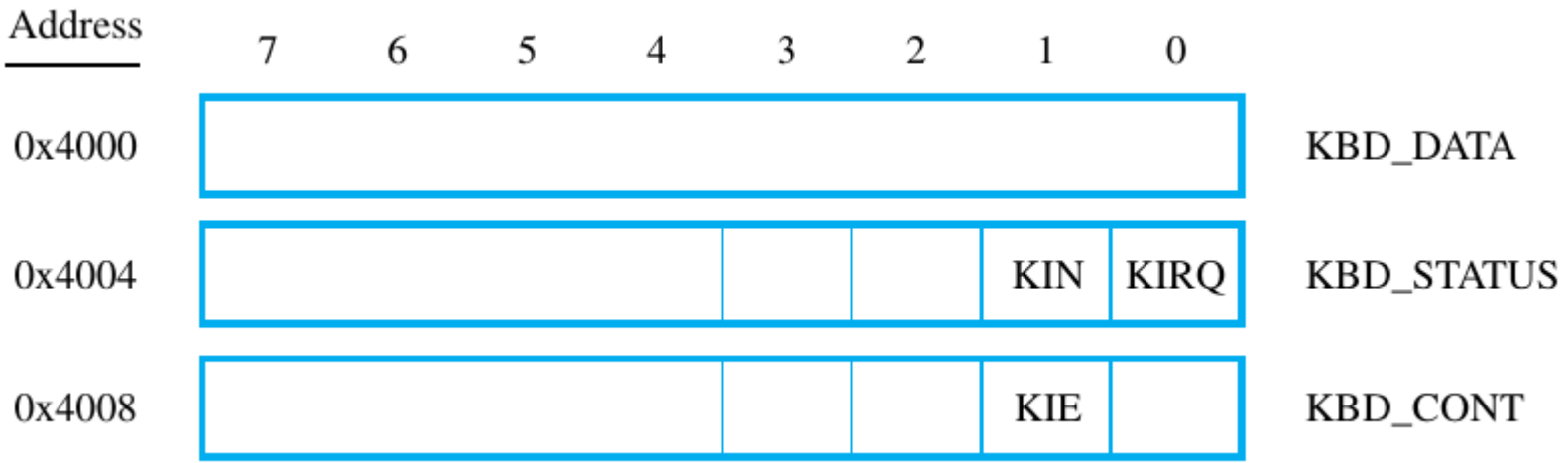


Keyboard PIC Registers

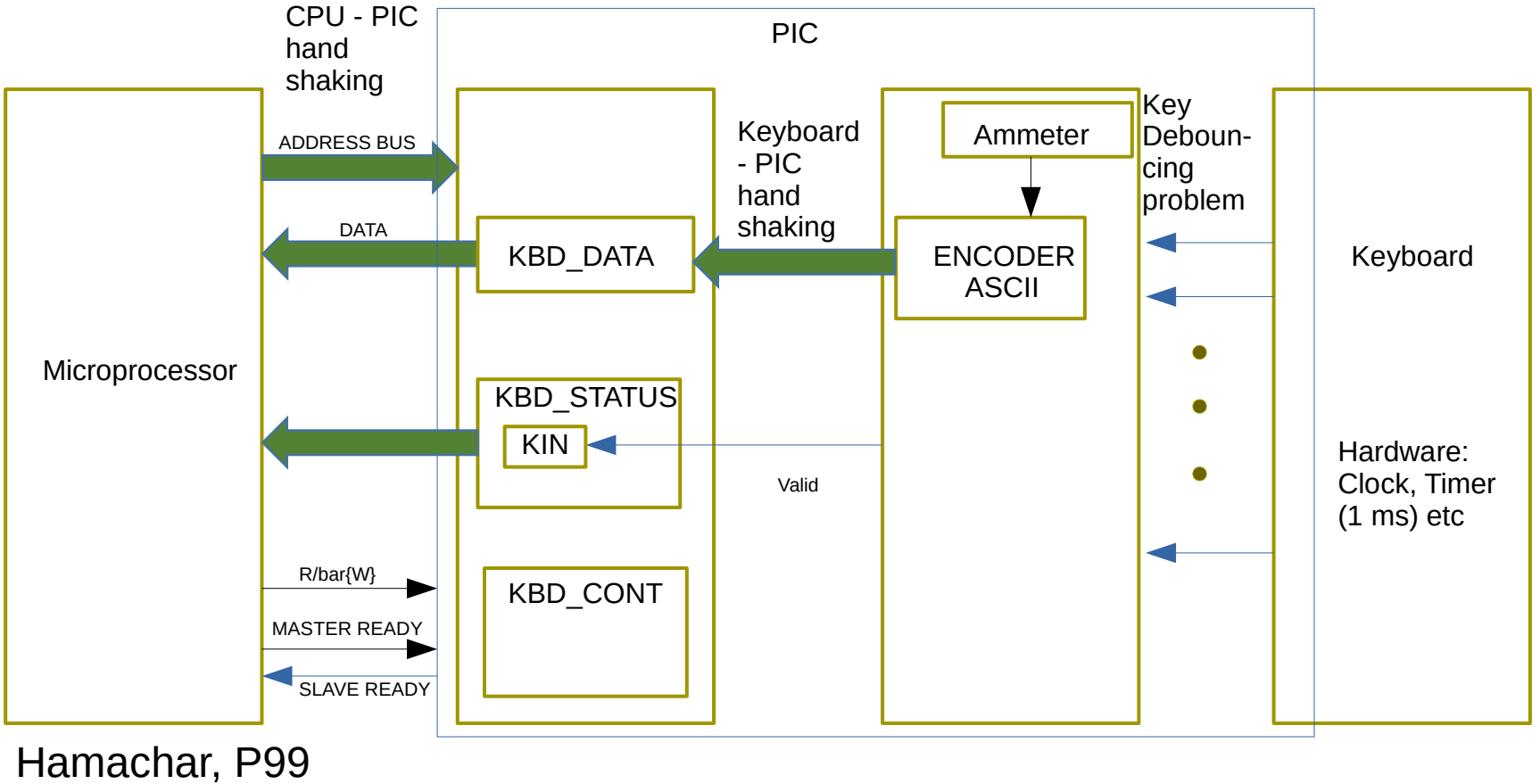
Hamachar, P99

- KBD_DATA
  - Data register holding ASCII code of the key pressed latest, in the keyboard
- KBD_STATUS
  - Status of the keyboard – meaning whether the key pressed has already read by the CPU (& hence the data in the KBD_DATA is invalid)
  - KIN – Keyboard input status bit, set to 1 if ASCII code word corresponding to the key pressed is yet to be read by CPU
  - KIRQ - The KIRQ bit is set to 1 if an interrupt request has been raised, but not yet serviced

- KBD_CONT
  - Keyboard control register – controls or configures the keyboard function
  - KIE – bit number 1 in KBD_CONT
    - Keyboard interrupt enable (KIE)
    - Interface the keyboard to the CPU through interrupt
- Keyboard Interfacing through MMIO

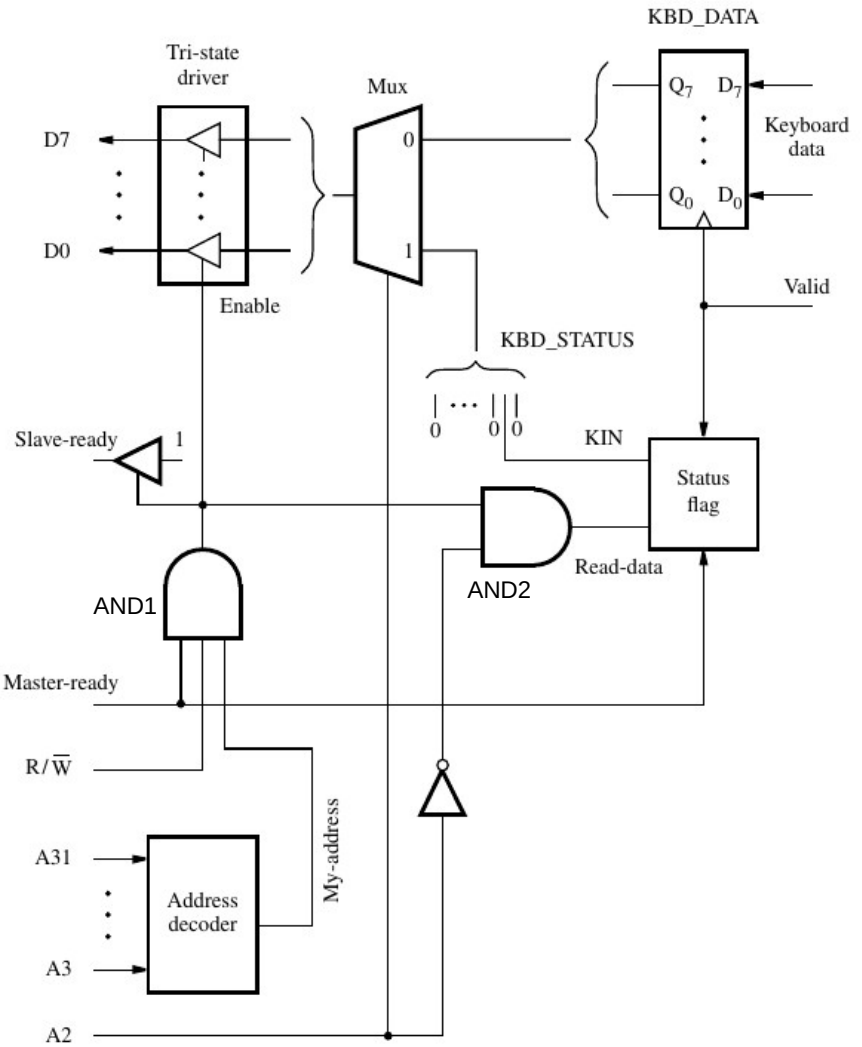


Key Board Interface



Hamachar, P99

Keyboard Interfacing Circuitry



Outline

- Pros & cons of Polling & Interrupt
- Motivation for DMA
- DMA

Motivation: DMA

- Steps to interface CPU with peripherals
  - 1. Addressing the peripheral (to pick the particular peripheral out of many peripherals out there)
  - 2. After picking up the peripheral, (within the session) how to handshake (how the communications to peripherals happens – hardware protocol part)
    - Polling
    - Interrupt
    - DMA (Direct Memory Access)
- Polling
  - Periodic polling by CPU
  - Advantages
    - Simplicity of implementation
  - Disadvantages
    - CPU spends a lot of time & resources to check whether the peripheral needs CPU's attention
      - Esp when the peripheral is active ONLY for a very small fraction of time
- Interrupt
  - Advantages
    - CPU can do its duty, only when the peripheral needs its attention, peripheral can "interrupt".
  - Disadvantages
    - Algorithm is involved
    - Still CPU is needed (In the case of memory transfer, each block of memory transfer has to pass through CPU)
- Motivation
  - Is it possible to bypass CPU completely? For the entire duration of memory transfer?
- DMA: Definition
  - A technique for transferring data from memory to IO device and vice-versa without passing through processor
- Implementation
  - With DMA, the processor is freed from involvement with the data transfer, thus speeding up overall computer operation. How?

DMA

- DMA vs other methods

Without DMA	With DMA
A processor is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work	The processor initiates the transfer, does other operations while the transfer is in progress, and receives an interrupt from the DMA controller when the operation is completed
CPU is still the master of the internal & external bus.	The CPU though it relinquishes the system bus, it does some operations which uses only the internal bus. CPU operations involving the system bus are suspended.

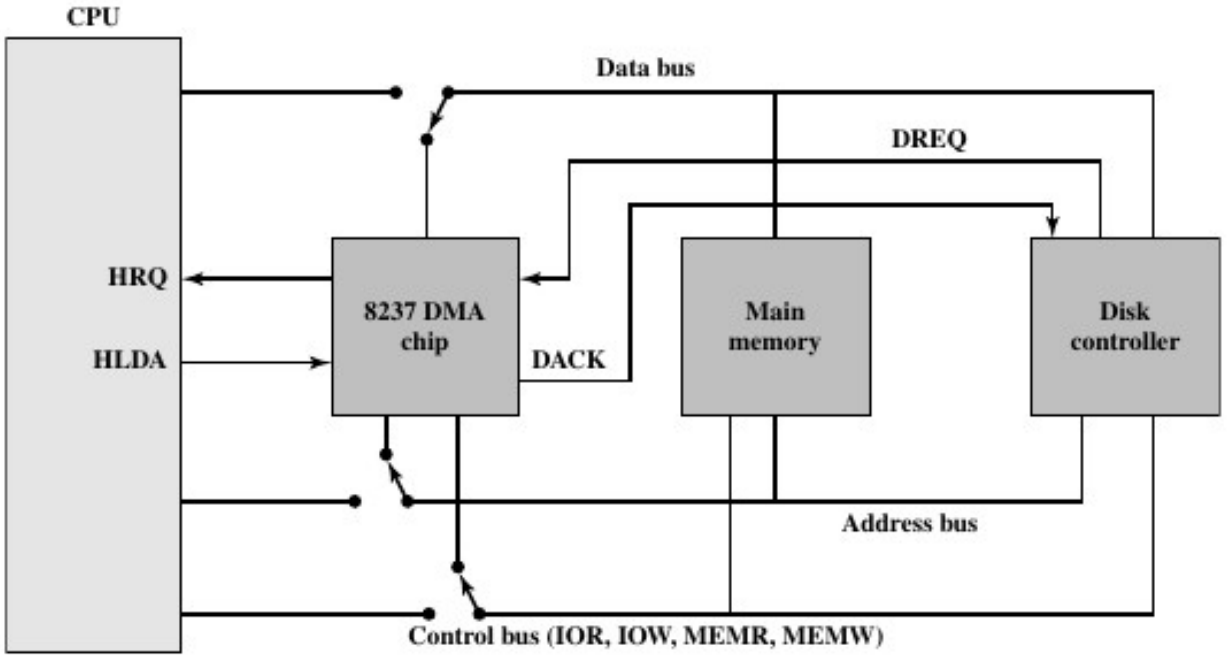
Concept of DMA

- Normally, a processor is the bus-master
  - It is always the processor which controls the Address and the Control Bus and therefore decides what is to be done (type and speed of data transfer) on the bus
  - Large memory transfers are realized by splitting into small chunks of memory, transferring them serially and checking for errors.
  - The above keeps the microprocessor busy, since it has to take care of transfer of each chunk of memory without error
- The Concept of DMA is to create an alternate bus-master
  - This alternate master is called DMA Controller (DMAC) – a special hardware subsystem
  - DMAC has to emulate the muP for large memory transfers
    - DMAC has to get full control over system bus temporarily, from the muP --> DMAC is the temporary system bus master
    - DMAC has to monitor the transfer of every chunk of memory is error free
- For this to work, both the processor and the **DMAC** should have access to memory and IO Registers via a shared system bus having data, address and control lines???
-

How does DMA Control work?

- DMAC can temporarily take over the bus as bus-master, by making a request to the processor (DMA-Request)
  - and the processor granting the bus to DMAC using REQUEST-GRANT
- DMAC is now the temporary bus-master as processor relinquishes (its control over address and control bus)
  - DMAC now controls the address bus and control bus and uses it to transfer the data bytes directly between an I/O Registers and a series of memory locations
- The bus-cycles (RD/WR cycle) for this transfer may typically be much faster than the processor bus-cycle

How does DMA work?



- On DMA-Req (DREQ) from PIC, DMAC borrows the buses using the pins HOLD and HLDA
  - HRQ - Hold Request is driven by DMAC to generate request to CPU
  - HLDA – Hold Ack is the grant signal given by the processor
  - DACK – DMA Ack, DMAC sends to disk controller
- Immediately after getting HLDA, CPU relinquishes the buses & DMAC gets access to the buses.
  - Disk controller takes charge of memory transfer, with the DMAC doing the job of CPU for the whole memory transfer.
- Applications
  - Disk drive controllers, graphics cards, network cards and sound cards.
  - Intra-chip data transfer in multi-core processors
  - Scatter-gather operations
    - Multiple buffers into single stream
    - Single data stream into multiple parallel buffers
  - Network-on-chip, in-memory computing architectures

Data Transfer Steps

- Processor programs DMAC to accept a DREQ (DMA Request) from PIC and carry out transfer of data between PIC and memory locations (example: PIC to Memory)
  - FROM location (example: PIC) and TO location (Memory Address) are written in DMAC registers as well as WORD-COUNT (number of transfers to be performed) in DC register
- When PIC has data ready for transfer, it asserts DREQ signal on DMA controller requesting for DMA transfer to begin
  - DMA controller sends HLDR signal to CPU (received as HRQ)
  - CPU sends HLDA back to DMA controller
  - DMA controller gives DACK (DMA acknowledgment) back to PIC implying Bus grant has happened
  - DMA controller places memory address on address bus. Memory transfer begins.
  - Every memory block transfer, is routed through DMAC
  - DMA controller updates memory address register (MAR) and word counter register
  - When DC register becomes zero, DMA controller sets HLDR=0 to release the bus
- Data transfer process terminates and processor regains control of the system bus

DMA transfer can be Block transfer or by Cycle-Stealing

Block Transfer

- DMA controller is master of memory bus
- Needed by the secondary memory like disk drives, that have data transmission and are not to be stopped or slowed without any loss of data transfer of blocks
- Pros: Block DMA transfer supports faster I/O data transfer rates
- Cons: Processor remains inactive for relatively long period by tying up the system bus

Cycle Stealing

- In this method, system allows DMA controller to use system bus to transfer one word, after which it should return back control of bus to CPU.
- Pros: It also reduces interference of DMA controller in CPU memory access i.e., CPU can avail system bus at any time.
- Cons:
  - This method reduces maximum I/O transfer rates
  - Involves multiple bus request, grant cycles

CPU's state during DMA Transfer

- a CPU can execute the current program while the DMAC is using the system buses:
  - The CPU fetches an instruction from the cache. It decodes the instruction and determines what data it needs to access.
  - The CPU checks the cache to see if the data is in the cache. If the data is in the cache, the CPU accesses the data directly from the cache.
  - If the data is not in the cache, the CPU generates a bus request to access the data from memory.
  - If the DMAC is using the system buses, the CPU will be stalled until the DMAC finishes its transfer. Once the DMAC finishes its transfer, the CPU will be able to access the data from memory.
- The CPU executes the instruction and stores the result in the cache or in memory.
- By alternating between accessing the cache and accessing memory, the CPU can continue to execute the current program even while the DMAC is using the system buses.

To Conclude

- Polling registers in Peripheral Controller used to detect state of Peripherals
  - Would require frequent polling and waste of CPU cycles
- Interrupt is an alternative means to draw attention to an external event occurring at a peripheral
  - Enables CPU to suspend its current task and handle external event through Interrupt subroutine (ISR) which reads status / data registers of a peripheral controller
  - CPU would resume suspended task after the ISR exits, without any glitch
  - Nested Interrupts and Interrupt Priority used to handle interrupts form multiple external events
- Interrupt Controller chip (like a Peripheral Controller) sometime used to connect multiple interrupts, define priorities for each interrupt and turn on / off any Interrupt

To Conclude

- Polling registers in Peripheral Controller used to detect state of Peripherals
  - Would require frequent polling and waste of CPU cycles
- Interrupt is an alternative means to draw attention to an external event occurring at a peripheral
  - Enables CPU to suspend its current task and handle external event through Interrupt subroutine (ISR) which reads status / data registers of a peripheral controller
  - CPU would resume suspended task after the ISR exits, without any glitch
  - Nested Interrupts and Interrupt Priority used to handle interrupts form multiple external events
- Interrupt Controller chip (like a Peripheral Controller) sometime used to connect multiple interrupts, define priorities for each interrupt and turn on / off any Interrupt

Advantages and Disadvantages of DMA

- Advantages
  - DMA allows a peripheral device to read from/write to memory without going through the CPU
  - DMA allows for faster processing since the processor can be working on something else while the peripheral can be populating memory
- Disadvantages
  - DMA transfer requires a DMA controller to carry out the operation, hence cost of the system increases
  - Cache Coherence problems: to be discussed later

Interrupt versus DMA

S. No	Issue	Interrupt	DMA
1	Principle of operation	The input/ output module / peripheral proactively interrupts the microprocessor, to request service, whenever need arises. The microprocessor or microcontroller has to be confgured accordingly for honoring the interrupts.	In Direct Memory Access (DMA), the CPU grants I/O module authority to read from or write to memory without involvement.
2	Time and power consumed	Interrupt-driven input/ output still consumes relatively considerable time and power because all data has to pass through processor.	DMA consumes less power and less time for a given task of memory transfer as the number of entities are just two: memory and the peripheral (or two memory modules)
3	I/O transfer rate	The I/O transfer rate is limited by the speed with which the processor can test and service a device.	The I/O transfer rate is limited by the speed set by the technologies of the memory and the peripherals.
4	CPU activity	The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.	CPU can do its own job but using only internal bus (as system bus is used by DMA).
5	Amount of data transfer	Smaller amount of transfer, interrupt is not a bad scheme	The very advantage of DMA is reaped when the volume of data transfer is large
6	Special case where CPU is slow	I/O data transfer rate happens at CPU rate	Data transfer can happen at rate higher than that can handled by CPU