# EE2016 Microprocessor Theory & Lab
# Fall, 2024

kturhjnty;oirl/ynk

## Week7: Interfacing Peripherals: Polling, Interrupts & DMA. Part I Polling

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
- What next? Interfacing microprocessor with memory & I/O
  - Step 1: Addressing (already did exp 3 & 4, AVR)
  - Interfacing peripherals, 2$^{nd}$ step polling and interrupt
- Polling
- Interrupt

# MMIO plus polling / interrupt / DMA

- Two problems
  - Addressing a peripheral
  - Once the peripheral is reached, hand shake to talk to the peripheral
- Addressing a peripheral
  - Memory mapped I/O and port mapped I/O
- Hand shake method
  - Polling
  - Interrupt
  - DMA
- Polling
  - CPU queries PERIODICALLY (say every ms), every device whether you have something to send to itself.
  - The peripheral responds whether it has bits to send or not. If it has, then it responds with the corresponding control signals and the payload. If not then it responds with a clear NO.
  - The CPU goes to next peripheral in the polling cycle.
  - The cycle repeats

- Advantages of polling
  - Simpler to implement
- Disadvantages of polling
  - CPU keeps on checking a peripheral even if the peripheral is seldom needs the service of CPU
  - Costly in terms of time and energy
  - inefficient

# Interfacing CPU

- Memory mapped I/O (MMIO)
  - Def: Memory mapped I/O uses the same address space to address both memory and registers of I/O devices. This implies part of whole address (range) space corresponds to memory and remaining part corresponds to all I/O devices.
  - In MMIO, the CPU sees everything as memory locations.
  - An I/O device responds to an address (issued by CPU), if this address falls within the range of addresses assigned to that device
  - Most common type of I/O interface
- Port mapped I/O
  - All peripheral devices (apart from memory) are grouped and is distinguished from memory for interfacing.
  - Works on a dedicated address space for I/O.
  - Each physical port is assigned a code and specific CPU instructions (IN, OUT) are used to access the same.
  - Eg. intel x86 processors.
- Only memory mapped I/O would be discussed in this course

# Memory mapped I/O versus Port mapped I/O

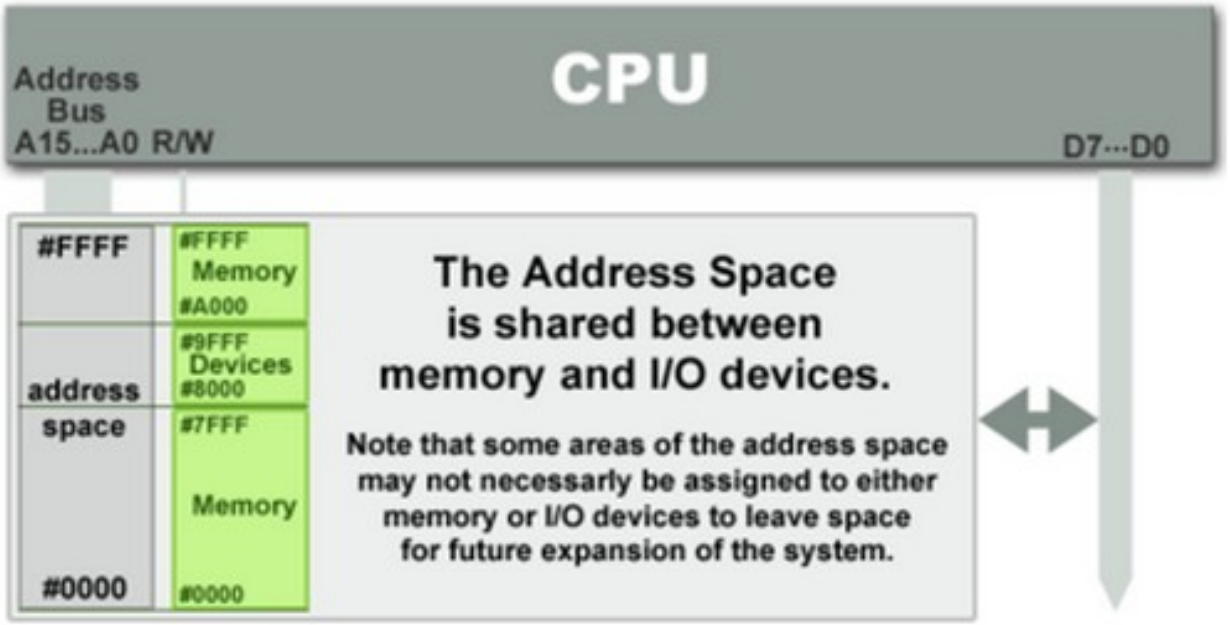
- Port mapped I/O
  - Each physical port is assigned a code and specific CPU instructions (IN, OUT) are used to access the same.
  - Different forms of the above two instructions can copy one, two or four bytes (outb, outw and outl, respectively) between the EAX register or one of that register's subdivisions on the CPU and a specified I/O port which is assigned to an I/O device.
  - I/O devices have a separate, dedicated address space from general memory, either accomplished by an extra "I/O" pin on the CPU's physical interface, or an entire bus dedicated to I/O.
    - Because the address space for I/O is isolated from that for main memory, this is sometimes referred to as isolated I/O
  - Port mapped I/O instructions are very limited, often simple LSA between CPU registers and I/O ports
    - For example to add a constant to a port-mapped device register would require three instructions: read the port to a CPU register, add the constant to the CPU register, and write the result back to the port.

- Port mapped I/O
  - Eg. intel x86 and x86-64 microprocessors
  - INS/INSB/INSW/INSD—Input from Port to String
    - Copies the data from the I/O port specified with the source operand field (second operand) to the destination operand field (first operand).
    - The source operand is an I/O port address (from 0 to 65,535) that is read from the DX register *
- Memory mapped I/O
  - CPU needs less internal logic, hence cheaper, faster, easier to build, consumes less power and less foot print
  - Eg. Almost all microcontrollers, embedded processors, AVR, ARM etc
  - The above follows the basic tenets of RISC & is advantageous in embedded systems
  - All of the CPU addressing modes are available for the I/O as well as memory
  - Only memory mapped I/O would be discussed in this course

# Memory mapped I/O versus Port mapped I/O

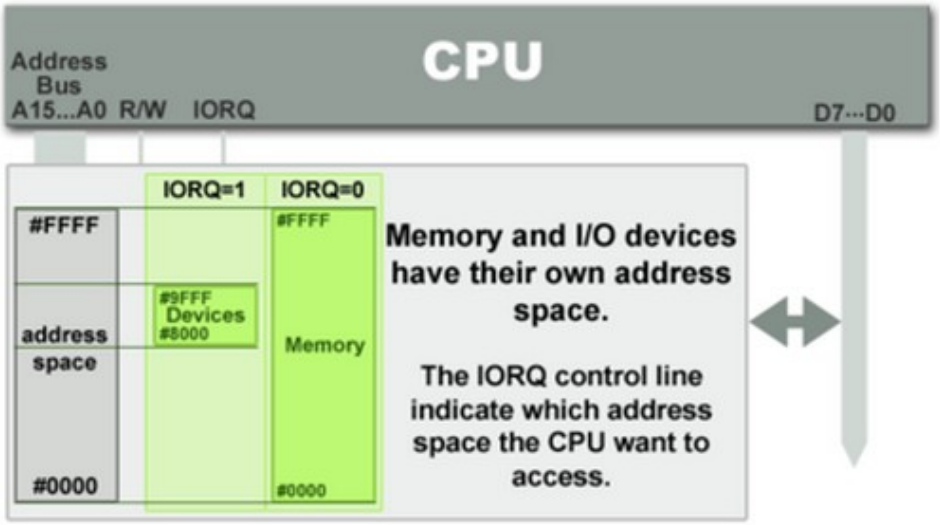
| S. No | Issue | Memory mapped I/O | Port Mapped |
|---|---|---|---|
| 1 | Address space | Address space is shared (a range of address for memory and disjoint range for each of I/O ports) | Address space is dedicated for memory. Another address space for I/O. |
| 2 | Memory & peripherals distinguishability | CPU sees everything as address maps (memory + I/O ports) | CPU distinguishes memory from all remaining peripherals through I/O ports. |
| 3 | Access of memory or I/O devices | An I/O device responds to an address (issued by CPU), if this address falls within the range of addresses assigned to that device. This holds for memory also. | Address bus is accompanied with the control signal IORQ. If IORQ is 0, then memory is accessed. If IORQ is 1, then I/O is accessed. (Address range might overlap – should not matter). |
| 4 | Instructions for access | All instructions which are used for I/O is also used for accessing memory. In all aspects the memory and the I/O are indistinguishable (except for the address range). | Each I/O physical port is assigned a code and specific CPU instructions (IN, OUT) are used to access the same. For memory access normal instructions are used. |
| 5 | Instructions for I/O | All addressing modes are available. | Port mapped I/O instructions are very limited, often simple LSA between CPU registers and I/O ports. All addressing modes are NOT available for I/O ports (through PIC registers). |
| 6 | Examples | Almost all other processors than intel x86, x86-64 | intel x86 and x86-64 microprocessors. |
| 7 | | | INS r/m8,DX source DX port, destination is reg r. |

# Memory mapped I/O



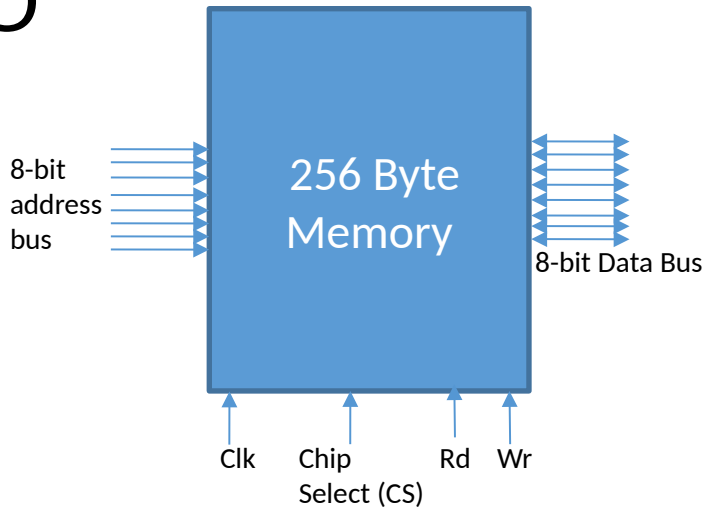
# Port mapped I/O

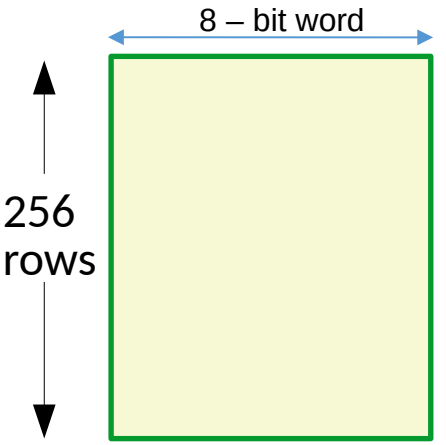
- The advantage to this system is that less logic is needed to decode a discrete address and therefore less cost to add hardware devices to a machine.
- On the older PC compatible machines, only 10 bits of address space were decoded for I/O ports and so there were only 1024 unique port locations;
  - modern PC's decode all 16 address lines.
  - To read or write from a hardware device, special port I/O instructions are used.
- From a software perspective, this is a slight disadvantage because more instructions are required to accomplish the same task.
  - For instance, if we wanted to test one bit on a memory mapped port, there is a single instruction to test a bit in memory, but for ports we must read the data into a register, then test the bit.



# Addressing: Memory Mapped I/O

- Memory Addressing Mechanism
  - Data lines: To READ /WRITE data
  - Address lines: To uniquely to identify which internal memory register
  - Control lines: To control access operations
    - Rd: To perform READ
    - To WR: To perform WRITE
    - Chip Select: To select / Deselect the specific memory chip amongst several
    - And ofcourse Vcc and Gnd pins
  - All of the above form internal bus system given that this memory is internal (eg. registers within the EU)
- For interfacing peripherals
  - Can the above method be used



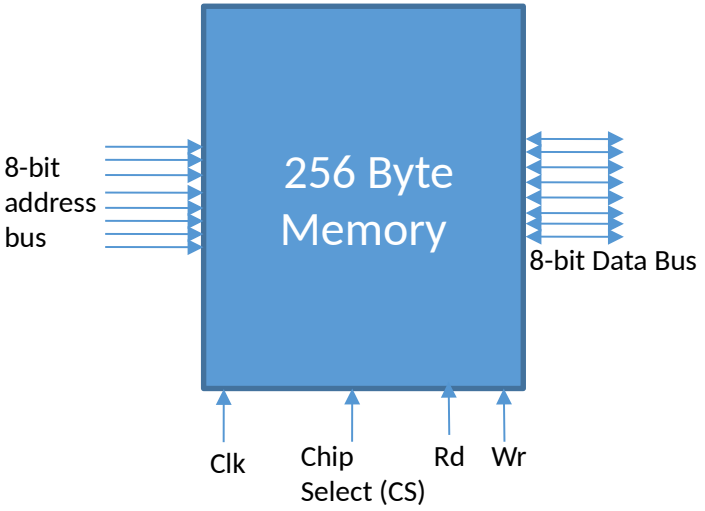



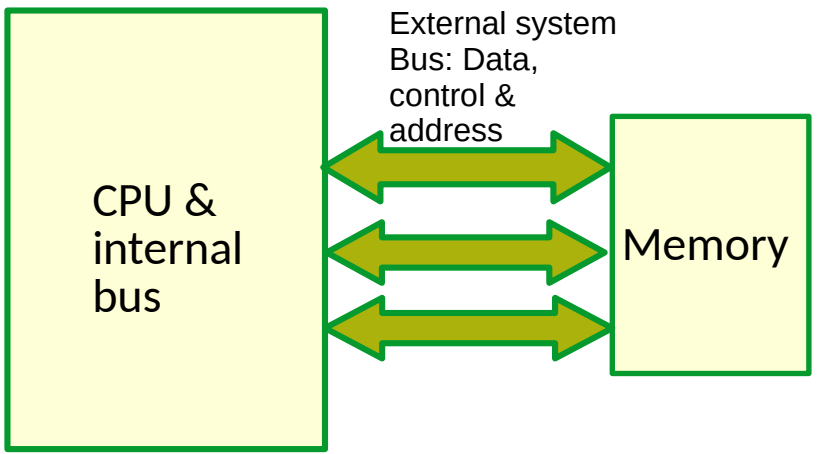
# Addressing: Motivation for memory mapped I/O

- From Processor to a Processor Board (computer Main board)
  - By extending the address bus, data bus and control bus outside the processor, we can add External memory, making them readable / writable similar to internal memory (registers within EU/CU)
  - Given that a piece of memory (hypothetically, say) corresponding to a peripheral, the above method could be used to interface a peripheral. It is very practical and this piece along with a firmware is called Peripheral Interface Controllers (PICs).
  - Peripherals Connect to a Processor through Peripheral Interface Controllers (PIC)
    - PICs are a bunch of registers, readable / writable by a processor similar to registers in internal memory
- Connecting peripherals are always through a Peripheral Interface Controller (PIC)
  - An intelligent peripheral (with built-in processor) can instead communicate with Computer board through a serial / parallel / network port
    - USB based peripherals are quite common now-a-days
  - But then the built-in processor will have a PIC and the peripheral
- From the processor point of view
  - PIC are a bunch of registers which can be read or written to
  - When?
    - Polling or Interrupt

# Addressing: Interfacing External Memory

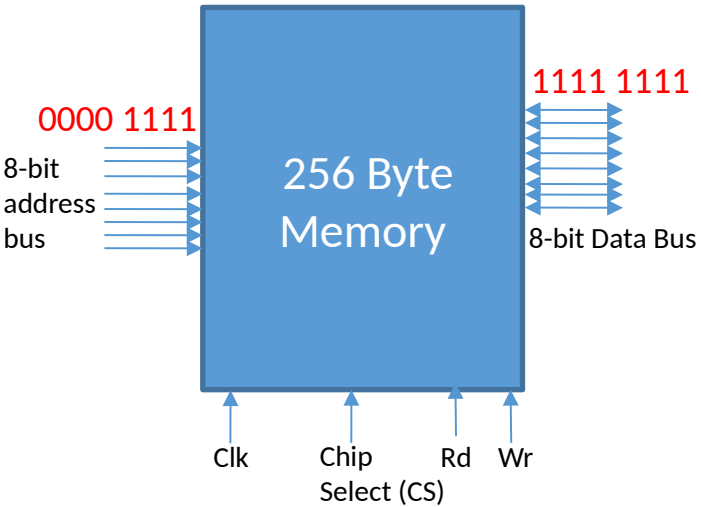
- Like internal memory, external memory also is a bunch of memory locations which are addressable
- I/O pins (connected through external system bus):
  - Data Lines: To read/write data
  - Address lines: To uniquely identify which register in memory to use
  - Control Lines: To control the access operation
    - Rd: To perform Read
    - Wr: To perform Write
    - Chip select: To select/deselect the specific memory chip from amongst several
    - And of-course Vcc and Gnd pins



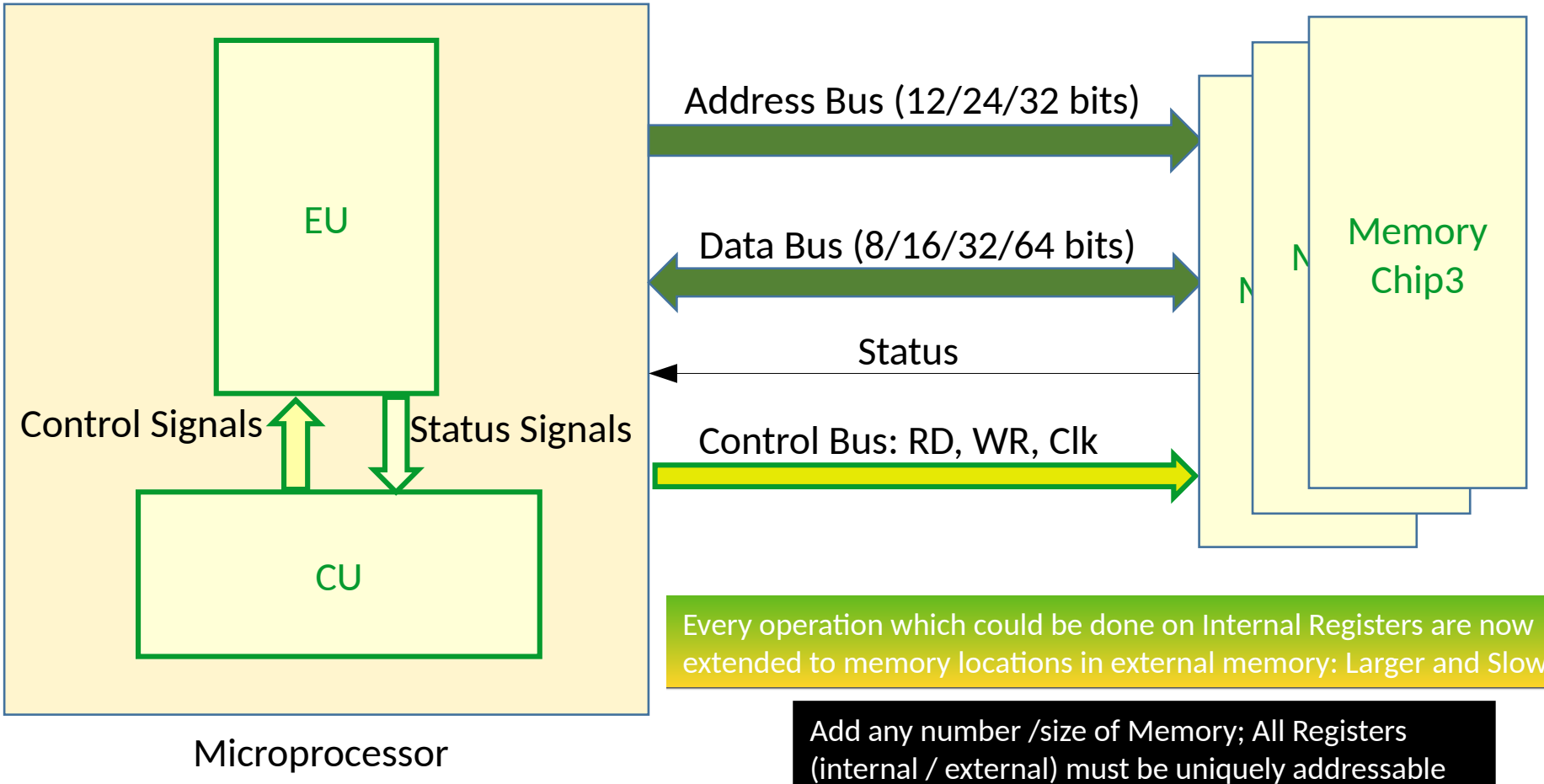



# Signals Controlling External Memory

- To write data to register $15_{10}$ (Hex: 0x0F)
  - Chip select (to be discussed later)
  - Address of Register: 0x0F = $(0000\ 1111)_2$
  - 8 bit data on the data bus to write (Eg. 0xFF)
  - Wr signal
- Similar signal assertion for Rd
- To read or write to this memory, the processor needs to control all the above pins using its address / data and control



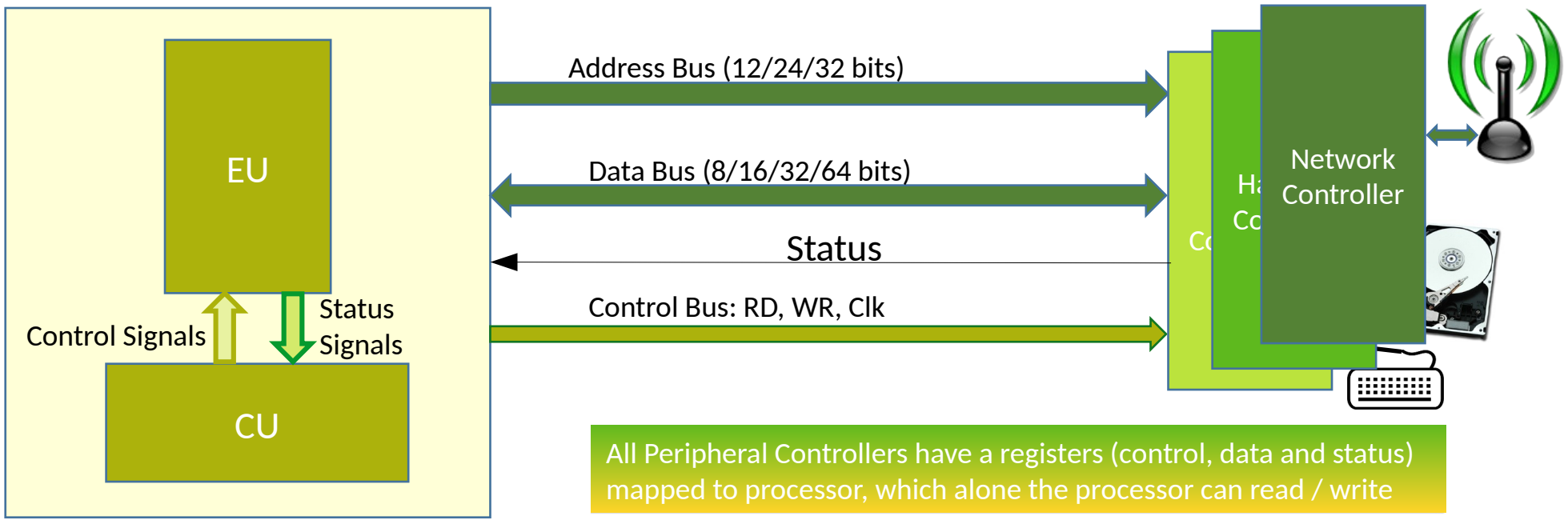
# External Memory is a set of registers (just like internal registers)




Every operation which could be done on Internal Registers are now extended to memory locations in external memory: Larger and Slower

Add any number /size of Memory; All Registers (internal / external) must be uniquely addressable

# Peripherals as a piece of memory

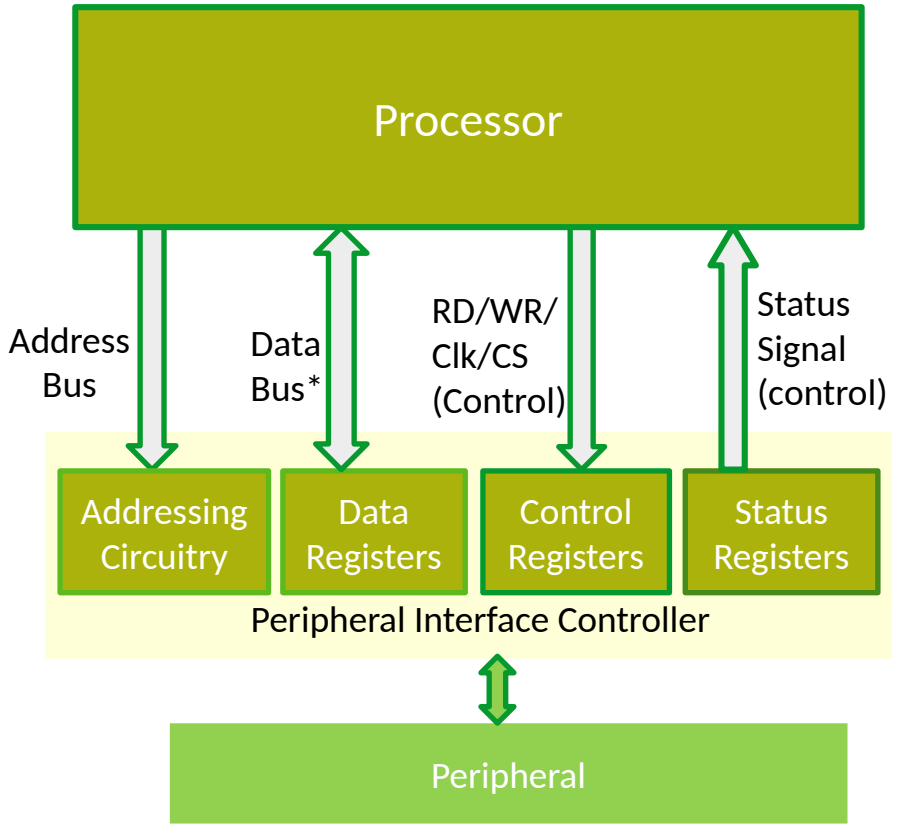



All Peripheral Controllers have a registers (control, data and status) mapped to processor, which alone the processor can read / write

Peripheral Control Registers are therefore no different from memory: to be uniquely addressable

# Peripherals & Peripheral Controllers

- Peripherals always connect to a processor through Peripheral Interface Controllers (PIC)
- PICs
  - have a bunch of registers which interface with processor
    - Data registers
    - Control registers
    - Status registers
  - Have an algorithm implemented which controls the peripheral
- Processor can merely read / write to the above registers using its address, data and control bus




* Direction of data bus depends on the peripheral. If it is keyboard, then the data bus is towards CPU. For monitor it is towards monitor.

# PIC Registers

## Control Register

- **Purpose**: To Control Peripheral
- Processor always writes to this register
- Each bit in the register can indicate one control
- Processor sets/resets each such control bit
  - Eg. For disc, start or stop moving head
  - Eg. Send packet for network controller

## Status Register

- **Purpose**: To Read status of Peripheral
- Processor always reads this register
- Each bit may indicate different status
  - Packet received in network controller
  - Hard disk sector Read Complete
  - Keyboard has valid key

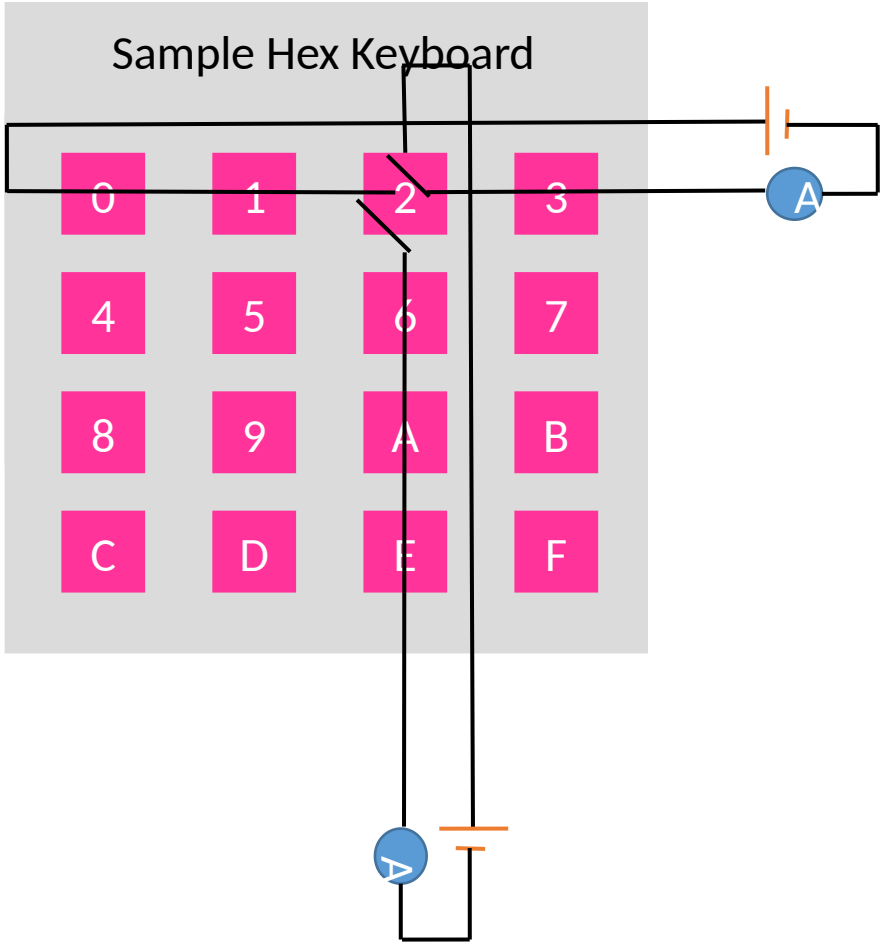
## Data Registers

- **Purpose**: To send/recieve data to/from peripheral
- Processor can read or write to these registers
- Eg.
  - Data received in the incoming packet from network controller
  - Pixel Data to Display in display controller
  - Data to print for printer controller

# Example: A Popular Peripheral, Keyboard

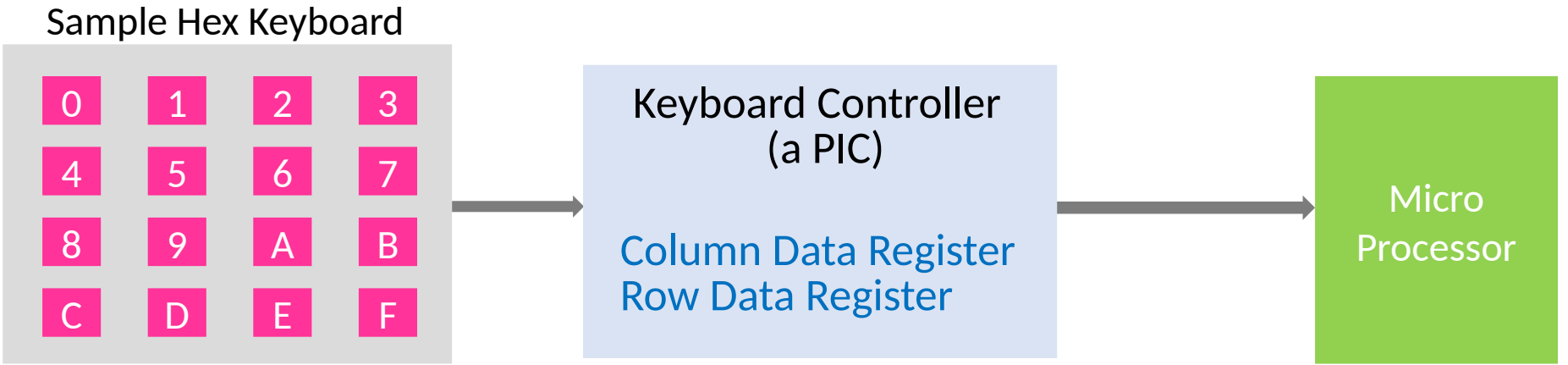
- Keyboard is a peripheral that has been a very integral part of every computer
  - Keyboard connects through a keyboard controller – a PIC
    - The memories (registers) in PIC solves the problem of 'addressing'. The algorithm (protocol) solves the problem of 'handshaking'
    - Intelligent keyboards (with a built-in processor) connects and communicates to a microprocessor / PC using Universal Serial Bus (USB) Interface
- Keyboard has a set of keys which can be set (pressed) only externally
  - Keyboard controller's job is to read these keyboard presses accurately and communicate it to microprocessor
  - No key press is to be missed; each key is to be read once only for each press

# Reading Keys of a keyboard




- 16 keys organised in four rows and four columns
- Each Row and column wired with a Supply Voltage and a Amp-meter
- Each key closes or opens horizontal and vertical circuit
  - Amp-meter detects which row and column selected by key
- Example: when key 2 is pressed
  - Column 2 and row 0 is selected

# Keyboard Controller




- Keyboard has a set of keys (example 16 keys)
- Keyboard controller generates a unique 4 bit - stream for each of these keys
  - Two bits for column (column data register) and two bits for row (row data register)
- Processor reads the two registers and concatenates them together to indicate key-pressed

# Keys to register values

- Similarly 64 keys could be organized in a 8 x 8 matrix with a circuit for each row and column
  - Column and Row Registers are now 4 bits each
- If key 12 is pressed, then in Row Register is 0001, the Column Register is 0011
  - Processor will read two registers and translates it to binary value 0000 1100 by software (Device Driver)

Register Values: 0001 0011

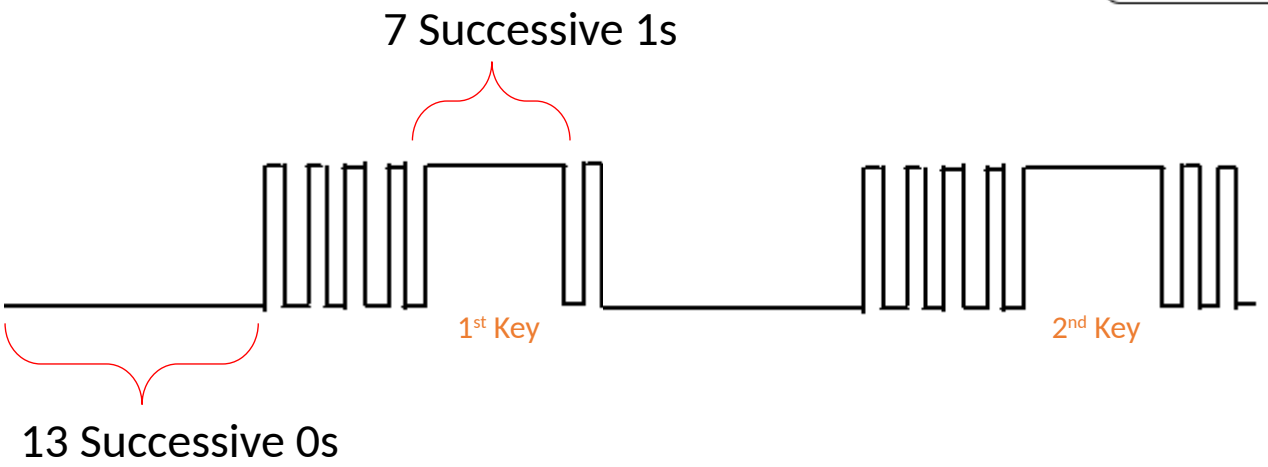
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Row 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Row 1 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Row 2 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| Row 3 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| Row 4 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| Row 5 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| Row 6 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| Row 7 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |

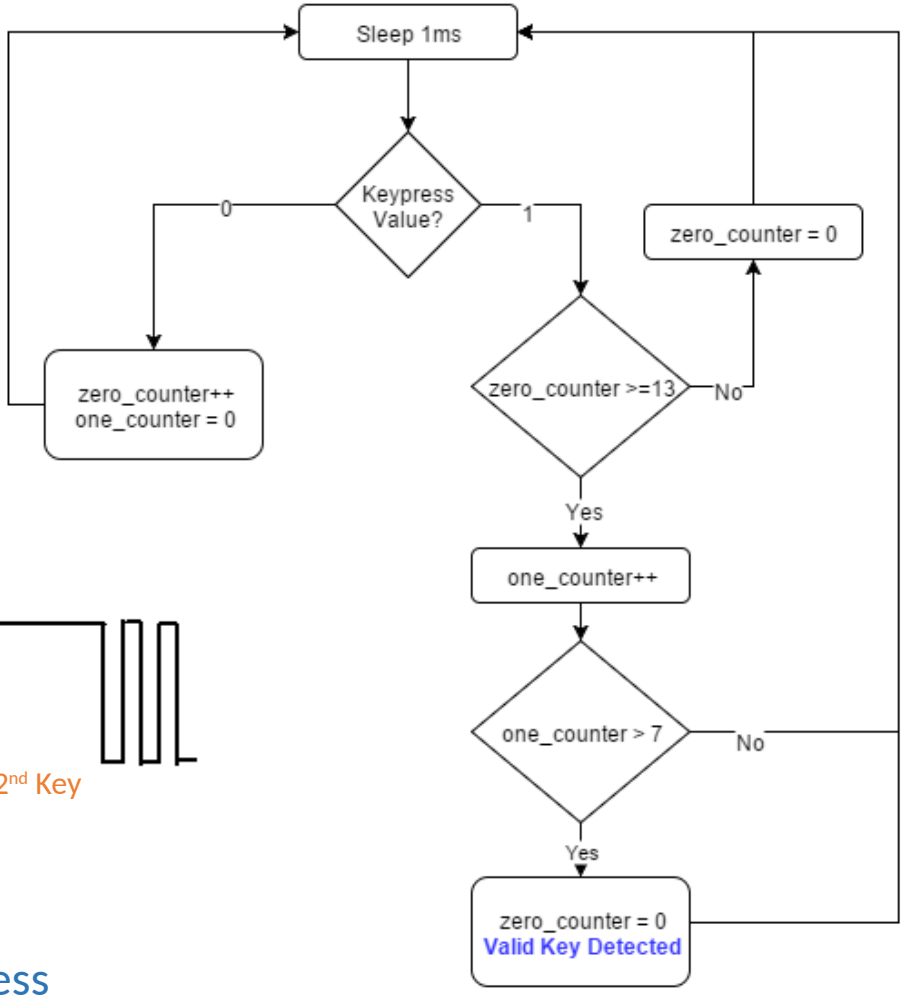
8 x 8 keyboard matrix

# Polling of Key press

- How does a Processor detect each key in Sequence?
  - Average typing speed is around 12 stroke/key press per second
    - i.e a split-time of 83 msec between keypresses
  - Even assuming very high speed of 50 strokes /sec implies a key press every 20 msec
    - Assume a minimum of 7 msec press time and 13 msec before another key is pressed
- Assume a processor reads (POLLS) keyboard registers every 1 msec
  - At least 7 successive ONES for key presses and at least 13 successive ZEROES before next key is pressed
  - Keyboard de-bounce: oscillations when contact is made and broken
    - Events of, less than 7 successive ONES and 13 successive ZEROs are ignored as key-bouncing and not detected as another key-stroke
- To detect an intentional keypress, there must be some minimum time 13 successive ZEROs and at least 7 successive ONES

# Key Board De-bounce & Key press Detection: Polling Method



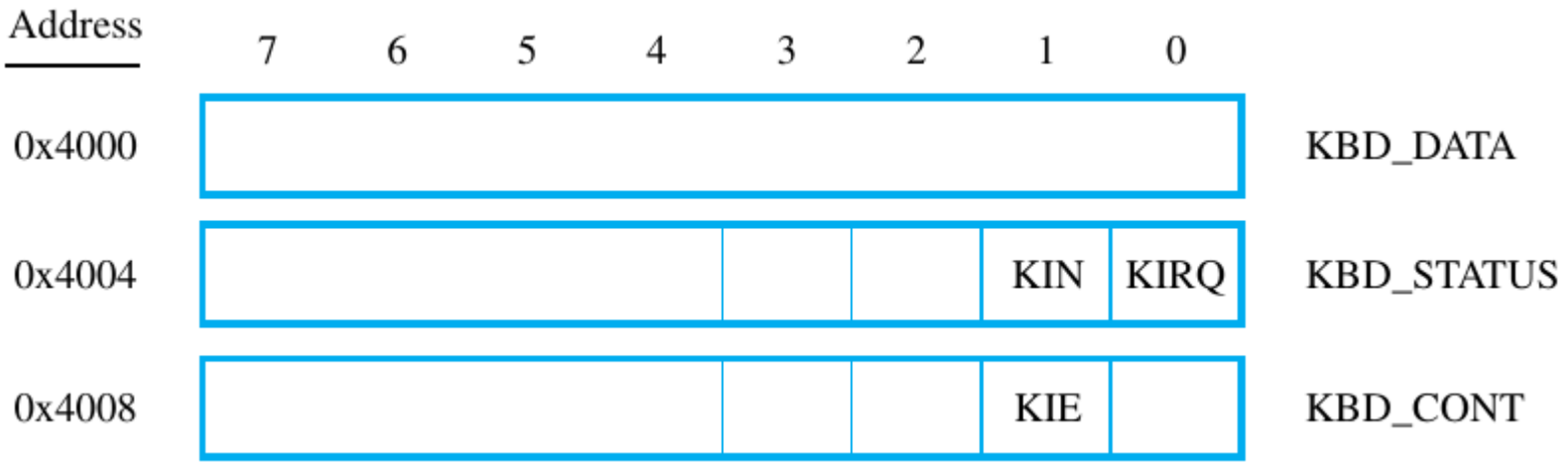

Required Behavior On Consecutive Key Press

# Keyboard PIC Registers

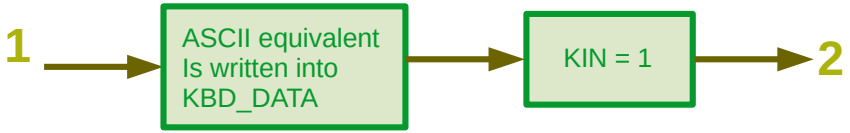
Hamachar, P99

- 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

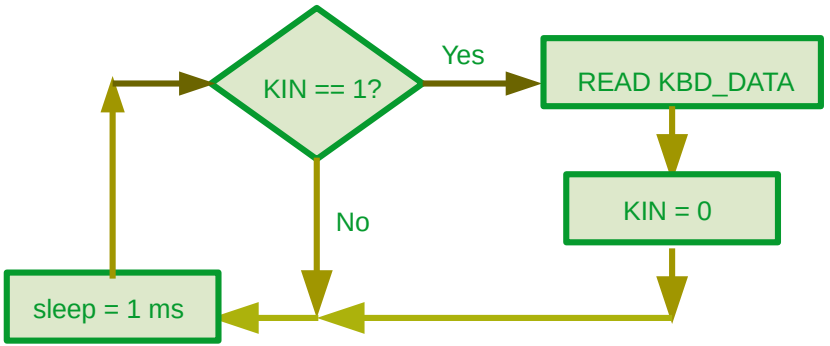
- KBD_CONT
  - Keyboard control register – controls or configures the keyboard function
  - KIE – bit number 1 in KBD_CONT
    - Keyboard interrupt enable (KIE)
    - Interface the keyboard to the CPU through interrupt
- Keyboard Interfacing through MMIO



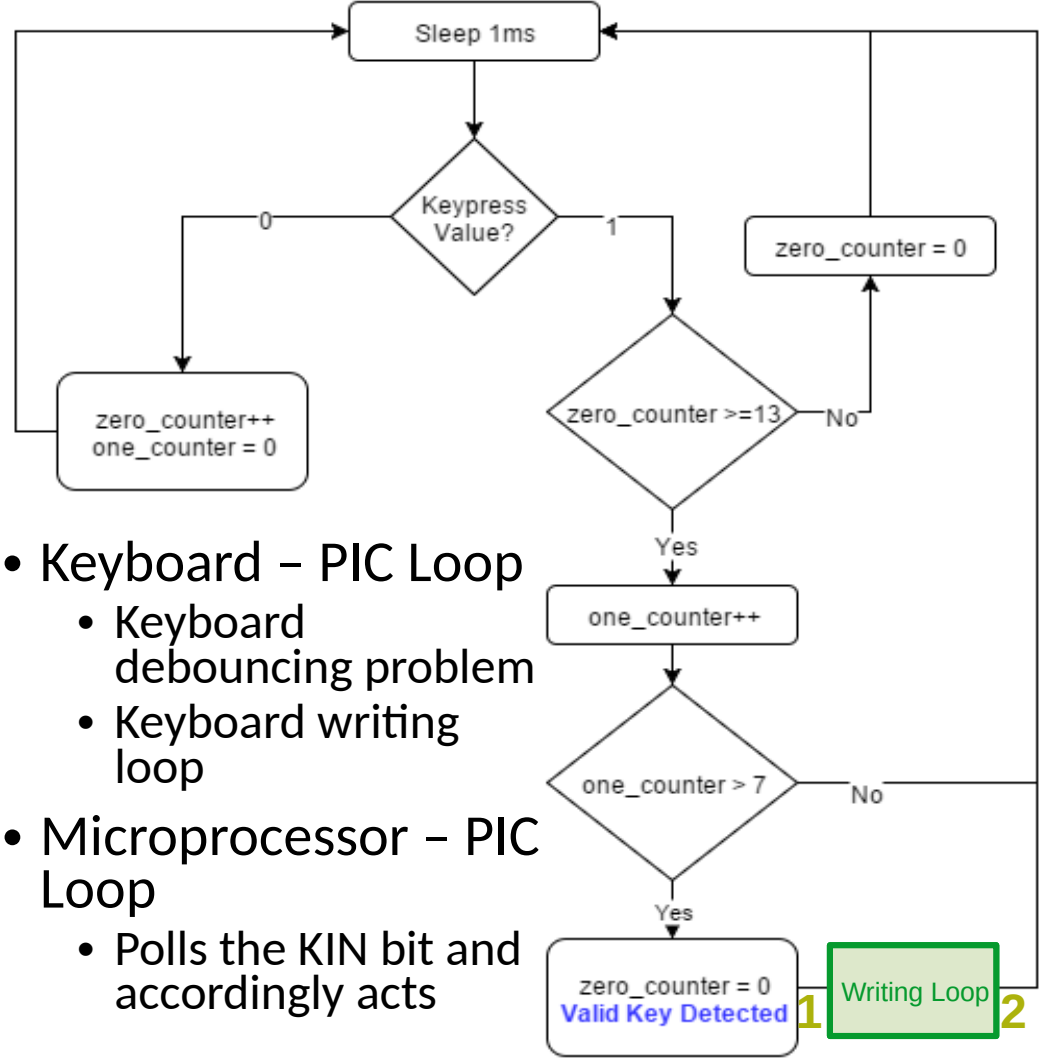
# Key Board De-bounce & Key press Detection: Polling Method




Keyboard's Writing Loop

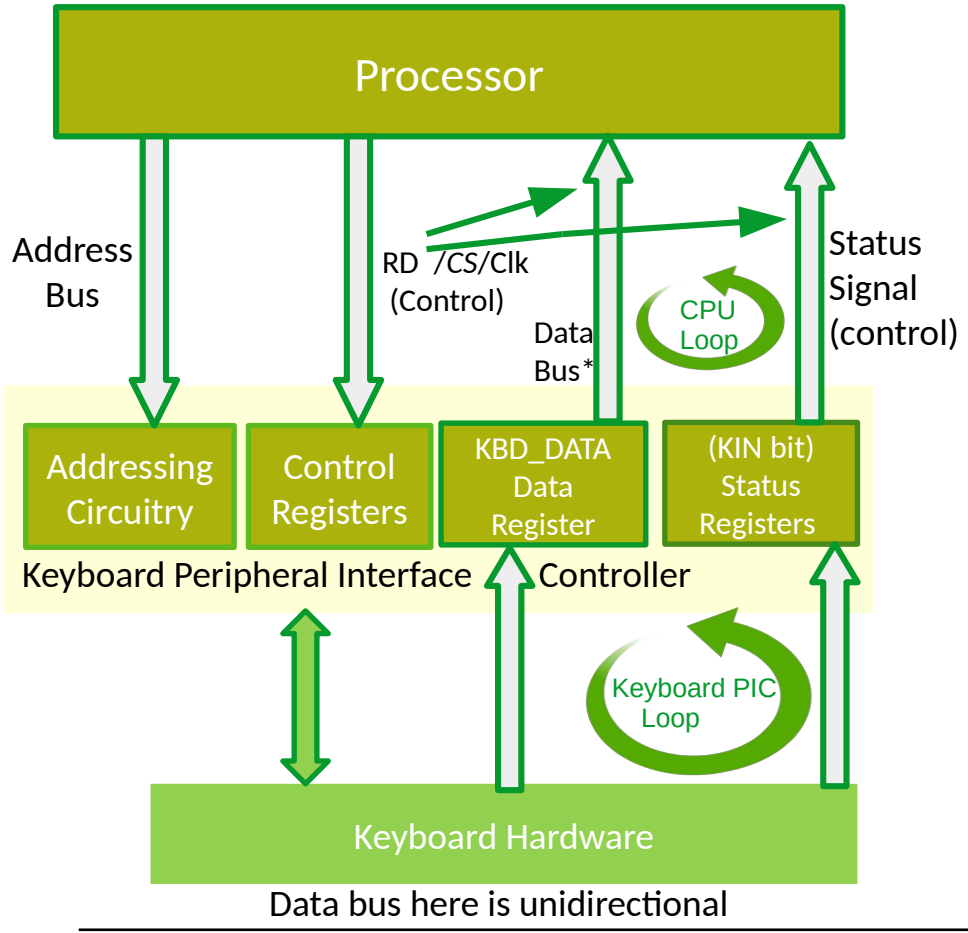


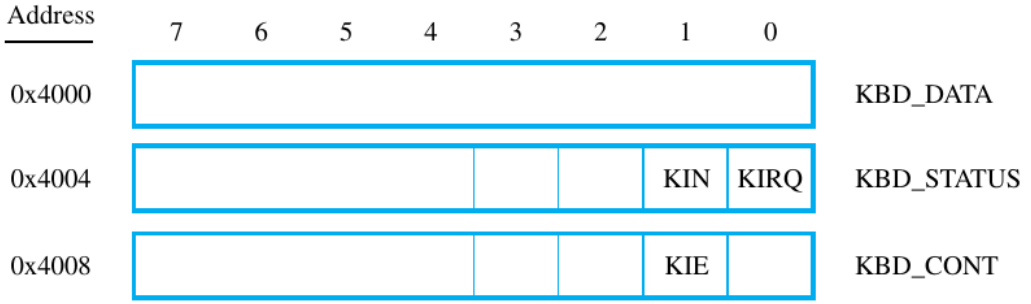

Microprocessor - PIC Loop:

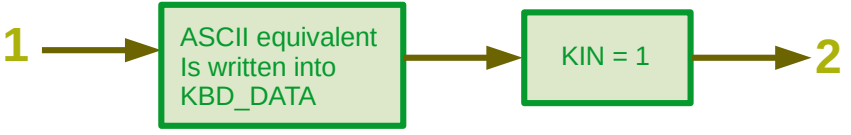



- Keyboard – PIC Loop
  - Keyboard debouncing problem
  - Keyboard writing loop
- Microprocessor – PIC Loop
  - Polls the KIN bit and accordingly acts

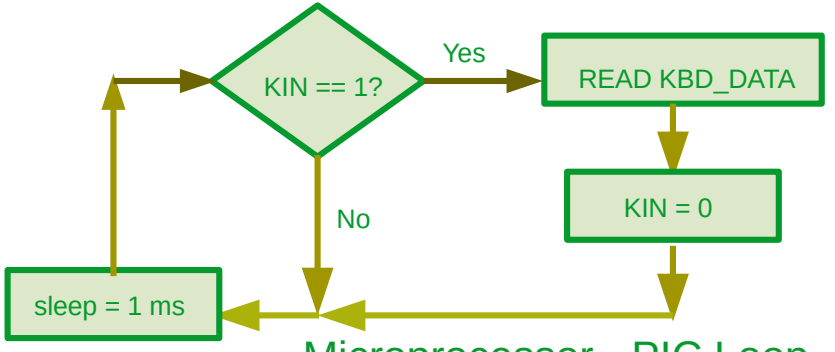
# Keyboard PIC Controller: Polling Control Loops




Keyboard's Writing Loop executed in PIC




Microprocessor - PIC Loop

# To Conclude: Solutions for Handshaking,

- In Polling, control registers in Peripheral Controller are used to detect state of Peripherals, regularly
  - Would require frequent polling and waste of CPU cycles
  -
- In future, we would see, interrupt and DMA are alternative ways of handshaking.
- 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