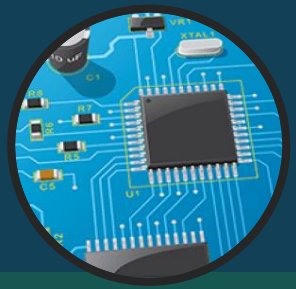
I/O Module Interrupts & DMA



Topic # 10

Spring 2020

Muhammad Imran Abeel imran.abeel@seecs.edu.pk

I/O Module

The computer system's Input/Output (I/O) Architecture is its interface to the outside world

- I/O Module has two major Functions

Interface to Processor and Memory via System Bus

Interface to one or more Peripherals

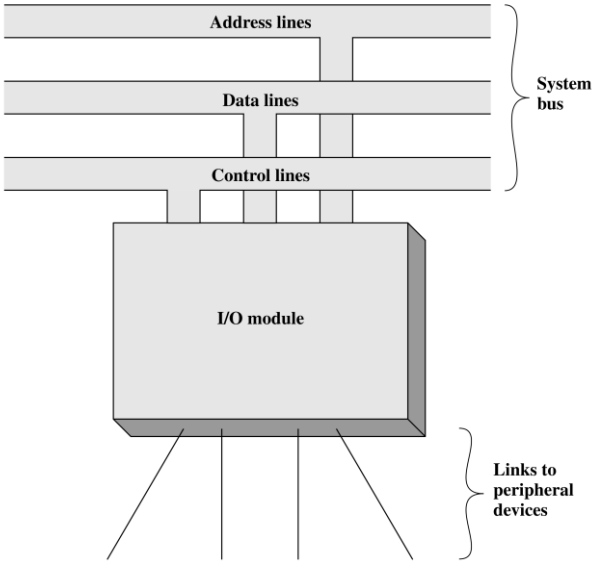
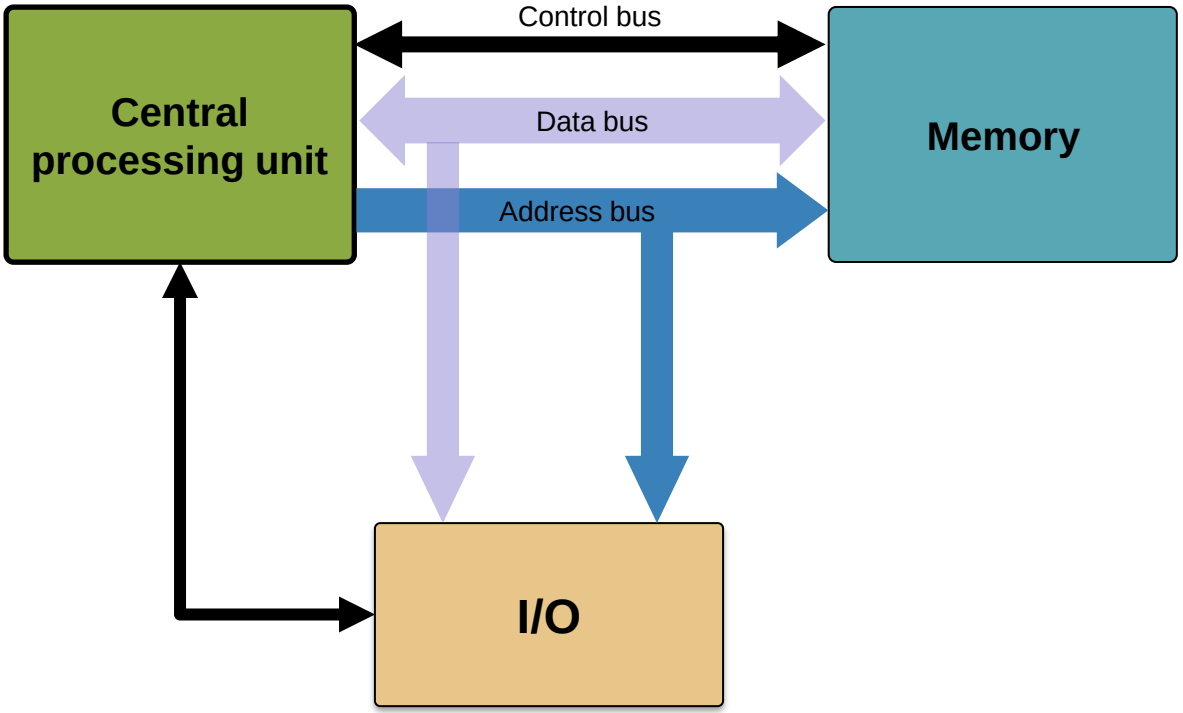


Figure 7.1 Generic Model of an I/O Module

Revisiting System Diagram



I/O Module

- Why we need an I/O module?

Wide variety of Peripheral Devices

Different amount of Data at different Speeds

Different Data Formats and Word Length

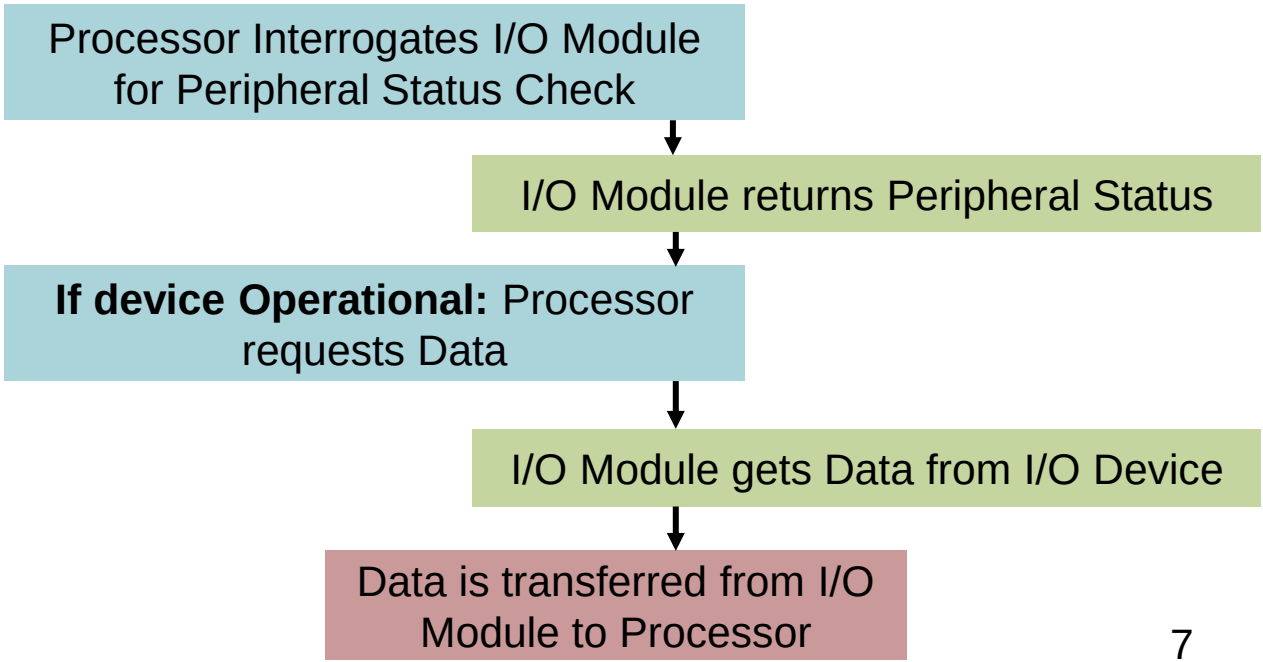
Functions of I/O Module

- **Control and Timing**
  - Co-ordinate flow of data
- **Processor Communication**
  - Command decoding
  - Data transfer (proc-I/O module)
  - Status reporting
  - Address decoding
- **Device Communication**
  - Commands
  - Status information
  - Data transfer(I/O – I/O module)
- **Data Buffering**
- **Error Detection**
  - Mechanical or electrical malfunction (paper jam, bad disk sector etc.)

I/O Module Functions

- To control the flow of Data between Internal Resources and External Devices

Steps required to transfer Data from I/O to Processor



I/O Module Block Diagram

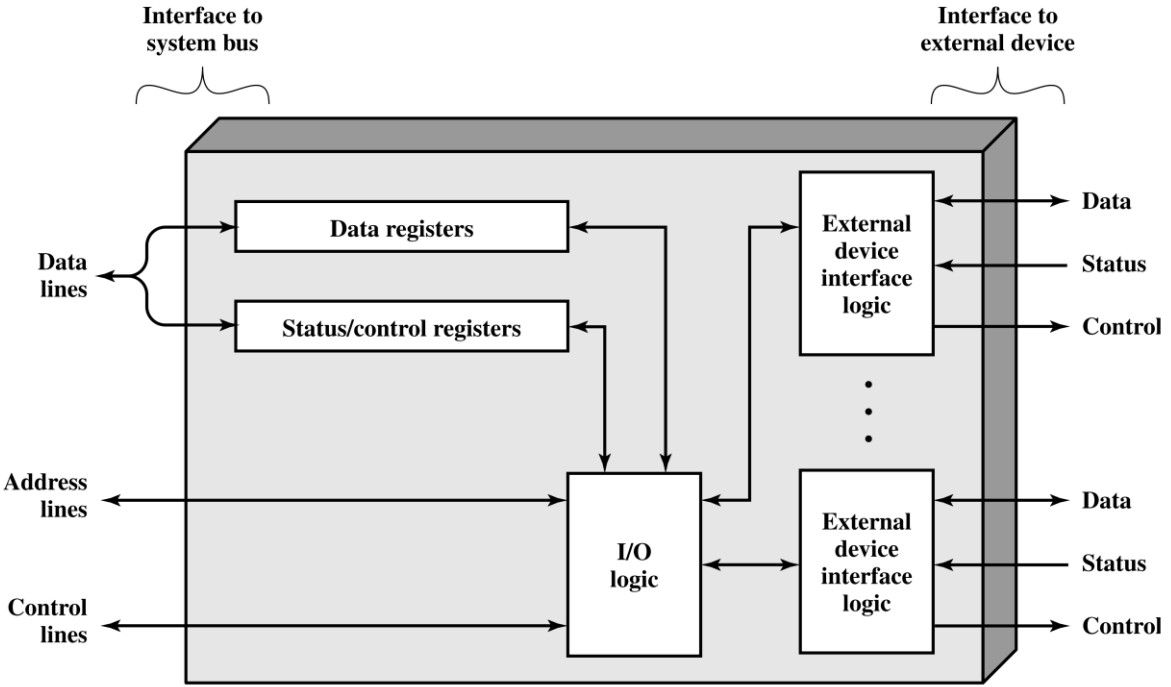
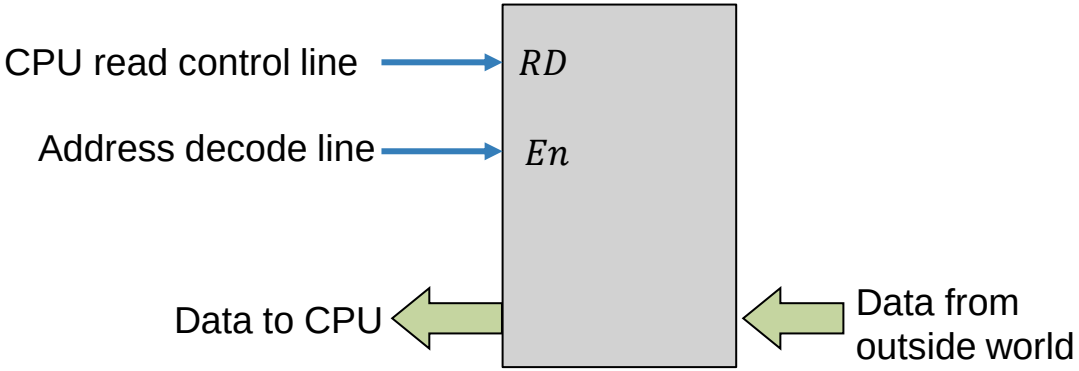


Figure 7.3 Block Diagram of an I/O Module

Input Port

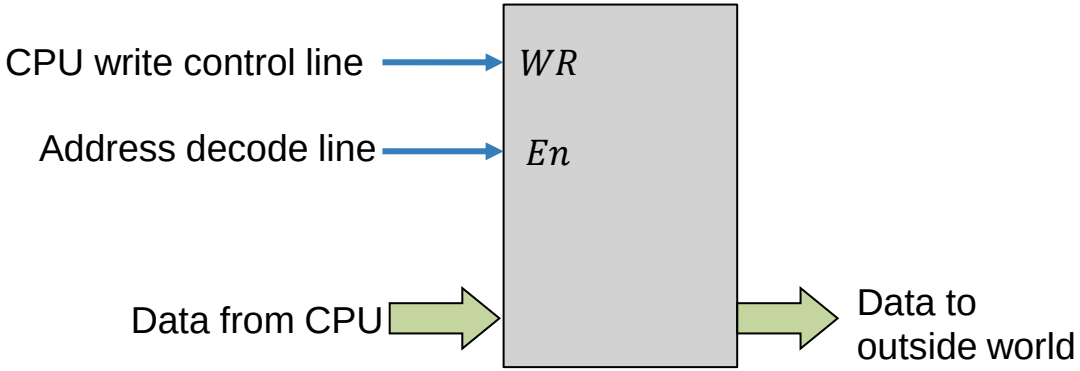
CPU's connection to outside world



- I/O Port uses a Buffer to capture Data
- Buffer makes the data available for CPU

Output Port

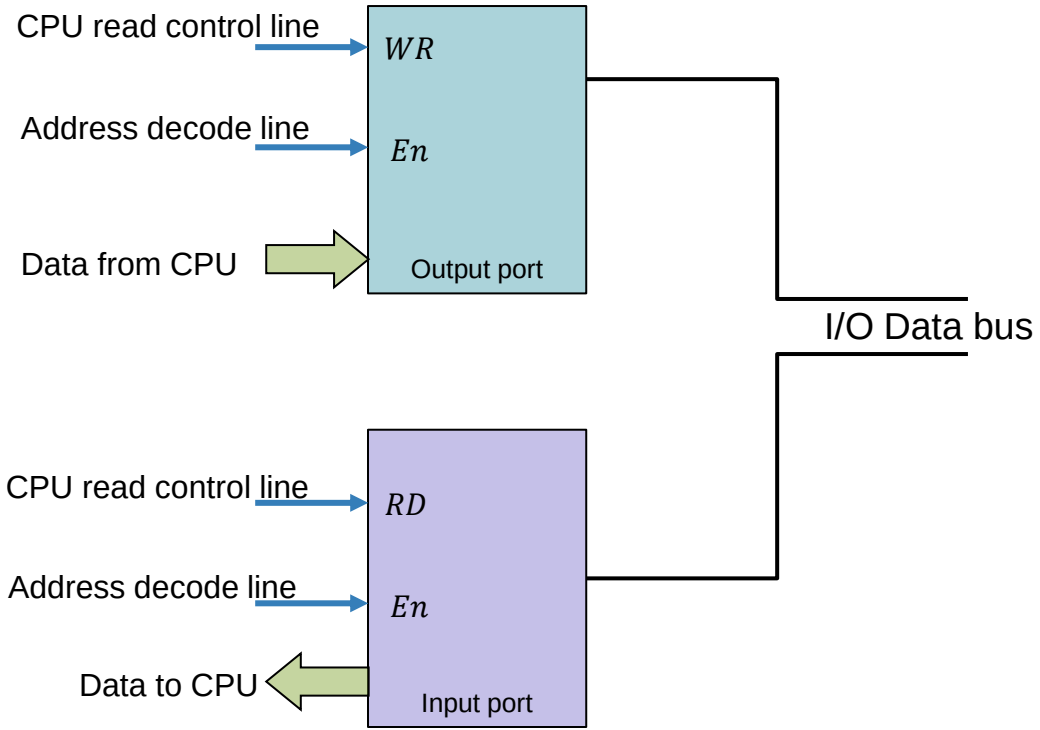
CPU's connection to outside world



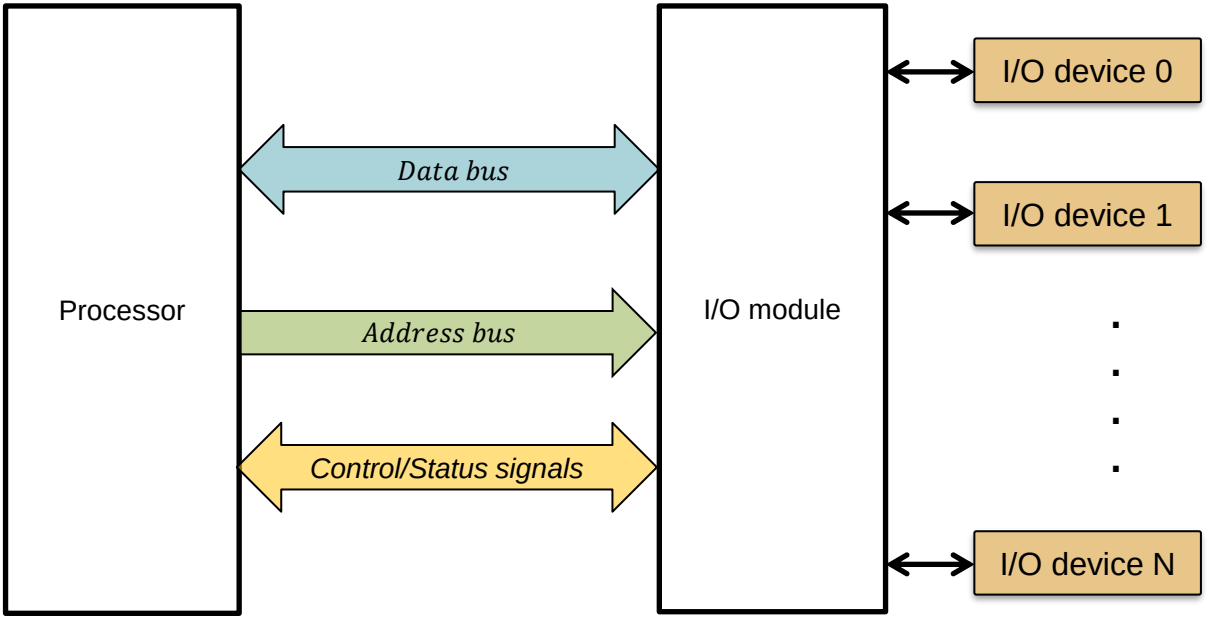
- I/O port uses a latch to capture data
- This makes the data available for the Device

I/O Port

CPU's connection to outside world



I/O Interface



I/O Devices

To the programmer, the difference between I/O-mapped and memory-mapped input/output operation is the instruction to be used

- For memory-mapped I/O, any instruction that accessed memory can access a memory-mapped I/O port
- I/O-mapped input/output uses special instruction to access I/O port

Handshaking

Polling

- Used in case of Programmed IO
- Processor checks peripheral status after regular intervals of time
- Peripherals cannot interrupt PC

Interrupts

- Processor issues a command to I/O and moves to next instruction
- Whenever the data is available, the I/O sends interrupt signal to CPU for exchange of data

Programmed I/O

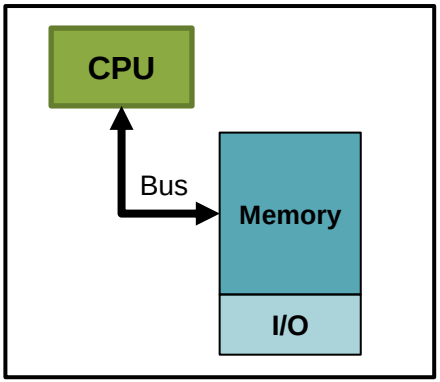
Processor executes a program, that gives it direct control of I/O operation including:

- Sensing Device Status
- Sending Read or Write Command
- Transferring Data

Programmed I/O

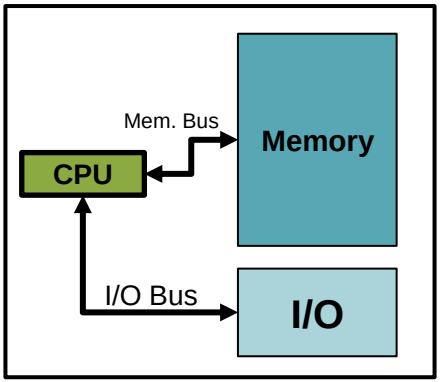
Memory Mapped I/O

- CPU read/writes data into the **shared** I/O address space



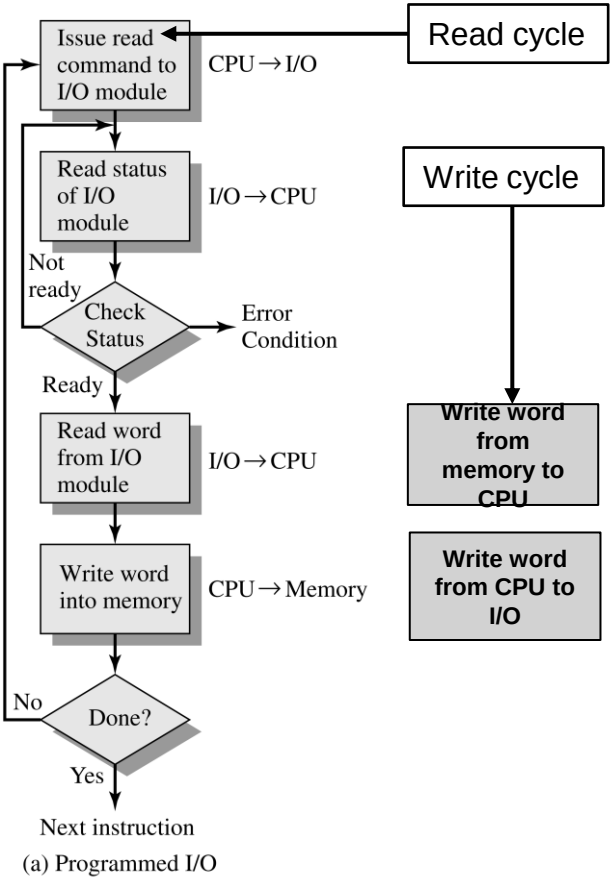
Standard Mapped I/O

- CPU read/writes data into the **dedicated** I/O address space



Note: The input and output operation looks very similar to a memory read or write operation except it usually takes more time since peripheral devices are slow in speed than main memory modules

Programmed I/O (R/W Cycle)



I/O Commands

There are four types of I/O Devices

- **Control:** Activate Peripheral and Instruct
- **Test:** used to Test Status Conditions associated with I/O Module and its Peripherals
- **Read:** Prompt I/O Module to obtain Data from I/O
- **Write:** Cause I/O Module to transmit Data to the Peripherals

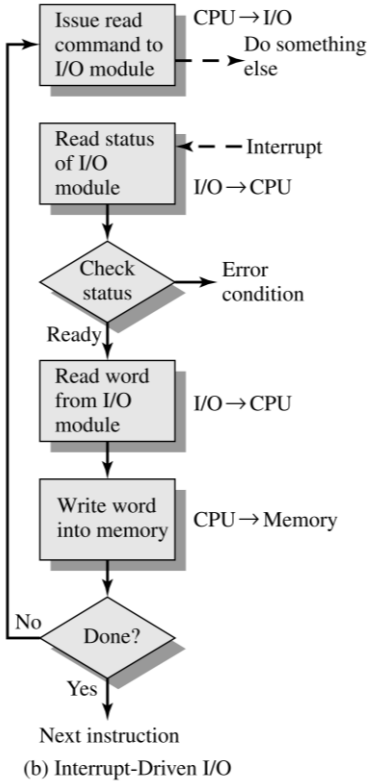
Interrupt Driven I/O

Processor issues R/W Command and moves to other work

When Data is ready, the I/O interrupts the CPU

Interrupt Control I/O

- For processor, the action for an input is as follows:



(b) Interrupt-Driven I/O

Processor issues Read Command

I/O Module returns Peripheral Status

If device operational, Processor requests Data

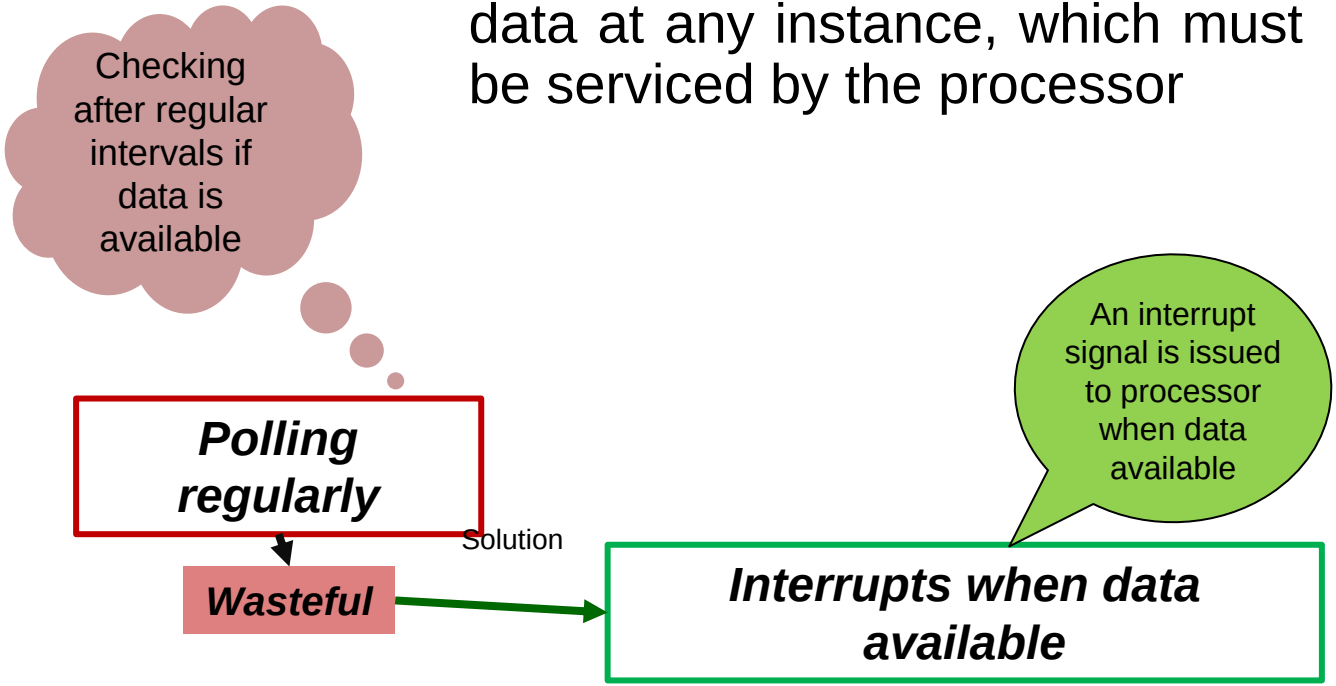
I/O Modules gets data from I/O device and issues interrupt signals

After each instruction Processor checks for Interrupt

Data is transferred from I/O Module to Processor

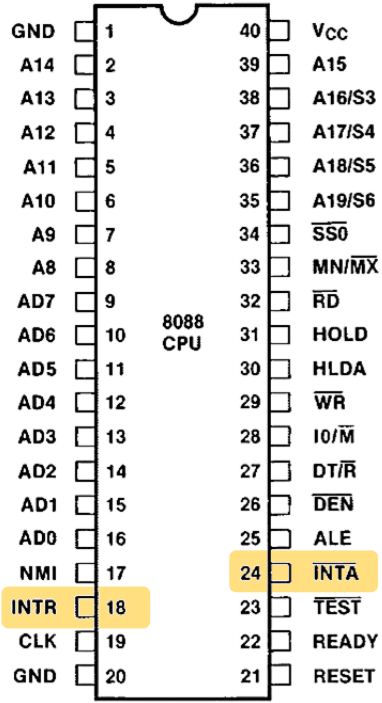
Microprocessor Interfacing: Interrupts

- A peripheral device can receive data at any instance, which must be serviced by the processor



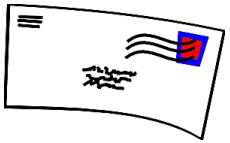
Microprocessor Interfacing: Interrupts

- Requires an extra pin or pins: INTR
  - If INTR is 1:
    - Processor suspends current program
    - jumps to an Interrupt Service Routine or ISR
  - Known as **interrupt-driven I/O**
  - Essentially, "polling" of the interrupt pin is built-into the hardware, so no extra time!



Microprocessor Interfacing: Interrupts

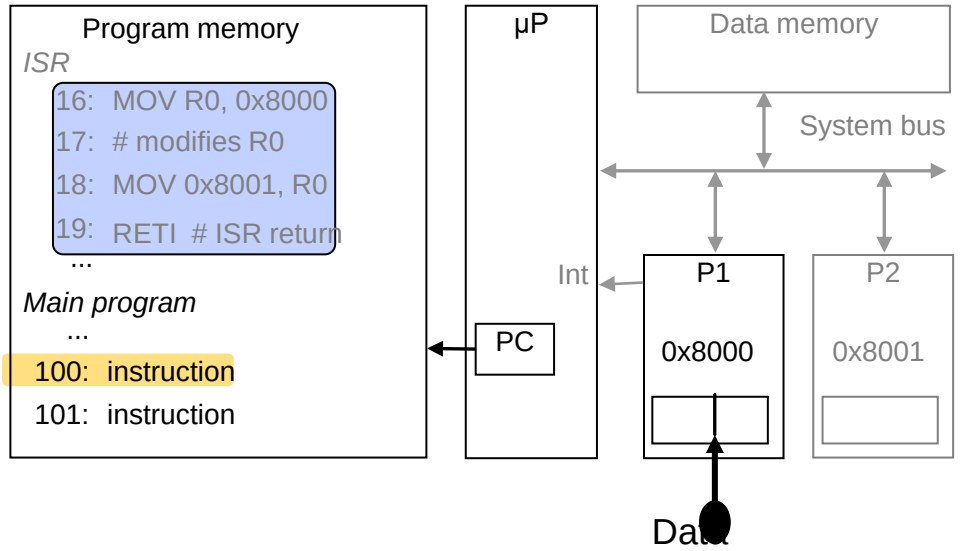
- What is the Address (Interrupt Address Vector) of the ISR?
  - ***Fixed Interrupt***
    - Fixed address built into microprocessor, cannot be changed
    - Either ISR stored at address or a jump to actual ISR stored if not enough bytes available
  - ***Vectored Interrupt***
    - Peripheral must provide the address
    - Common when microprocessor has multiple peripherals connected by a system bus
  - **Compromise: interrupt address table**



Interrupt-driven I/O using Fixed ISR Location

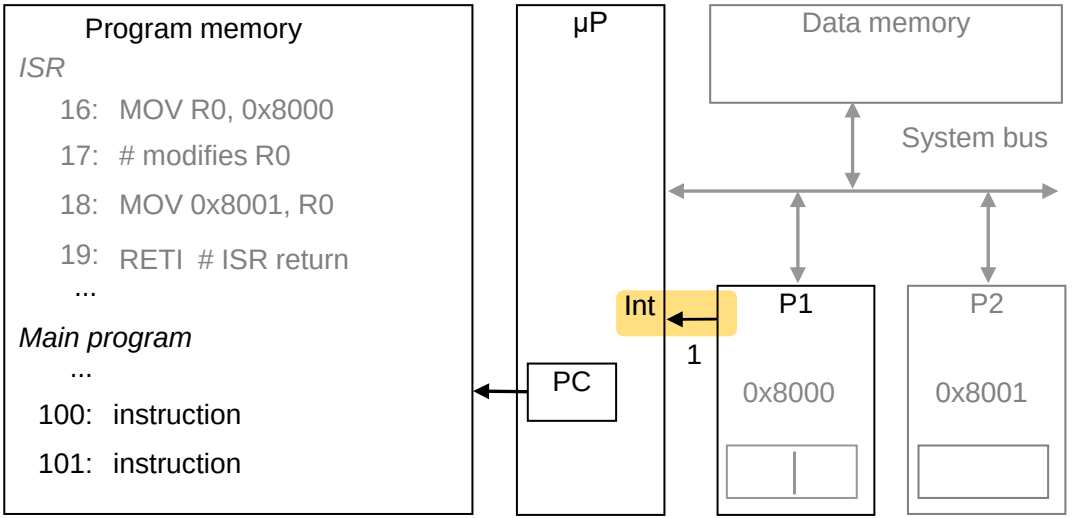
1(a): μP is executing its main program instruction at Address 100

1(b): P1 receives input data in a register with address 0x8000



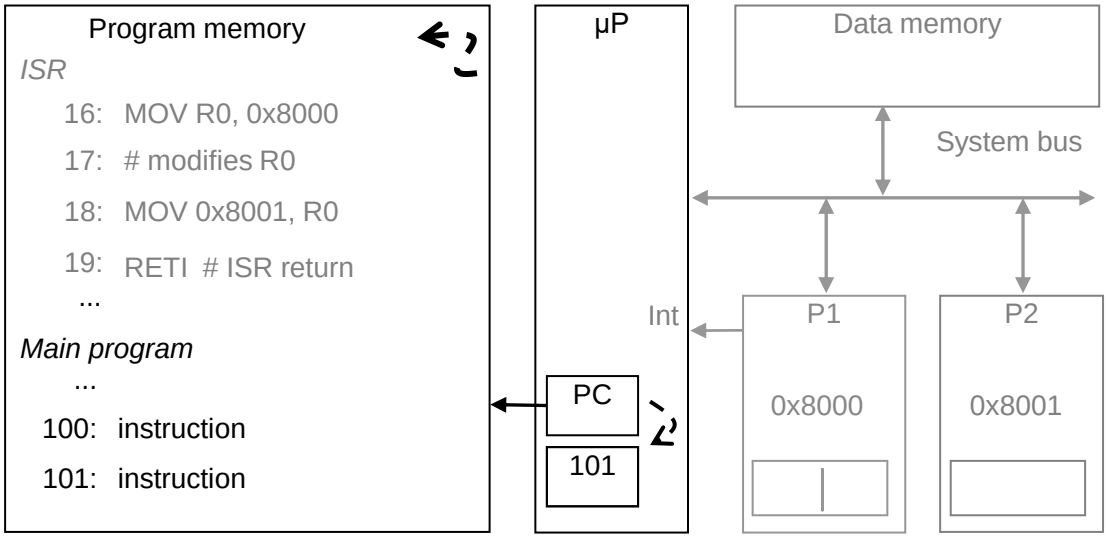
Interrupt-driven I/O using Fixed ISR Location

2: P1 asserts INT to request servicing by the Microprocessor



Interrupt-driven I/O using Fixed ISR Location

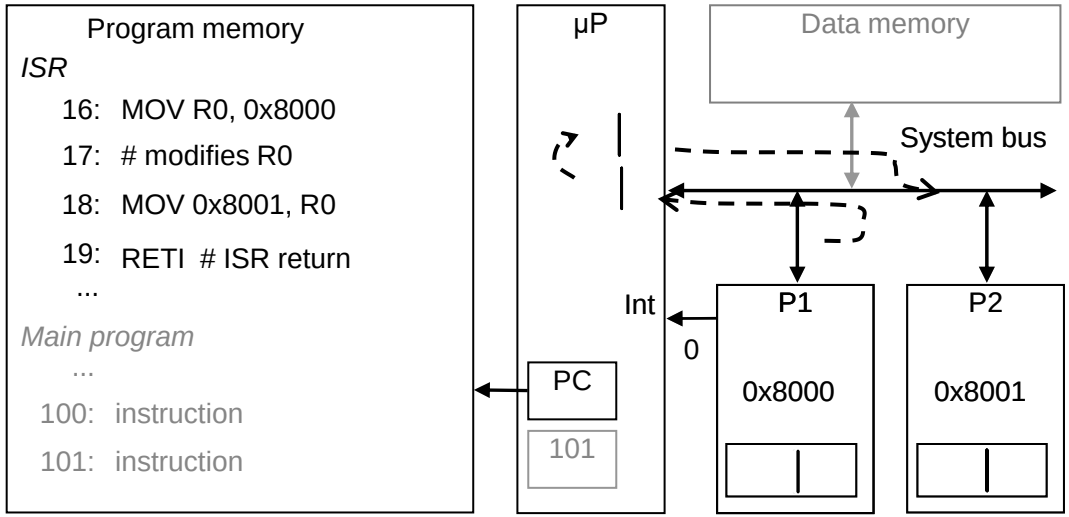
3: After completing instruction at 100, μP finds INT asserted, pushes the PC's current value of 101 to the System Stack, and sets PC to the ISR fixed location address of 16



Interrupt-driven I/O using Fixed ISR Location

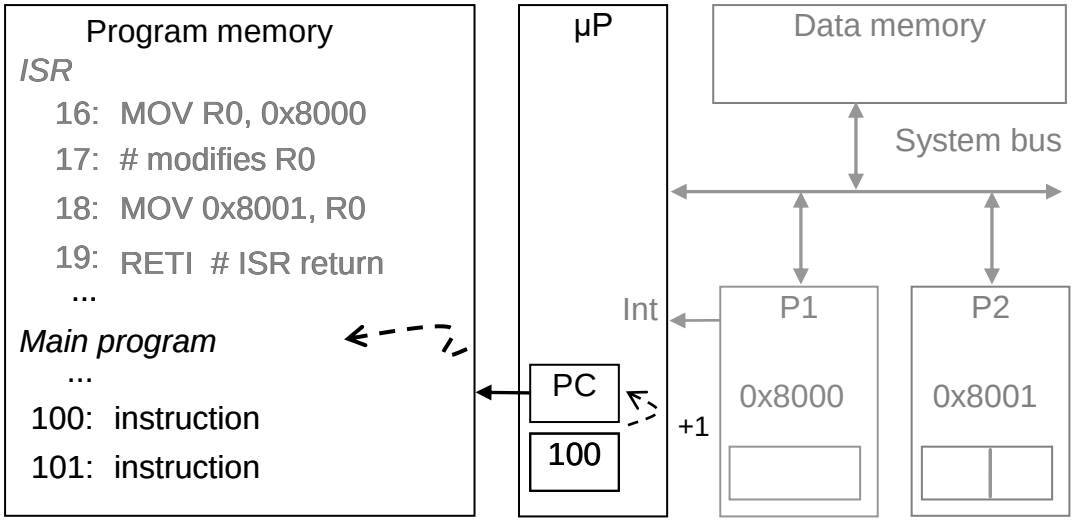
4(a): The ISR reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001.

4(b): After being read, P1 de-asserts INT

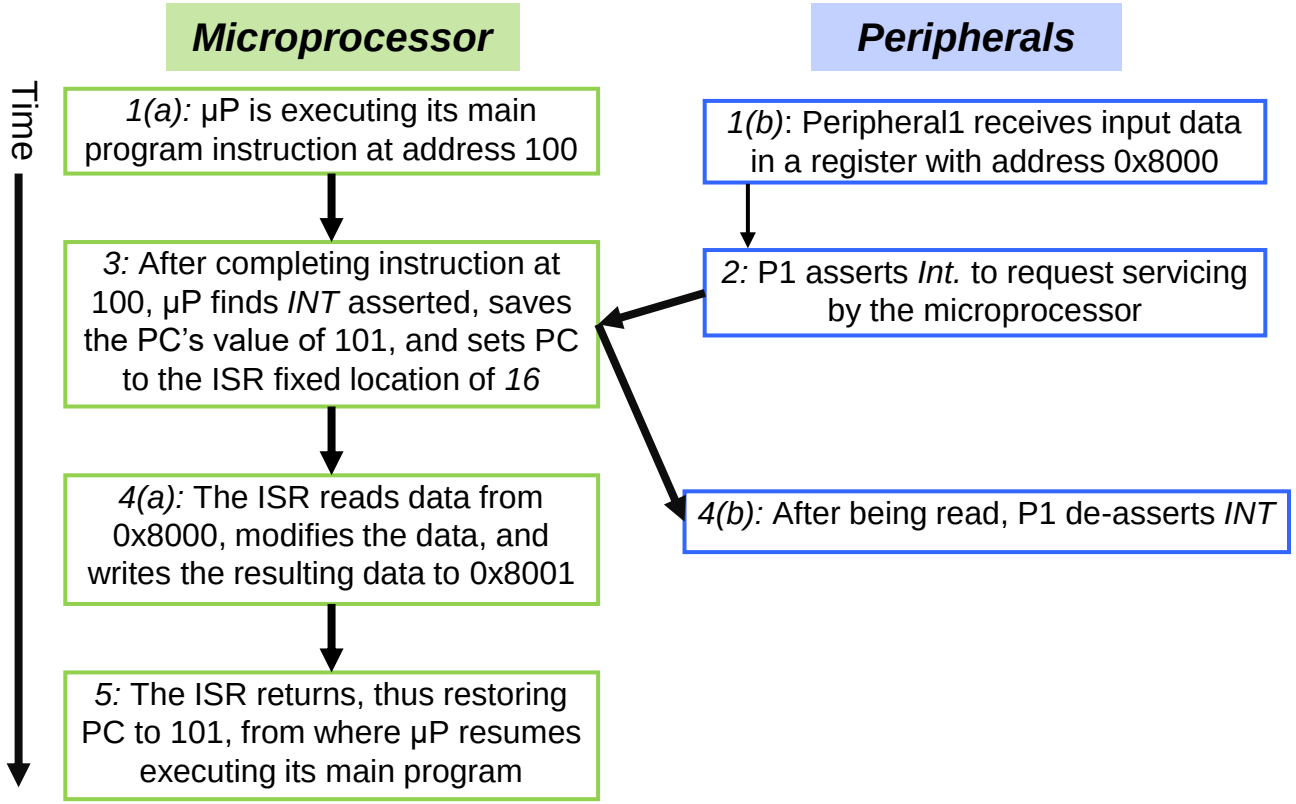


Interrupt-driven I/O using Fixed ISR Location

5: The ISR returns, thus restoring PC to 101, where μP resumes executing the main program



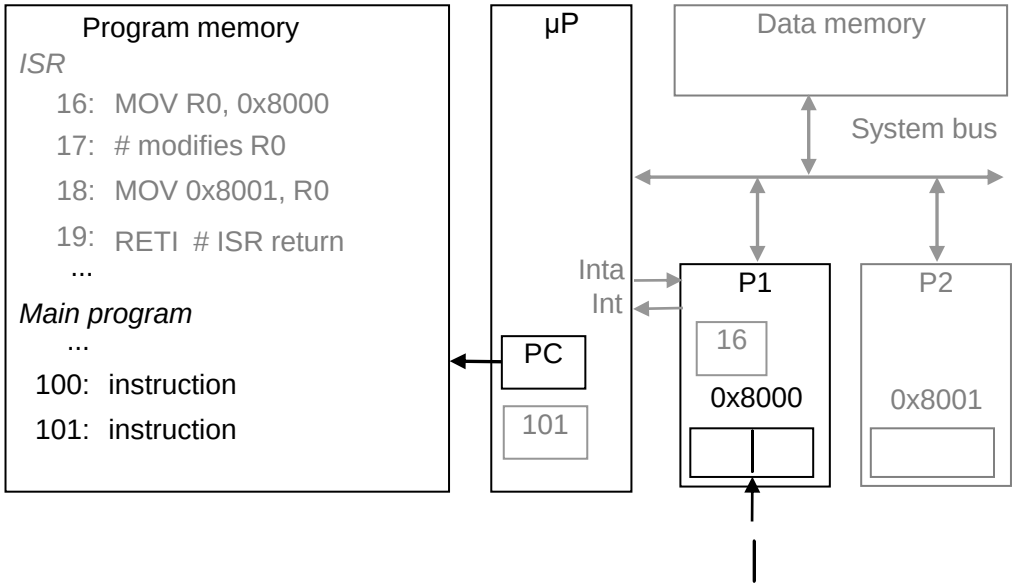
Interrupt-driven I/O using Fixed ISR Location



Interrupt-driven I/O using Vectored Interrupt

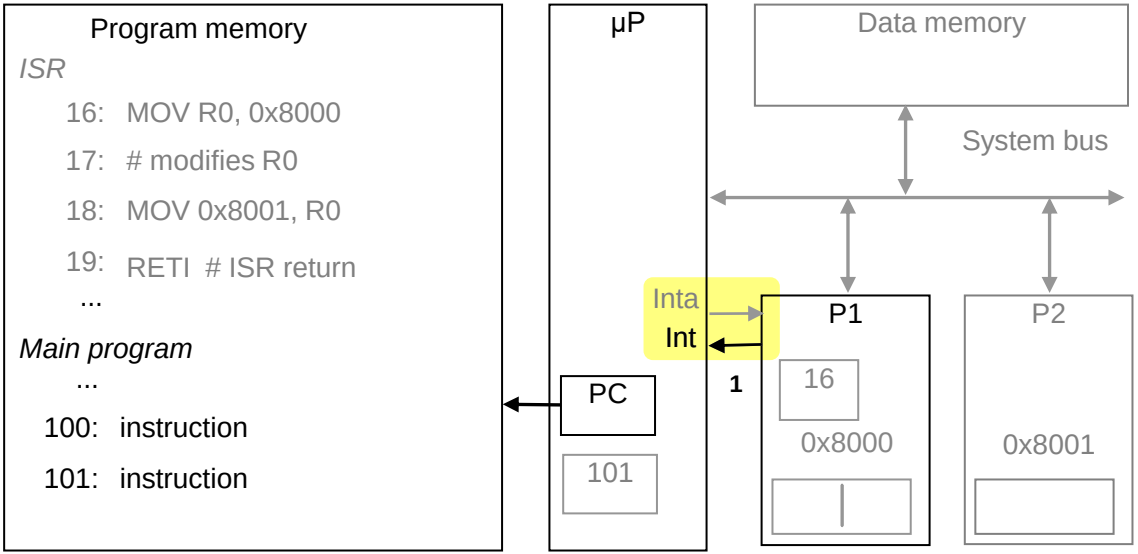
1(a): P is executing its main program Instruction at Address 100

1(b): P1 receives input data in a register with address 0x8000



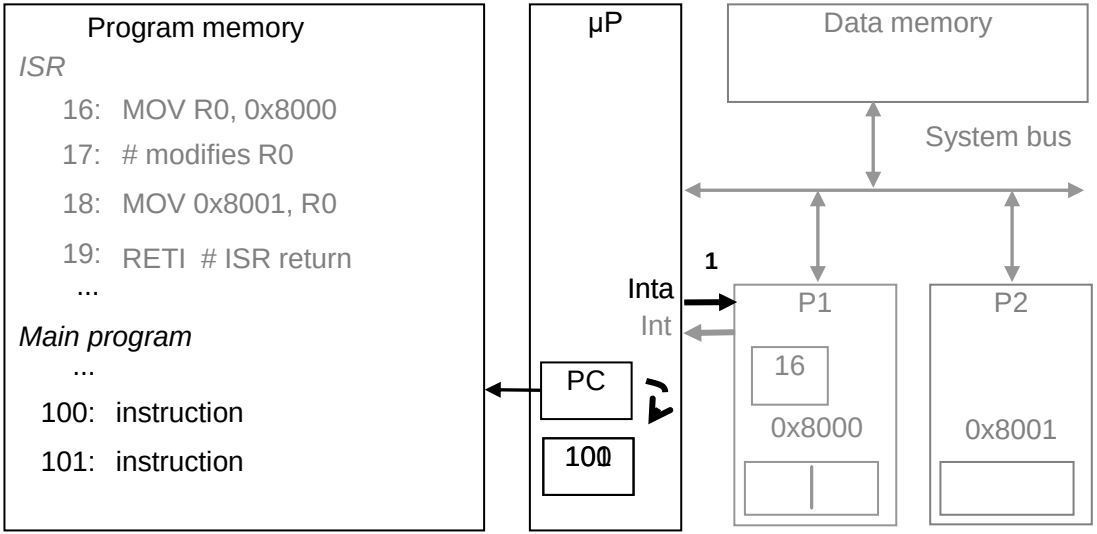
Interrupt-driven I/O using Vectored Interrupt

2: P1 asserts *INT* to request servicing by the Microprocessor



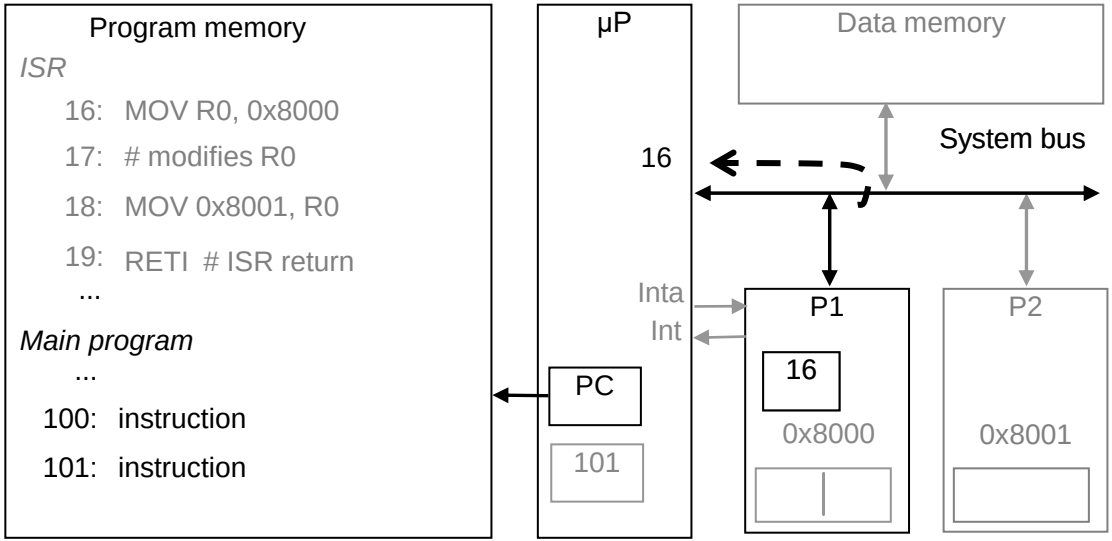
Interrupt-driven I/O using vectored interrupt

3: After completing instruction at 100, µP finds *INT* asserted, saves the PC's value of 101, and **asserts *INTA***



Interrupt-driven I/O using vectored interrupt

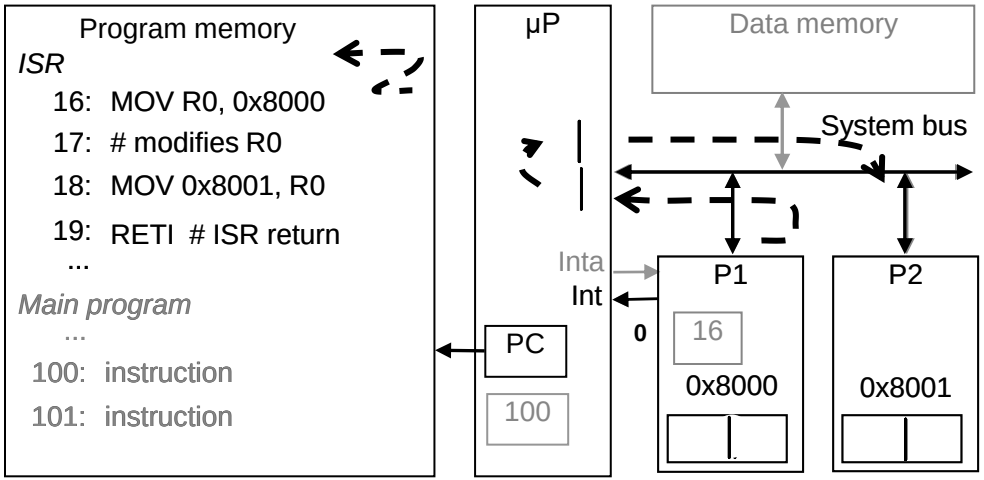
4: P1 detects *INTA* and puts **Interrupt Address Vector 16** on the Data Bus



Interrupt-driven I/O using vectored interrupt

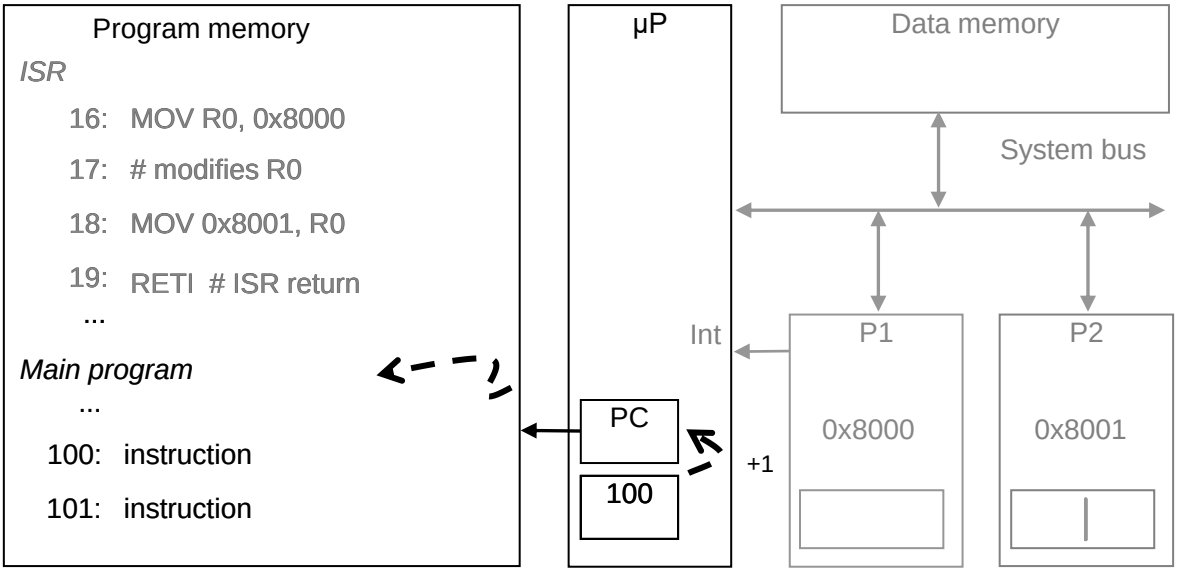
5(a): PC jumps to the address on the bus (16). The ISR there reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001.

5(b): After being read, P1 de-asserts *Int*.

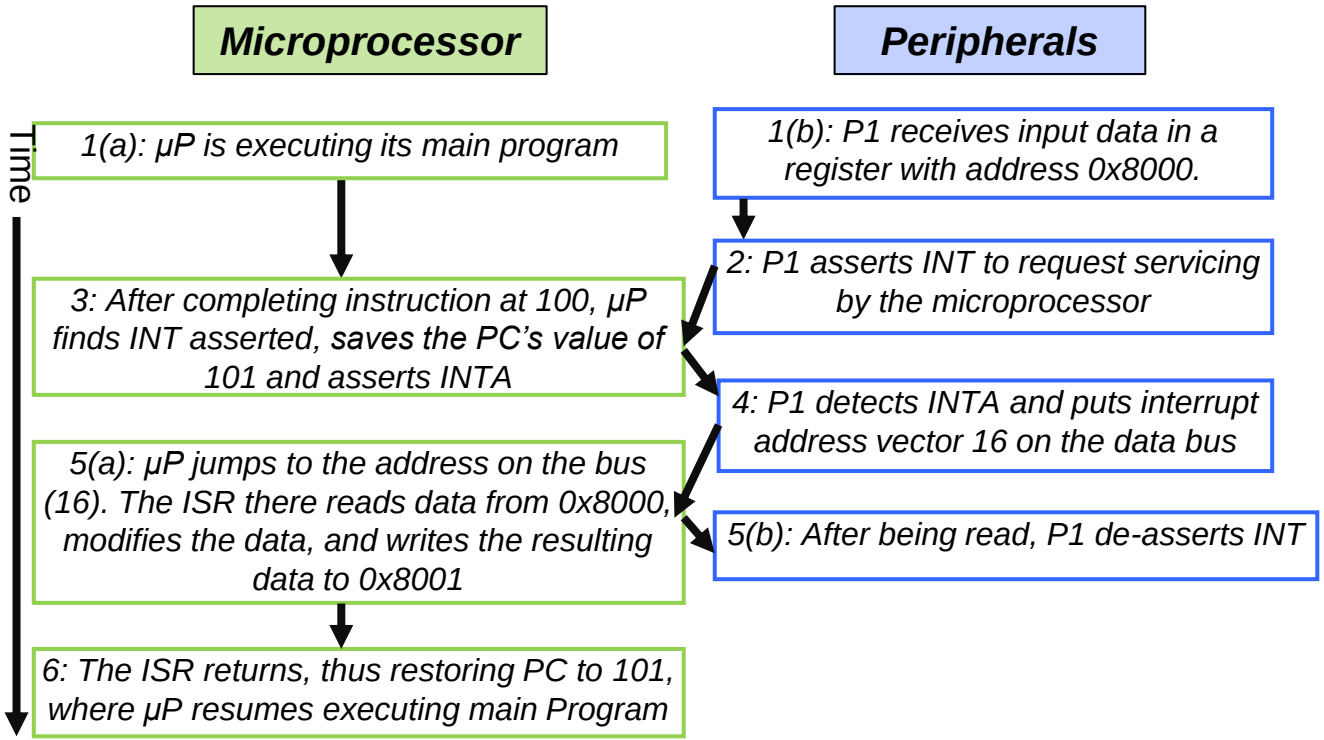


Interrupt-driven I/O using vectored interrupt

6: The ISR returns, thus restoring the PC to 101, from where the µP resumes its main program



Interrupt-driven I/O using Vectored Interrupt



Interrupt Address Table

- Compromise between Fixed and Vectored interrupts
  - One Interrupt Pin
  - Table in Memory holding ISR Addresses (maybe 256 words)
  - Peripheral doesn't provide ISR Address, rather only gives index into the table
    - Fewer bits are sent by the peripheral
    - Can move ISR location without changing Peripheral

Additional Interrupt Issues

- **Maskable vs. Non-maskable Interrupts**
  - **Maskable:** programmer can set bit that causes processor to ignore interrupt
    - Important when in the middle of a time-critical code
  - **Non-maskable:** a separate interrupt pin that can't be masked
    - Typically reserved for drastic situations, like power failure requiring immediate backup of data to non-volatile memory

- **Jump to ISR**
  - Some microprocessors treat jump same as call of any subroutine
    - Complete state saved (PC, registers) – may take hundreds of cycles
  - Others only save partial state, like PC only
    - Thus, ISR must not modify registers, or else must save them first
    - Assembly-language programmer must be aware of which registers stored

Design issues for Interrupts

- Two design issues arise in implementing interrupt I/O:

Which Device issues the interrupt?

If multiple interrupts have occurred, how processor will handle that?

Device Identification

- Four techniques are commonly used:

1. **Multiple Interrupt Lines**: Each device has multiple interrupt lines, the processer will pick the highest priority lines
2. **Software Poll:** A subroutine polls the interrupts according to the priority
3. **Daisy Chain:** The priority of a module is determined by the position of the module in the daisy chain, i.e. A module nearer to the processor in the chain has higher priority
4. **Bus Arbitration:** An I/O module must first gain control of the bus before it can raise the interrupt request line. Thus, only one module can raise the interrupt line at a time.

Drawbacks of Prog. and Int. Driven I/O

- Both Programmed and Interrupt driven I/O suffers from these two drawbacks:

1. The I/O transfer rate is limited by the speed with which the processor can test and service a device

2. The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer

Direct Memory Access

- An Additional module is added on system bus to mimic the processor operation to directly transfer data from I/O to memory

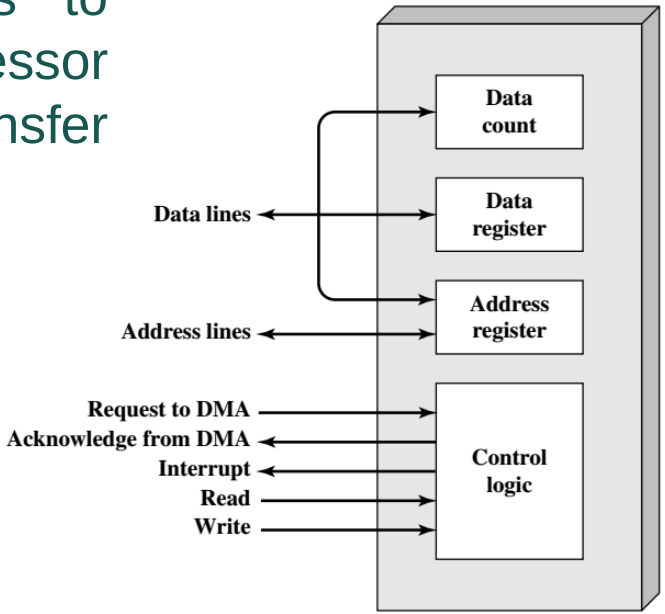


Figure 7.11 Typical DMA Block Diagram

Questions?

THANK YOU!