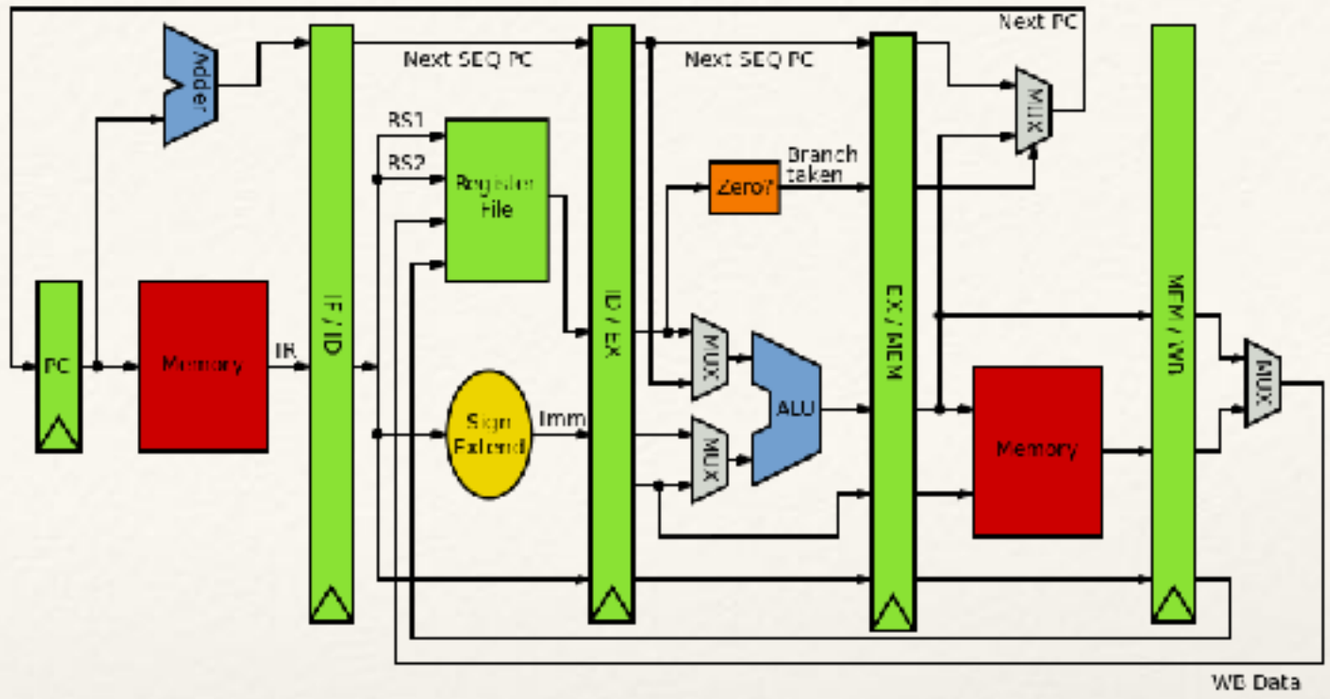


COMP2120 Computer Organization

Input/Output
Chapter 7

Dr. Ronald H.Y. Chung

THE UNIVERSITY OF HONG KONG
DEPARTMENT OF
COMPUTER SCIENCE

Generic Model of an I/O Module

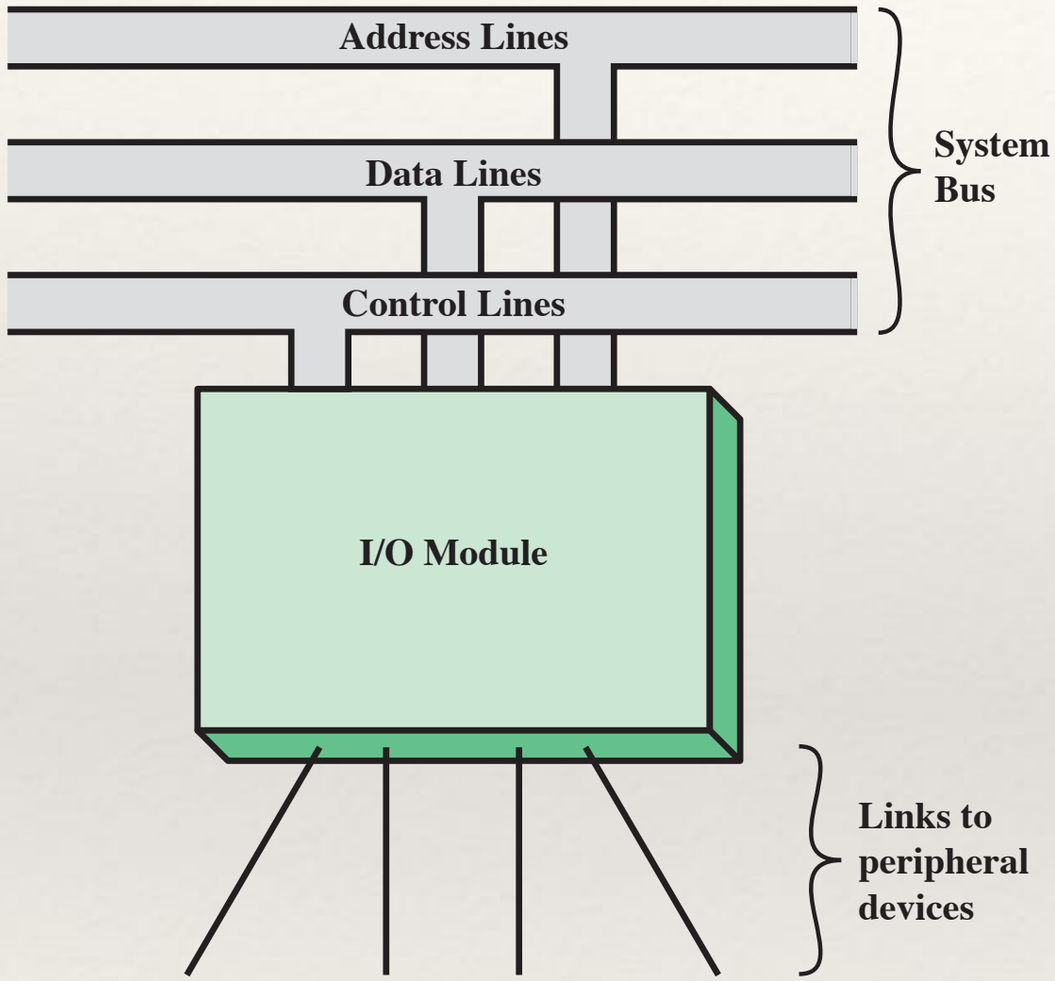


Figure 7.1 Generic Model of an I/O Module

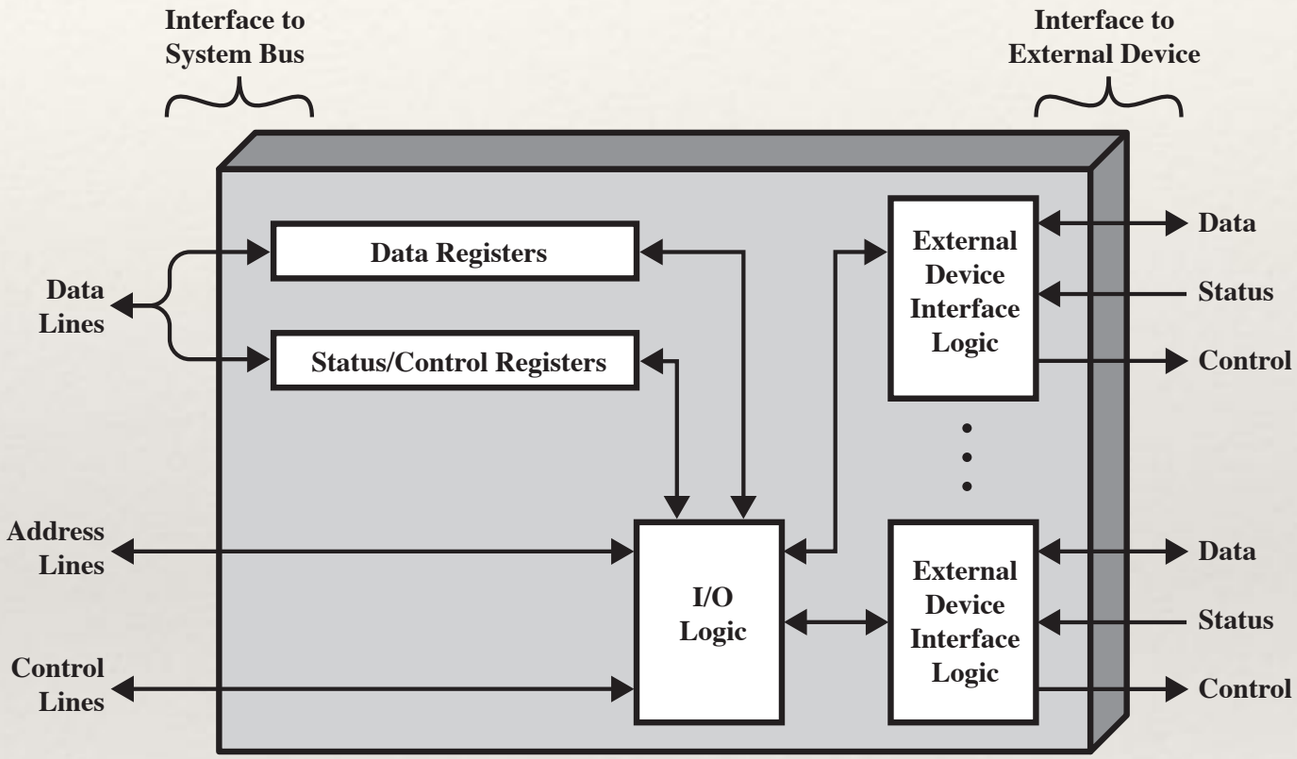
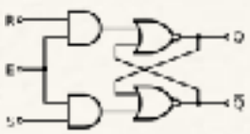
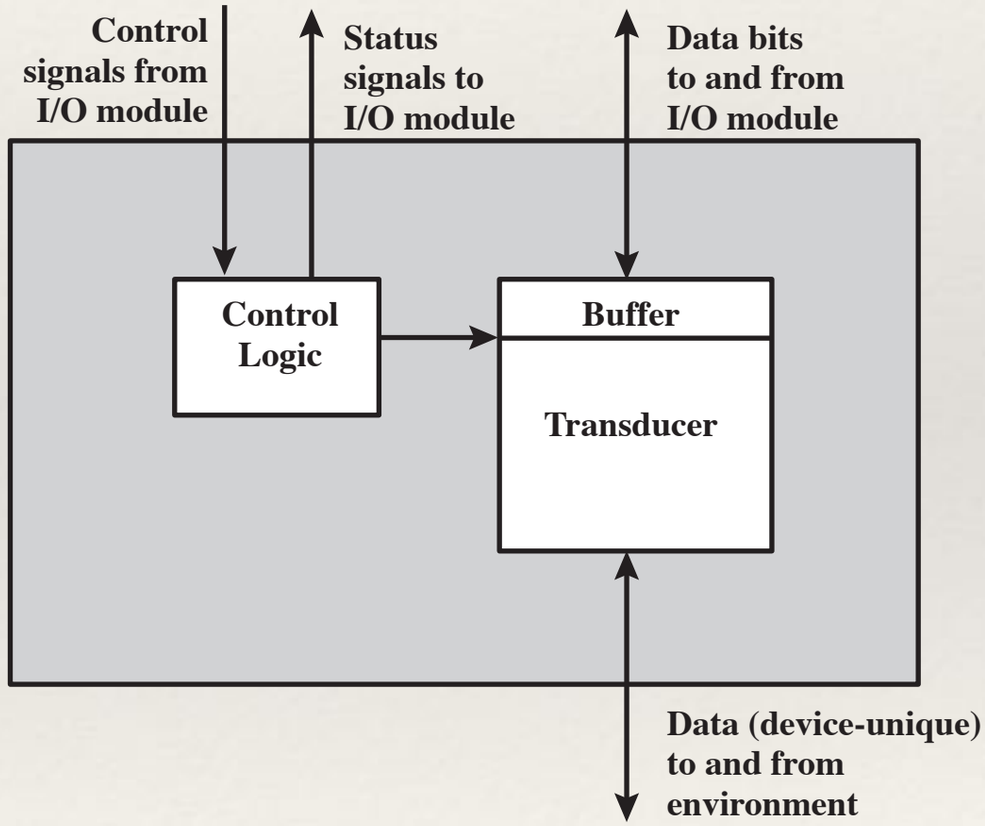


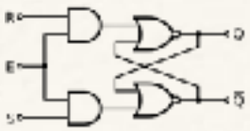
Figure 7.3 Block Diagram of an I/O Module



External Devices

- Provide a means of exchanging data between the external environment and the computer
- Attach to the computer by a link to an I/O module
  - The link is used to exchange control, status, and data between the I/O module and the external device
- Peripheral device
- An external device connected to an I/O module
- Three Categories:
  - Human readable
    - Suitable for communicating with the computer user
    - Video display terminals (VDTs), printers
  - Machine readable
    - Suitable for communicating with equipment
    - Magnetic disk and tape systems, sensors and actuators
  - Communication
    - Suitable for communicating with remote devices such as a terminal, a machine readable device, or another computer





Major Functions for an I/O Modules

The major functions for an I/O module fall into the following categories:

Control and timing

- Coordinates the flow of traffic between internal resources and external devices

Processor communication

- Involves command decoding, data, status reporting, address recognition

Device communication

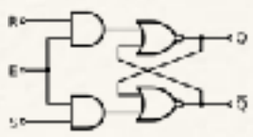
- Involves commands, status information, and data

Data buffering

- Performs the needed buffering operation to balance device and memory speeds

Error detection

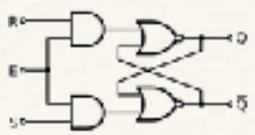
- Detects and reports transmission errors



I/O Techniques

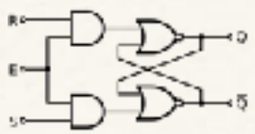
- Programmed I/O
  - Data are exchanged between the processor and the I/O module
  - Processor executes a program that gives it direct control of the I/O operation
  - When the processor issues a command it must wait until the I/O operation is complete
  - If the processor is faster than the I/O module this is wasteful of processor time
- Interrupt-driven I/O
  - Processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work
- Direct memory access (DMA)
  - The I/O module and main memory exchange data directly without processor involvement

	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)



I/O Commands

- There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:
- Control
  - Used to activate a peripheral and tell it what to do
- Test
  - Used to test various status conditions associated with an I/O module and its peripherals
- Read
  - Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer
- Write
  - Causes the I/O module to take an item of data from the data bus and subsequently transmit that data item to the peripheral



I/O Instructions

With programmed I/O there is a close correspondence between the I/O-related instructions that the processor fetches from memory and the I/O commands that the processor issues to an I/O module to execute the instructions

The form of the instruction depends on the way in which external devices are addressed

Each I/O device connected through I/O modules is given a unique identifier or address

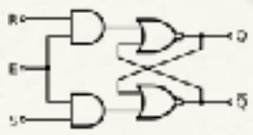
When the processor issues an I/O command, the command contains the address of the desired device

Thus each I/O module must interpret the address lines to determine if the command is for itself

Memory-mapped I/O

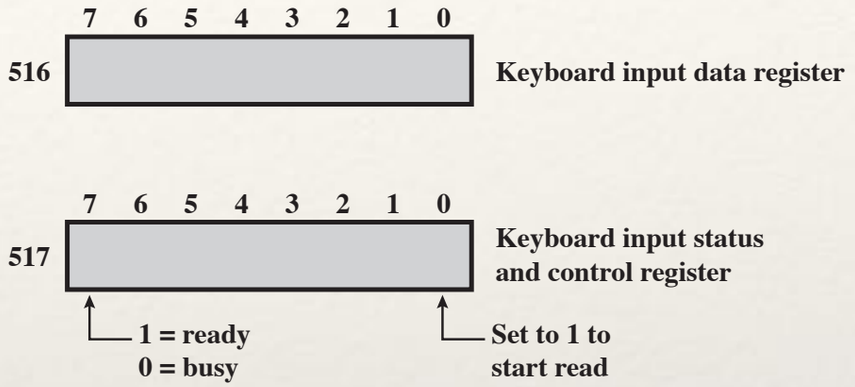
There is a single address space for memory locations and I/O devices

A single read line and a single write line are needed on the bus



I/O Mapping

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set

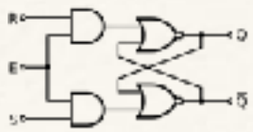


ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

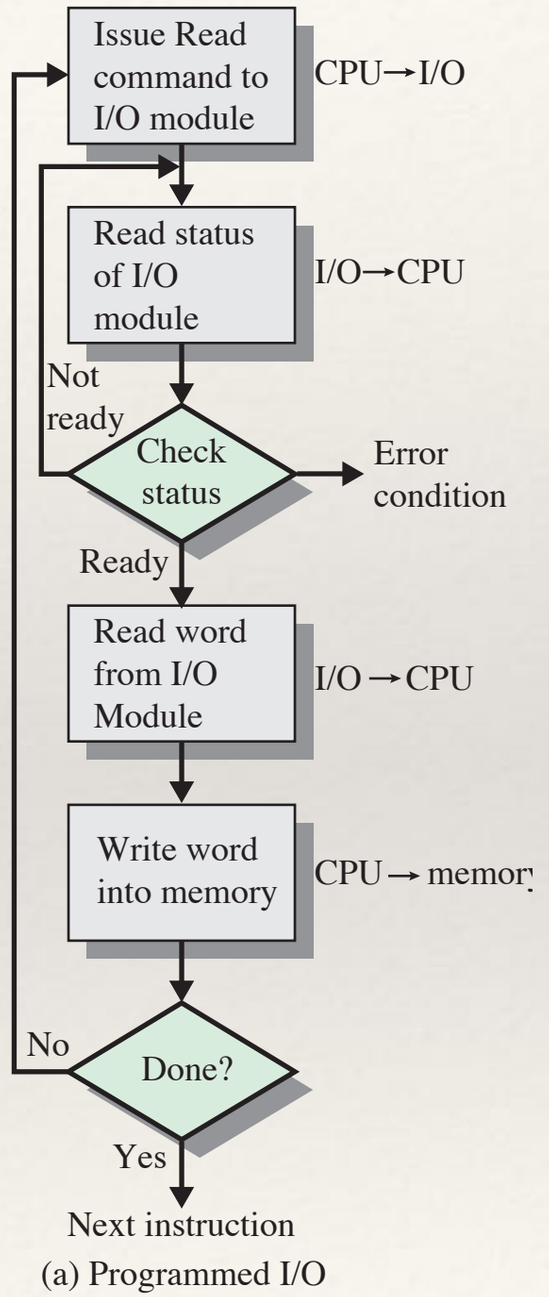
ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O



Programmed I/O

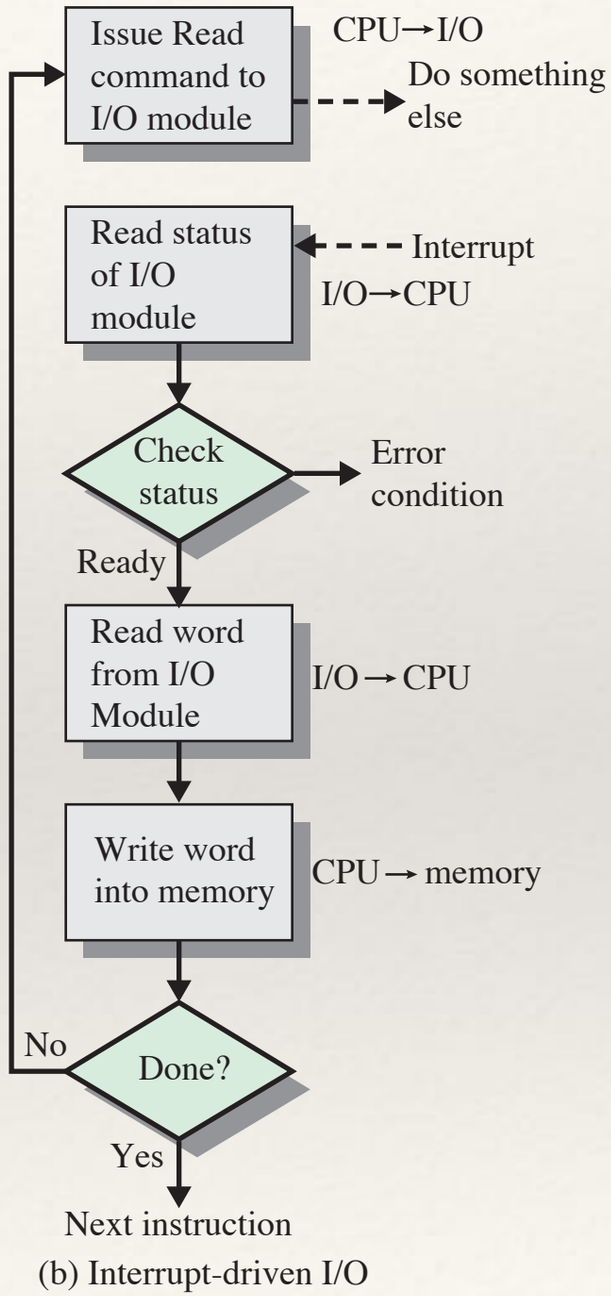
- Example: Read in a block of data from a peripheral device (e.g., a record from tape) into memory
- Data are read in one word (e.g., 16 bits) at a time. For each word that is read in, the processor must remain in a status-checking cycle until it determines that the word is available in the I/O module's data register
- Main disadvantage of this technique
  - It is a time-consuming process that keeps the processor busy needlessly.



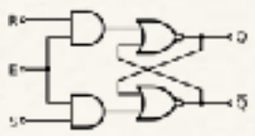
(a) Programmed I/O

Interrupt-Driven I/O

- The problem with programmed I/O is that the processor has to wait a long time for the I/O module to be ready for either reception or transmission of data
- An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work
- The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor
- The processor executes the data transfer and resumes its former processing

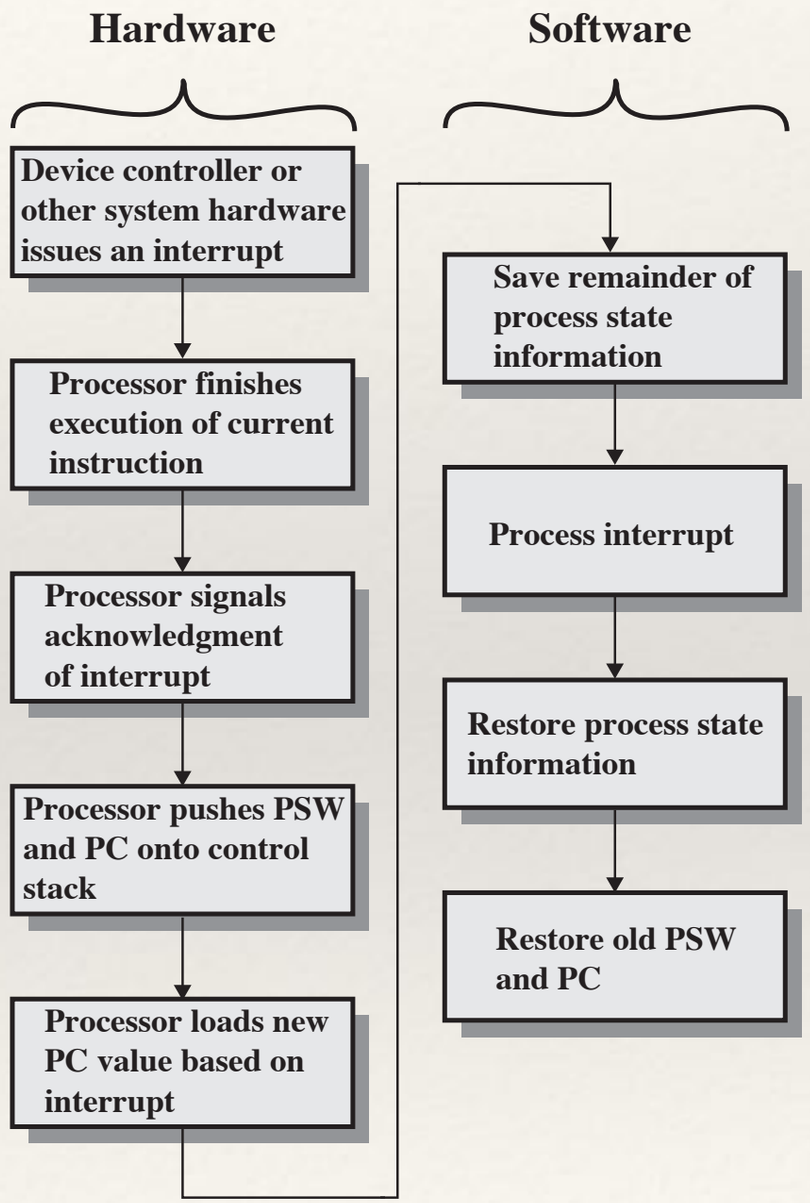


(b) Interrupt-driven I/O



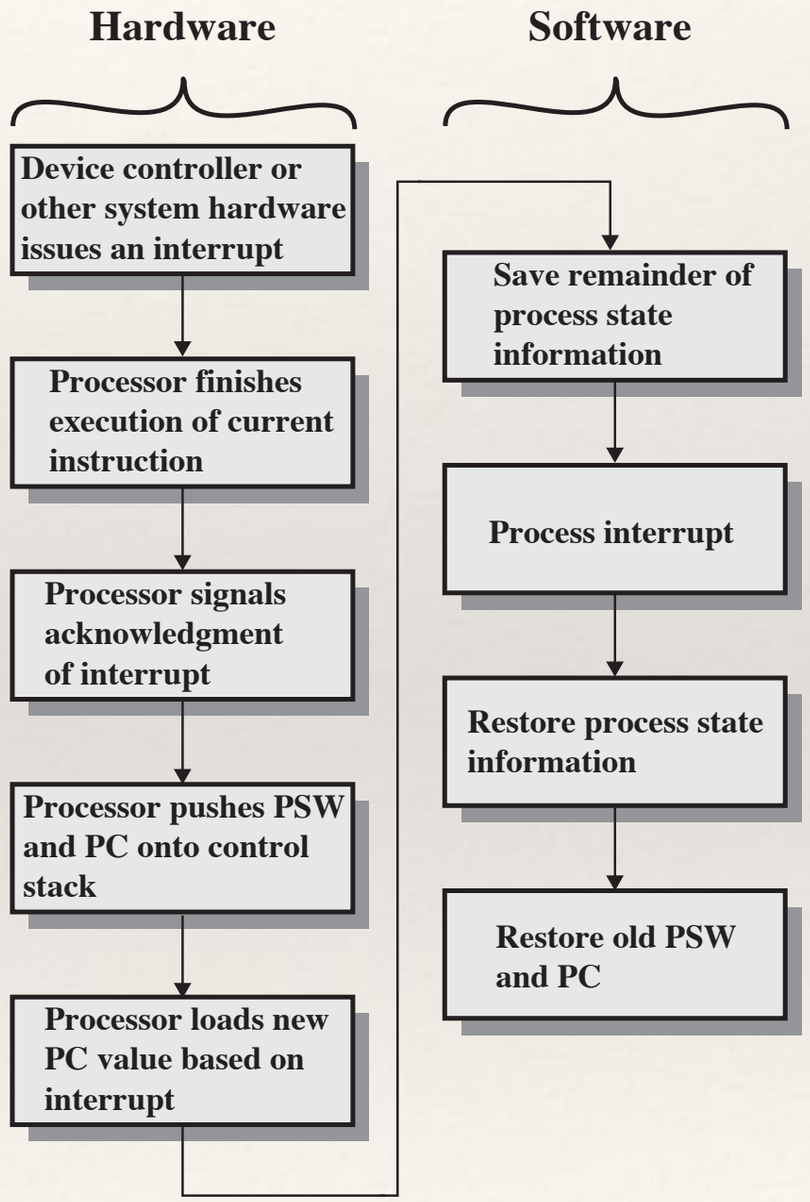
Interrupt Processing - Hardware Side

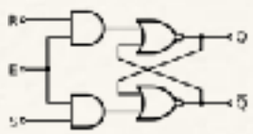
1. Device issue an interrupt signal to processor processor finish current instruction execution
2. Processor check for interrupt, find one, send an interrupt acknowledge (INTA) signal to device.
3. Device remove interrupt.
4. Processor need to remember the current position of the program before jumping to the interrupt servicing routine. This is done by putting the PC and the flag registers (Processor Status Word PSW) to the stack.
5. Processor then load PC with address of interrupt routine.



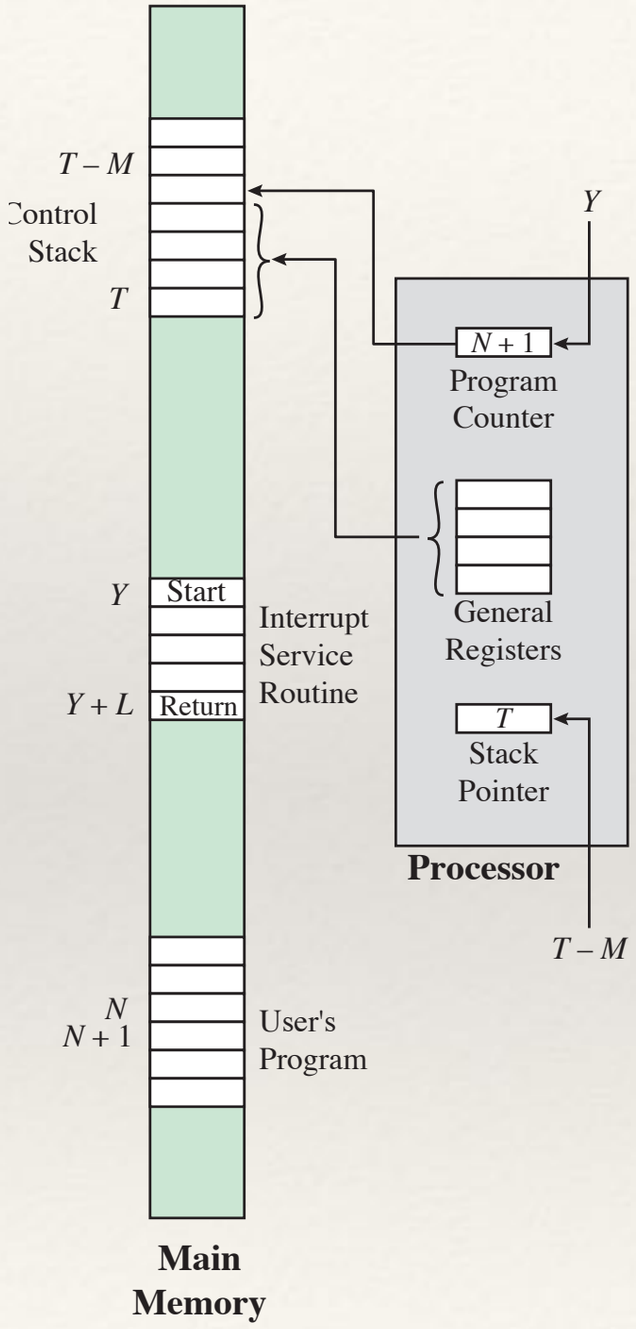
Interrupt Processing - Software Side

1. The Interrupt routine will save those registers that have been used by the interrupt routine into the stack (so that they can be recovered).
2. The Interrupt routine perform required action, e.g. Reading data from device.
3. When finished, the interrupt routine will restore the saved registers.
4. The last instruction executed by the interrupt routine is a Return from Interrupt instruction (RETI), which will restore the PSW and PC from the stack, in the reverse order that they were saved.

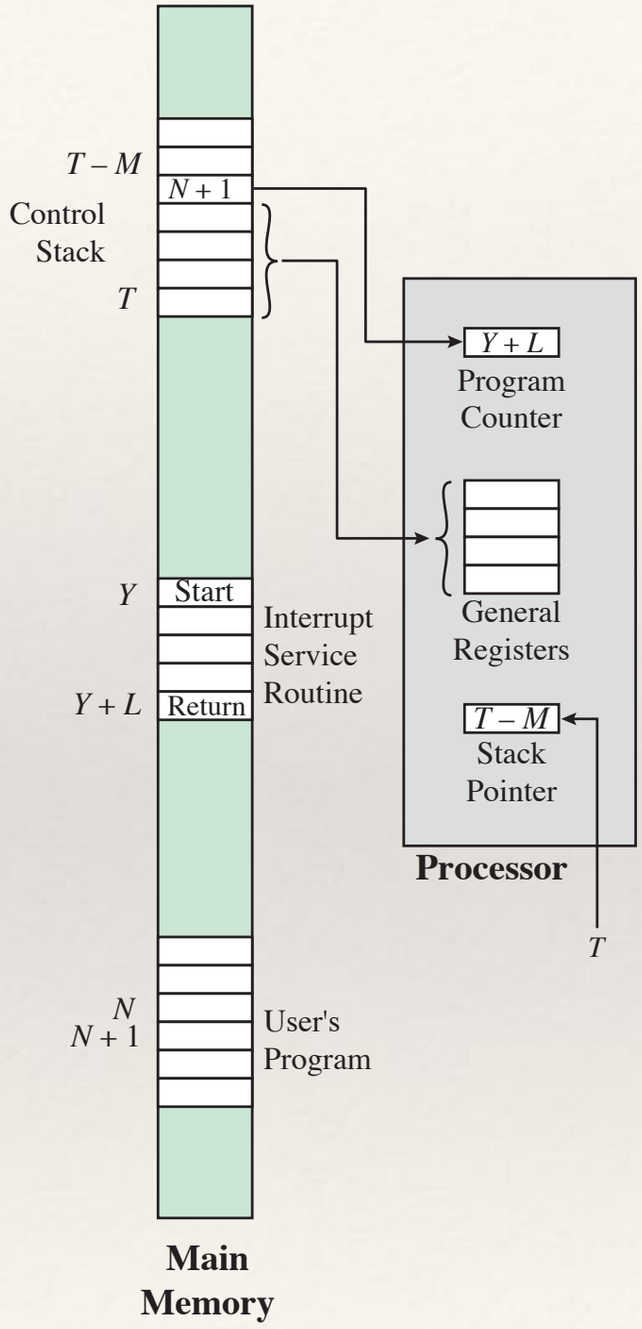




Changes in Memory and Registers for an Interrupt



(a) Interrupt occurs after instruction at location N

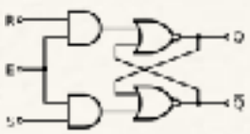


(b) Return from interrupt

Design issues of Interrupt I/O:

Two design issues arise in implementing interrupt I/O:

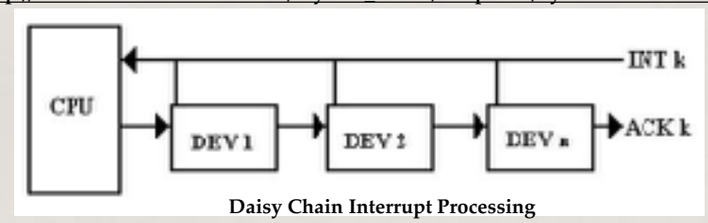
- Because there will be multiple I/O modules how does the processor determine which device issued the interrupt?
- If multiple interrupts have occurred how does the processor decide which one to process?



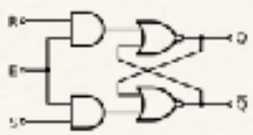
Device Identification

- Four general techniques
  - Multiple interrupt lines
    - Between the processor and the I/O modules
    - Most straightforward approach to the problem
    - Consequently even if multiple lines are used, it is likely that each line will have multiple I/O modules attached to it
  - Software poll
    - When processor detects an interrupt it branches to an interrupt-service routine whose job is to poll each I/O module to determine which module caused the interrupt
    - Time consuming
  - Daisy chain (hardware poll, vectored)
    - The interrupt acknowledge line is daisy chained through the modules
    - Vector – address of the I/O module or some other unique identifier
    - Vectored interrupt – processor uses the vector as a pointer to the appropriate device-service routine, avoiding the need to execute a general interrupt-service routine first
  - Bus arbitration (vectored)
    - An I/O module must first gain control of the bus before it can raise the interrupt request line
    - When the processor detects the interrupt it responds on the interrupt acknowledge line
    - Then the requesting module places its vector on the data lines

Source: http://www.edwardbosworth.com/My5155_Slides/Chapter12/SystemBusFundamentals.htm



Daisy Chain Interrupt Processing



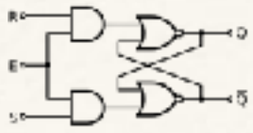
Drawbacks of Programmed and Interrupt-Driven I/O

- Both forms of I/O suffer from two inherent drawbacks:

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

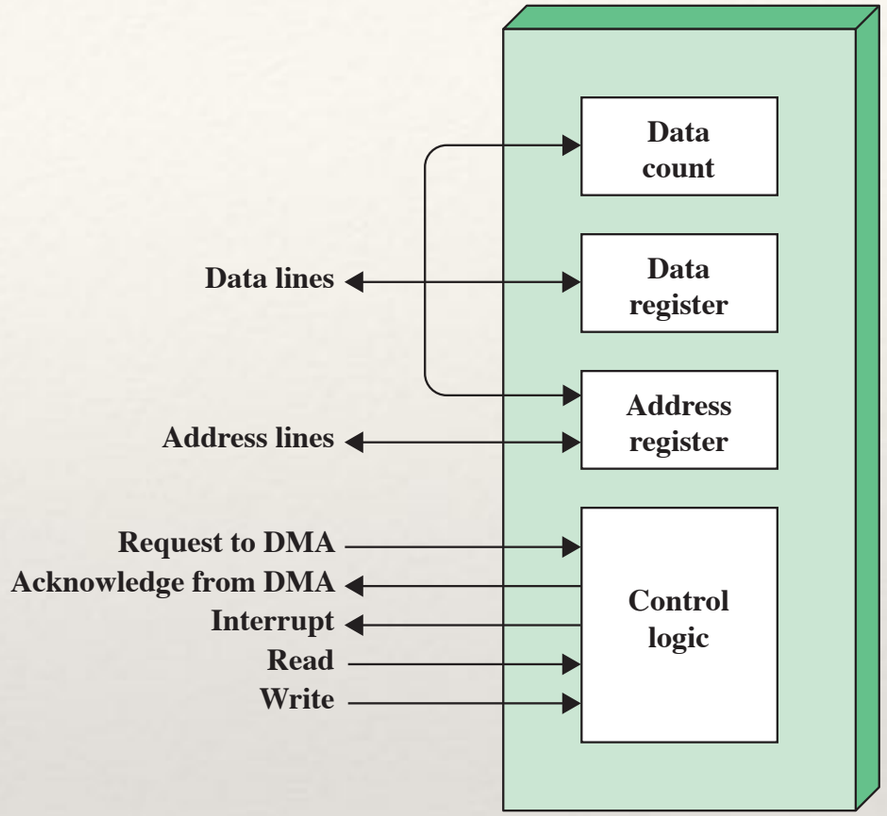
  2) The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer

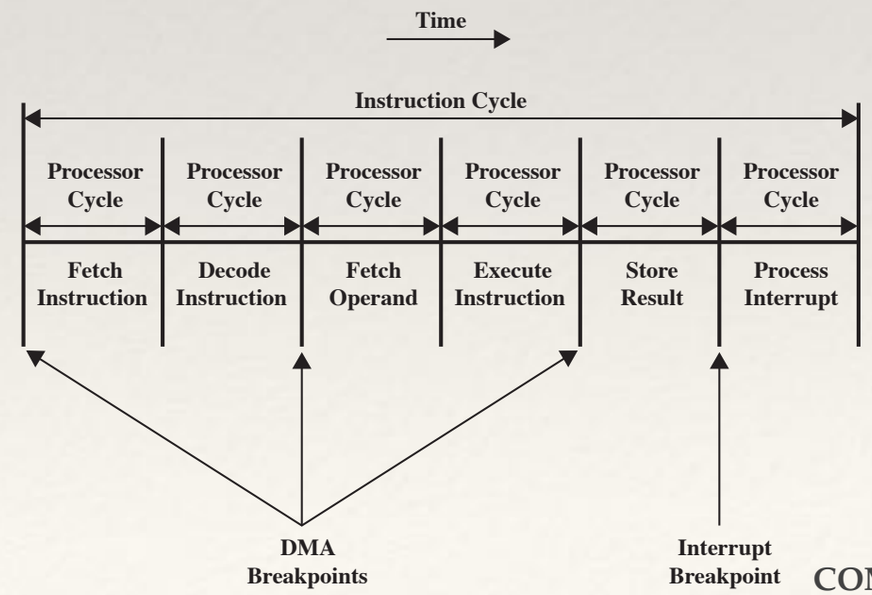
- When large volumes of data are to be moved a more efficient technique is ***Direct Memory Access*** (DMA)

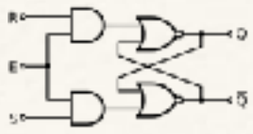


Direct Memory Access

- The DMA module is capable of mimicking the processor
  - i.e. Taking over control of the system from the processor
- The DMA module must use the bus only when the processor does not need it
  - In effect, DMA steals bus cycles (***cycle stealing***)
  - When DMA is using the bus, the processor that needs to use the bus must be suspended
  - When DMA finish using the bus, the processor that resume operations
  - The processor will see an elongated clock
- DMA operations
  - The processor will tell the DMA processor to perform the following, for example: Read the device and place the data in memory location x to y.
  - The processor then continues with other work. It has delegated this I/O operation to the DMA module
  - DMA module sends an interrupt signal to the processor when the transfer completes

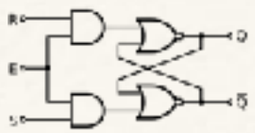






Evolution of I/O Function

1. The CPU directly controls a peripheral device.
2. A controller or I/O module is added. The CPU uses programmed I/O without interrupts.
3. Same configuration as in step 2 is used, but now interrupts are employed. The CPU need not spend time waiting for an I/O operation to be performed, thus increasing efficiency.
4. The I/O module is given direct access to memory via DMA. It can now move a block of data to or from memory without involving the CPU, except at the beginning and end of the transfer.
5. The I/O module is enhanced to become a processor in its own right, with a specialized instruction set tailored for I/O
6. The I/O module has a local memory of its own and is, in fact, a computer in its own right. With this architecture a large set of I/O devices can be controlled with minimal CPU involvement.



Chapter 7 - Summary

- Input/Output
  - I/O modules
    - Module function
    - I/O module structure
  - Programmed I/O
    - Overview of programmed I/O
    - I/O commands/ instructions
  - Interrupt-driven I/O
    - Interrupt processing
    - Design issues
  - Direct memory access
    - Drawbacks of programmed and interrupt-driven I/O
    - DMA function
  - The evolution of the I/O function