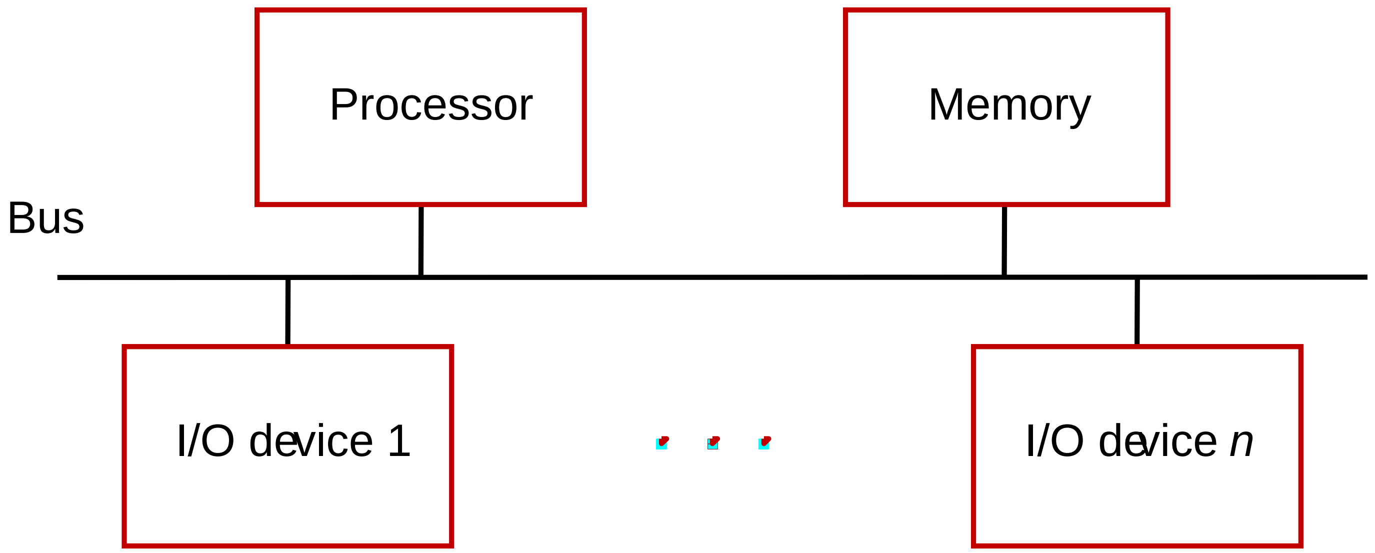
INPUT/OUTPUT ORGANIZATION

Accessing I/O Devices

Accessing I/O devices



- 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 recognizes the address, and responds to the control signals.

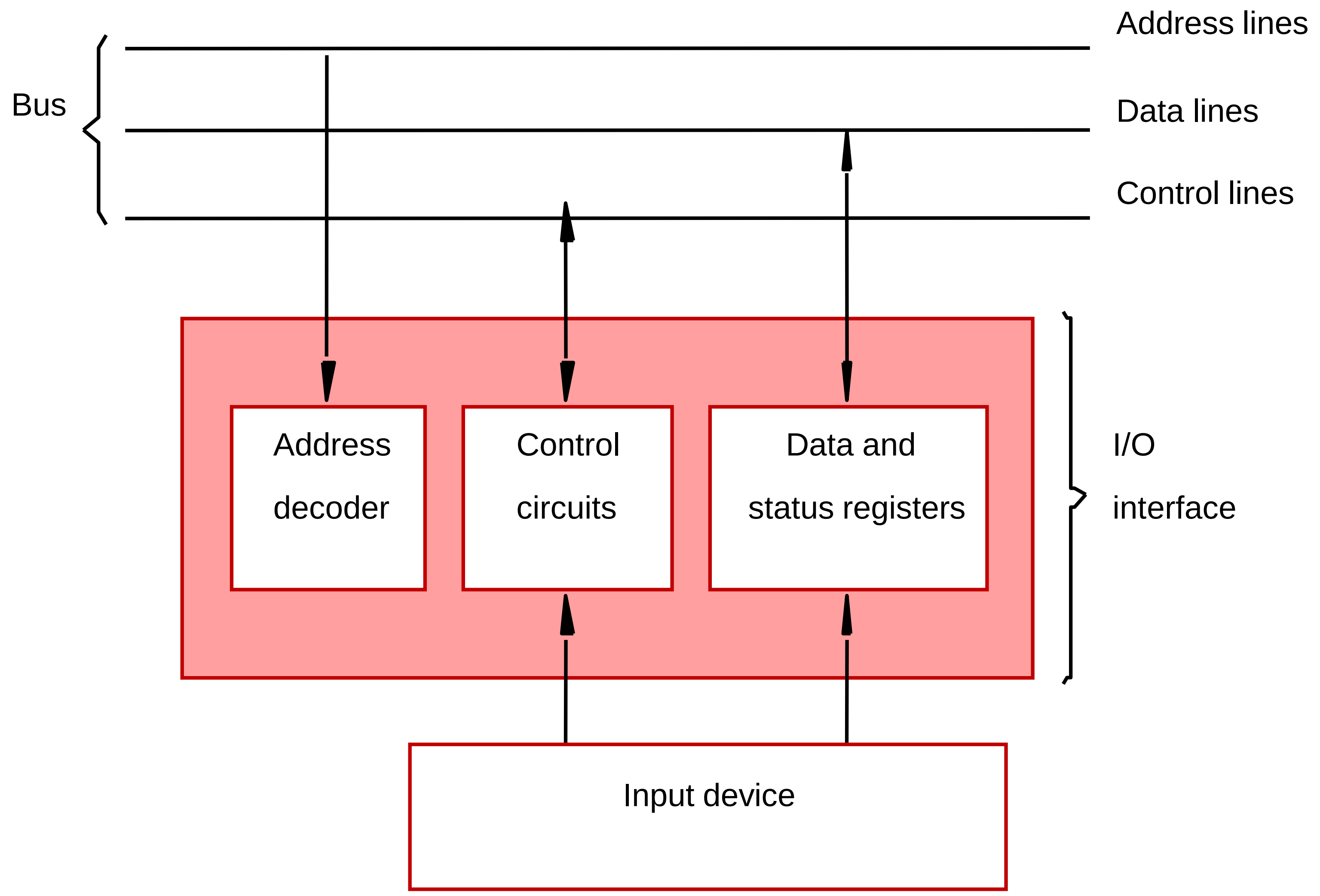
Accessing I/O devices (contd..)

- **I/O devices and the memory may share the same address space:**
  - **Memory-mapped I/O.**
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
  - Some addresses in the address space of the processor are assigned to these I/O locations-usually implemented as bit storage circuits (flip-flops) organized in the form of registers-*I/O registers*.
    - Load R2, DATAIN- reads the data from the DATAIN register and loads them into processor register R2.
    - Store R2, DATAOUT- sends the contents of register R2 to location DATAOUT, which is a register in an output device.

I/O devices and the memory may have different address spaces:

- Special instructions to transfer data to and from I/O devices.
- I/O devices may have to deal with fewer address lines.
- I/O address lines need not be physically separate from memory address lines.
- In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

Accessing I/O devices (contd..)



•.

•I/O device is connected to the bus using an I/O interface circuit which has:

- Address decoder, control circuit, and data and status registers.

•Address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.

•Data register holds the data being transferred to or from the processor.

•Status register holds information necessary for the operation of the I/O device.

•Data and status registers are connected to the data lines, and have unique addresses.

•I/O interface circuit coordinates I/O transfers

Accessing I/O devices (contd..)

- 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 transfers between them.
- **Program-controlled I/O:**
  - Processor repeatedly monitors a status flag to achieve the necessary synchronization.
  - Processor polls the I/O device.
- Two other mechanisms used for synchronizing data transfers between the processor and memory:
  - Interrupts.
  - Direct Memory Access.

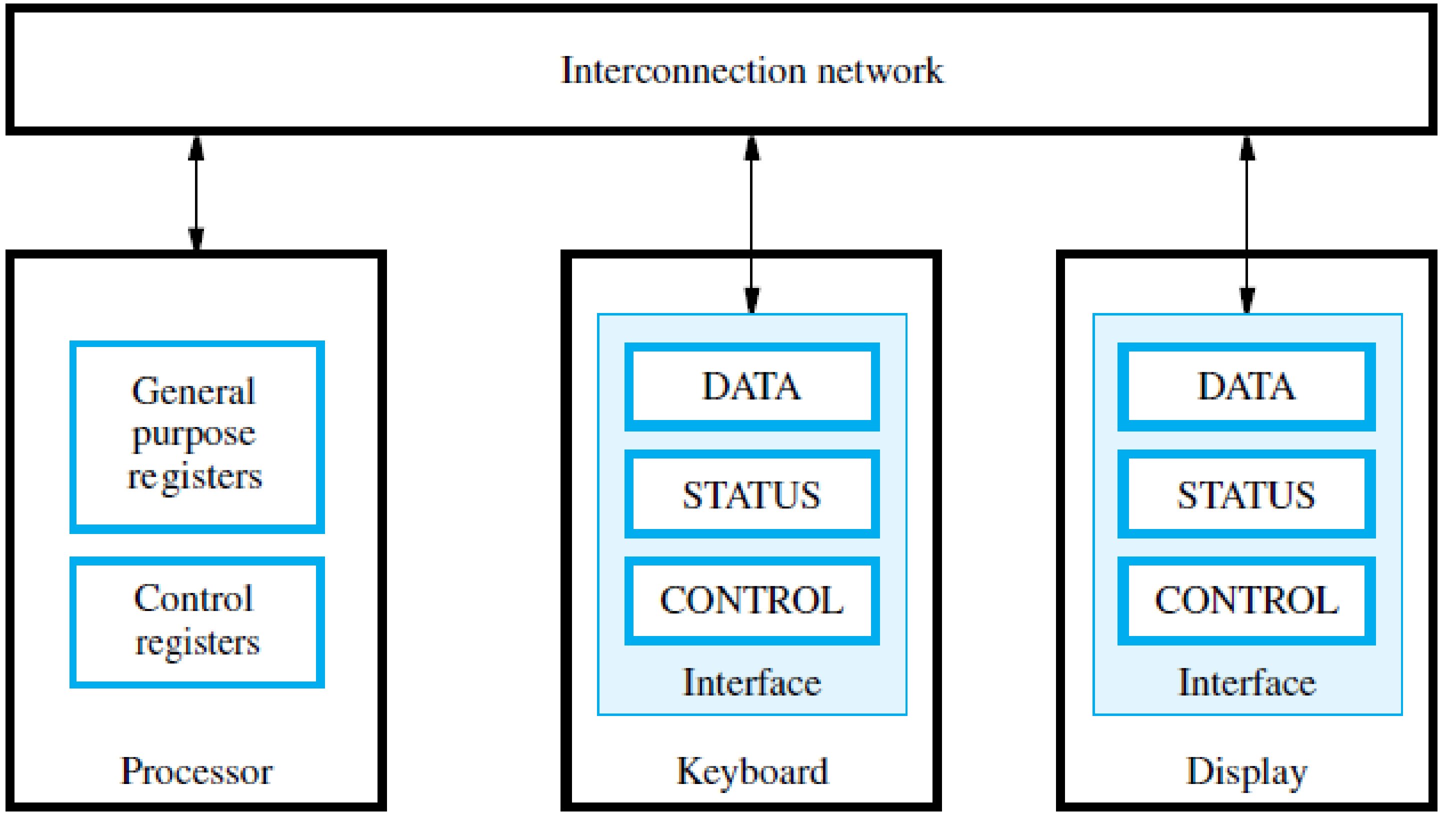


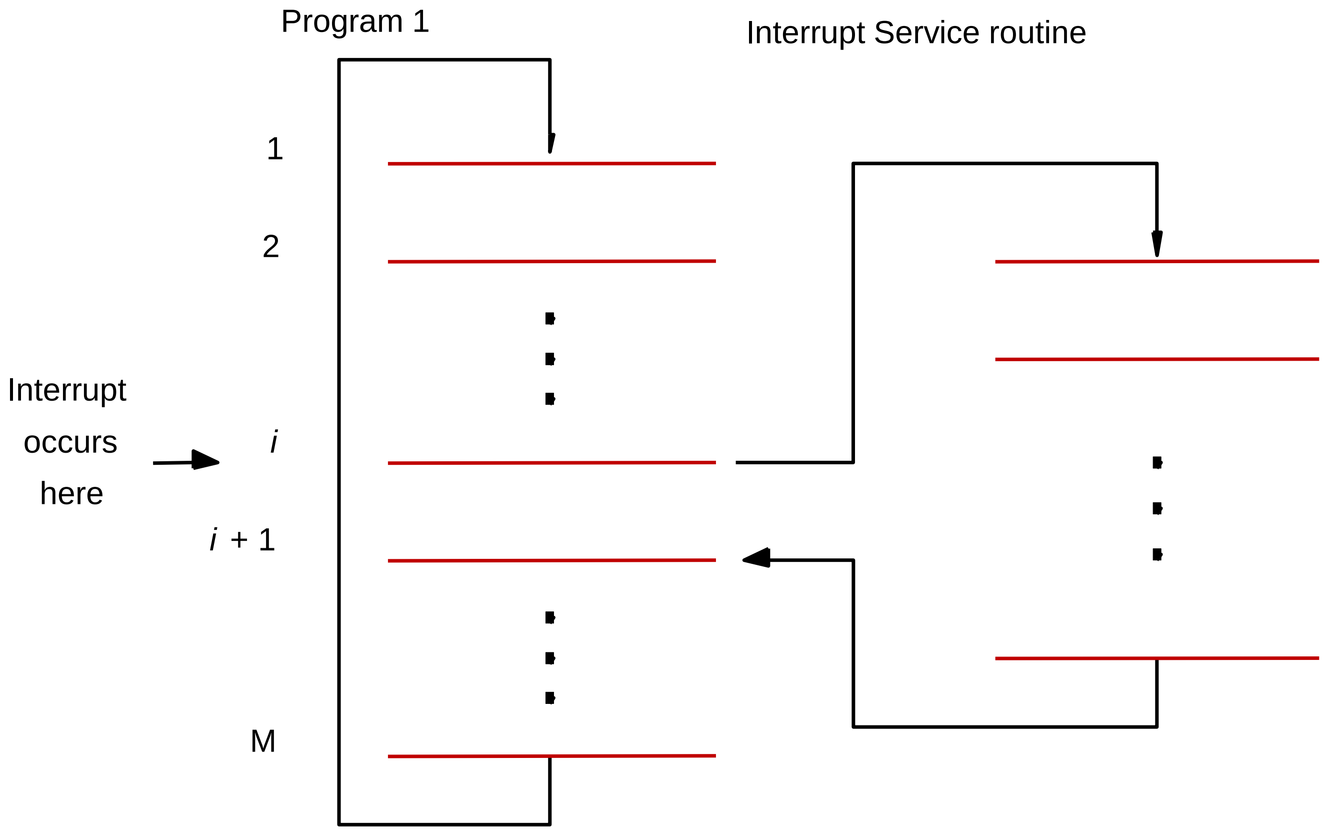
Figure 3.2 The connection for processor, keyboard, and display.

Interrupts

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.
  - At least 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**.

Interrupts (contd..)



Interrupts (contd..)

- 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 the PC are usually stored on the processor stack.

Interrupts (contd..)

- Processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal.
  - accomplished by means of a special control signal, called ***interrupt acknowledge*,** which is sent to the device through the interconnection network.
  - the transfer of data between the processor and the I/O device interface
    - The execution of an instruction in the interrupt-service routine that accesses the status or data register in the device interface implicitly informs the device that its interrupt request has been recognized.

Interrupts (contd..)

- **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.
  - 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.

Interrupts (contd..)

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.
- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called <u>interrupt latency</u>.
- 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 ISR
- In processors, with small number of registers, all registers are saved automatically by the processor hardware at the time an interrupt request is accepted and then restored to their respective registers as part of the execution of the Return-from-interrupt instruction

Interrupts (contd..)

- Some computers provide two types of interrupts
  - One saves all register contents, and the other does not.
- A particular I/O device may use either type, depending upon its response time requirements.
- Another interesting approach is to provide duplicate sets of processor registers.
  - a different set of registers can be used by the interrupt-service routine, thus eliminating the need to save and restore registers -*shadow registers*.

Enabling and Disabling of Interrupts

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as *Interrupt-enable* and *Interrupt-disable* for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be Interrupt-disable.
  - Last instruction of an interrupt service routine can be Interrupt-enable.

Sequence of events involved in handling an interrupt request from a single device

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed and saves the contents of the PC and PS registers.
3. Interrupts are disabled by clearing the IE bit in the PS to 0.
4. The action requested by the interrupt is performed by the interrupt-service routine, during which time the device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. Upon completion of the interrupt-service routine, the saved contents of the PC and PS registers are restored (enabling interrupts by setting the IE bit to 1), and execution of the interrupted program is resumed.

Handling 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?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

Handling multiple devices(contd..)

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?
- 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.

Handling multiple devices-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)to the processor over the bus.
- The processor’s circuits determine the memory address of the required interrupt-service routine.
- A commonly used scheme is to allocate permanently an area in the memory to hold the addresses of interrupt-service routines.
- These addresses are usually referred to as *interrupt vectors*, and they are said to constitute the *interrupt-vector table*

Vectored Interrupts(Contd..)

- When an interrupt request arrives, the information provided by the requesting device is used as a pointer into the interrupt-vector table, and the address in the corresponding interrupt vector is automatically loaded into the program counter

Interrupt Nesting

  - During the execution of an interrupt service routine of device, the processor does not accept interrupt requests from any other device.
  - Since the interrupt service routines are usually short, the delay that this causes is generally acceptable.
- However, for certain devices this delay may not be acceptable.
  - Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?

Interrupt Nesting (contd..)

- I/O devices are organized in a priority structure:
  - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.
- A priority level is assigned to a processor that can be changed under program control.

or

- 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.
  - Processor’s priority is encoded in a few bits of the processor status register.

Simultaneous Requests

- The processor must have some means of deciding 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.
- When vectored interrupts are used, we must ensure that only one device is selected to send its interrupt vector code. This is done in hardware, by using arbitration circuits

Direct Memory Access

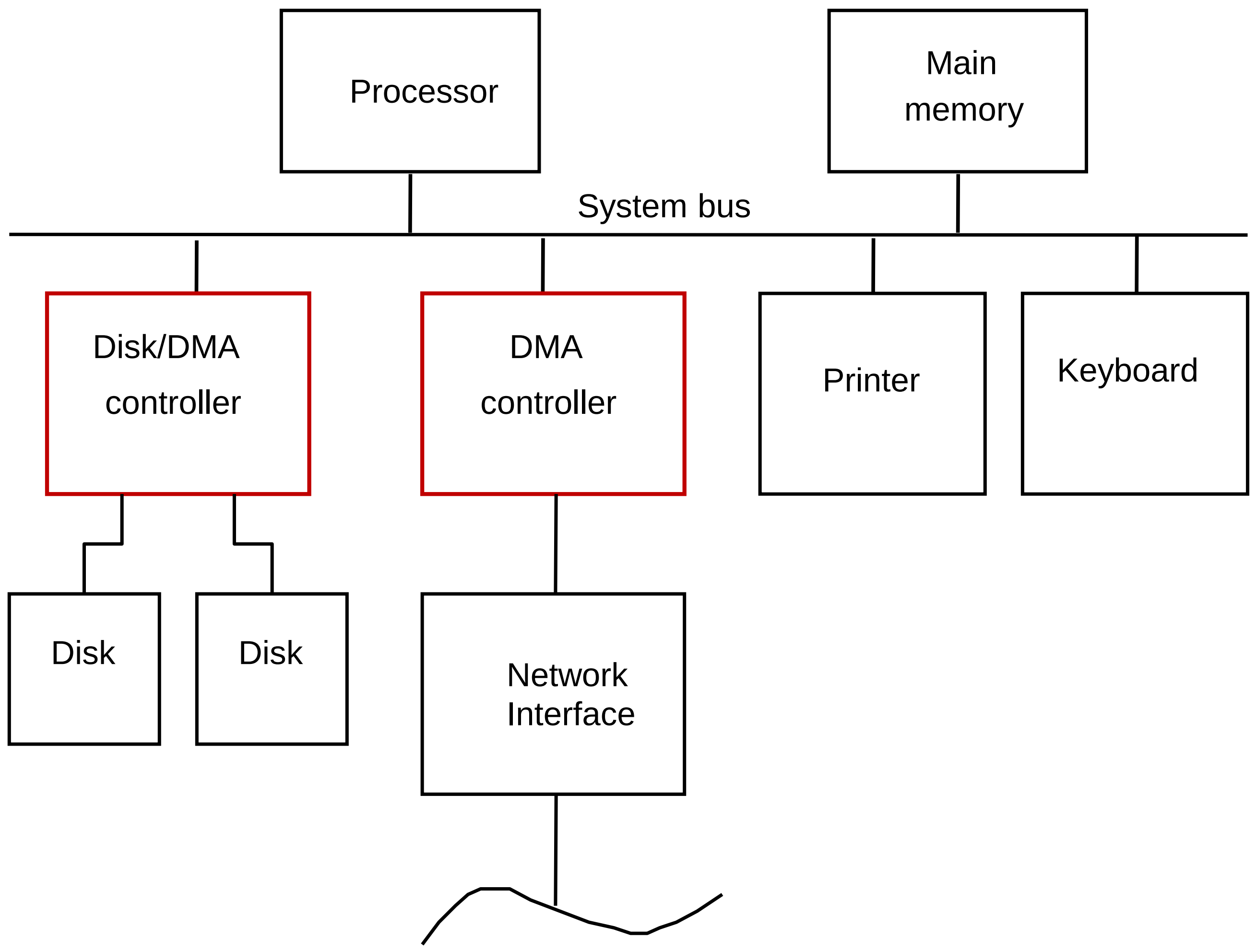
Direct Memory Access (contd..)

- Direct Memory Access (DMA):
  - A special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor.
- Control unit which performs these transfers is a part of the I/O device's interface circuit. This control unit is called as a DMA controller.
- DMA controller performs functions that would be normally carried out by the processor:
  - For each word, it provides the memory address and all the control signals.
  - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

Direct Memory Access (contd..)

- DMA controller can transfer a block of data from an external device to the main memory, without any intervention from the processor.
  - However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.
- To initiate the DMA transfer, the processor informs the DMA controller of:
  - Starting address,
  - Number of words in the block.
  - Direction of transfer (I/O device to the memory, or memory to the I/O device).
- Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

Direct Memory Access



Use of DMA controllers in a computer system

Use of DMA controllers in a computer system

- DMA controller connects a high-speed network to the computer bus.
- Disk controller, which controls two disks also has DMA capability.
- It provides two DMA channels.
- It can perform two independent DMA operations, as if each disk has its own DMA controller.
- The registers to store the memory address, word count and status and control information are duplicated.

Reference

- Carl Hamacher, Zvonko Vranesic,Safwat Zaky, Naraig Manjikian, *“Computer Organization And Embedded Systems”* 6th Edition, McGraw-Hill (Selected sections from Chapters 3and 8)