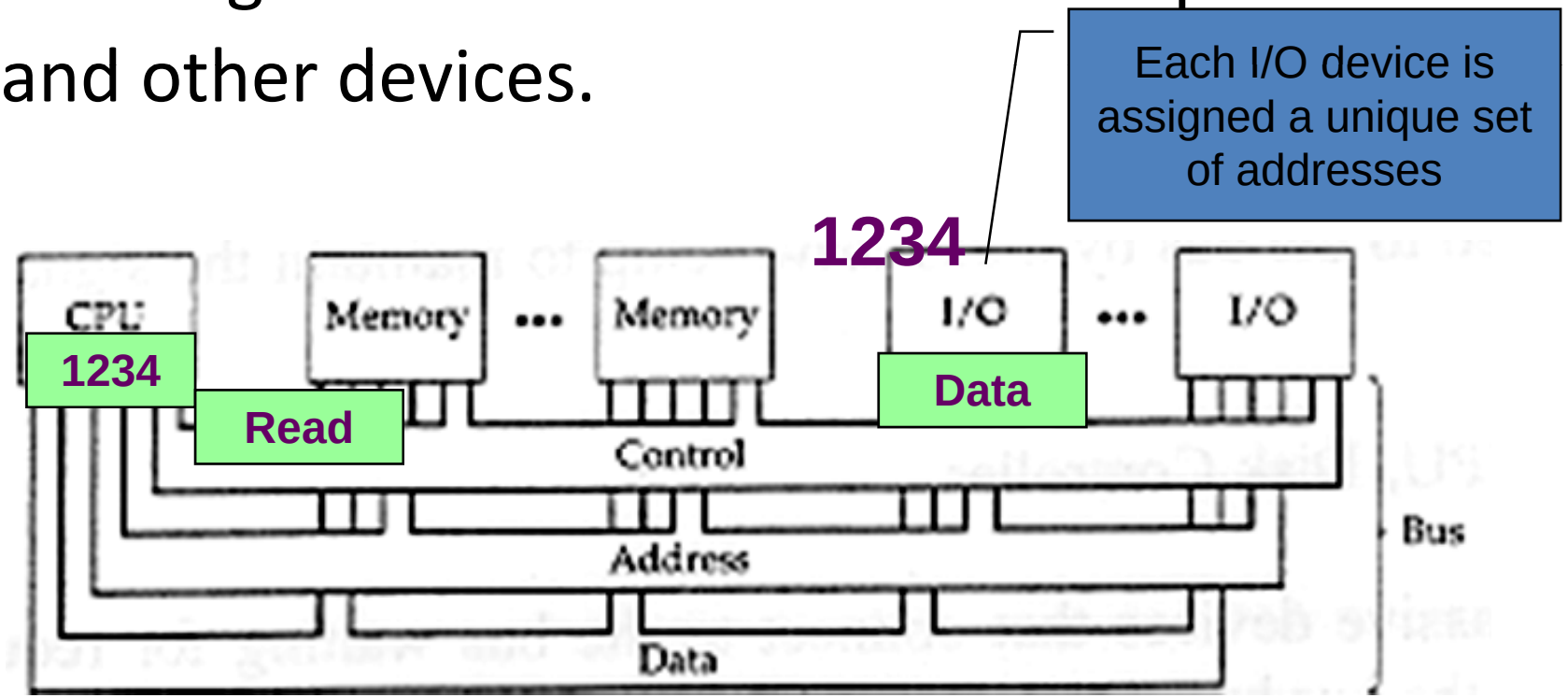
Contents

- Input / Output Operations
  - Program-controlled I/O
  - Interrupt-driven I/O
  - Direct memory access I/O
- Input / Output Interface
  - Parallel interface
  - Serial interface

Input / Output (I/O) Operations (1)

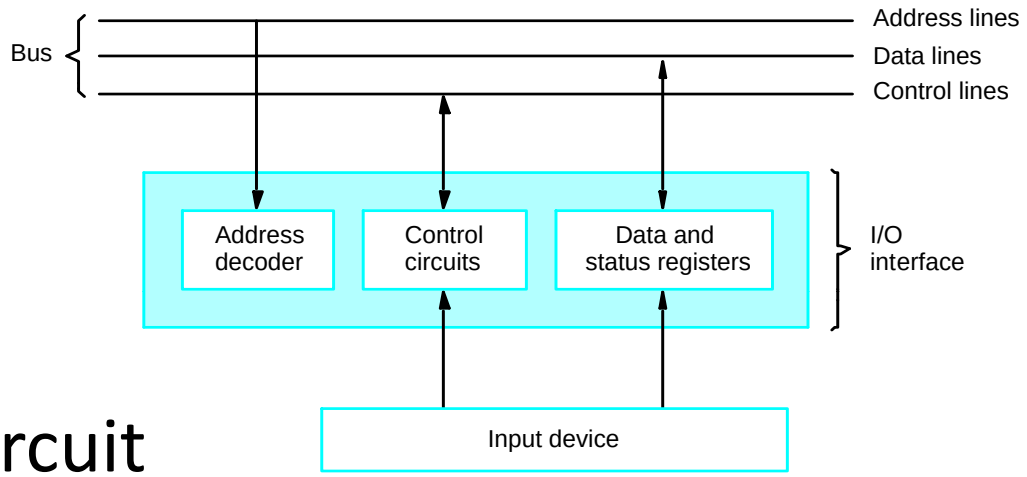
- Computers can perform I/O operations to exchange information with human operators and other devices.



I/O Operations (2)

- Usually, the CPU merely acts as an intermediate path when executing an I/O instruction but the actual data transfer is between memory unit and I/O devices.
- ***Memory-mapped I/O***
  - I/O devices and the memory share the same address space
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.

I/O Operations (3)



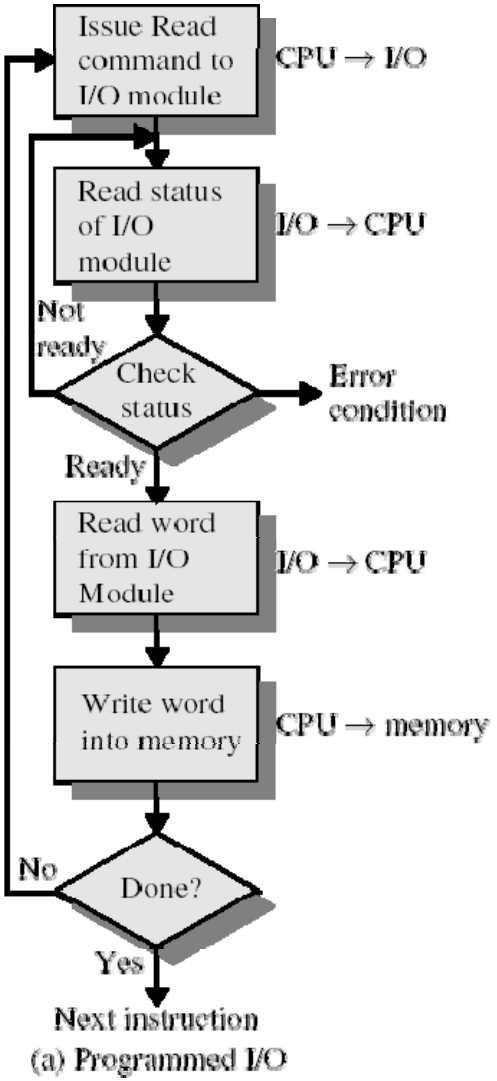
- I/O interface circuit
  - The address decoder enables the device to recognize its address.
  - The data register holds the data being transferred.
  - The status register contains information for the I/O operations.
  - The control circuits coordinate I/O transfers.

I/O Operations (4)

- The speeds of I/O devices are much slower than that of the processor, we must have some mechanisms to synchronize the transfer of data between them.
- Common used mechanisms for implementing I/O operations:
  - ***Program-controlled I/O***
  - ***Interrupt-driven I/O***
  - ***DMA I/O***

Program-controlled I/O (1)

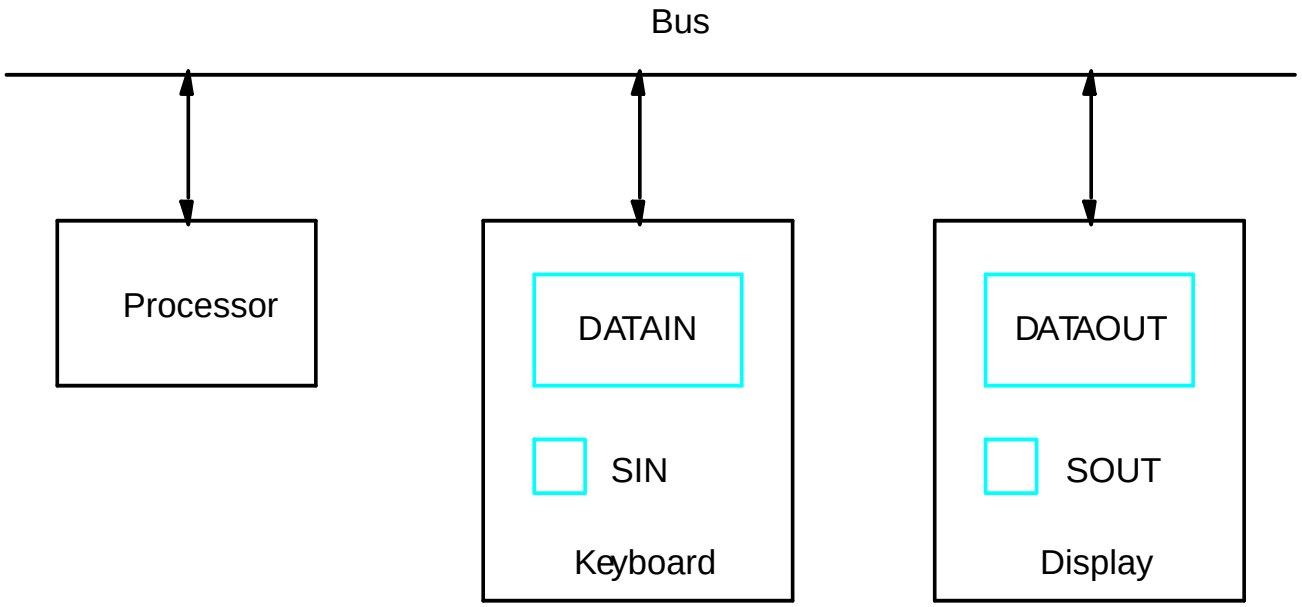
- In *program-controlled I/O*, the CPU repeatedly checks (polls) the I/O device status register to achieve the required synchronization between the CPU and the I/O device.
- Commonly used in low-end microprocessors such as embedded systems or in real-time systems.
- Disadvantage – the CPU wastes most of its time busy waiting for I/O.



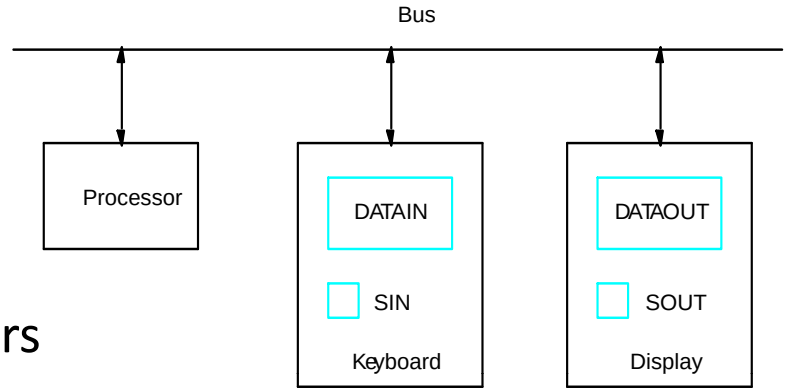
(a) Programmed I/O

Program-controlled I/O (2)

- A simple example: Read in character input from a keyboard and produce character output on a display screen.



Program-controlled I/O (3)



- DATAIN / DATAOUT: 8-bit data registers
- SIN: a status control flag, SIN=1 means a character is entered at the keyboard and is ready for processor to read.
- SOUT: a status control flag, SOUT=1 means that the display is ready to receive a character.
- The data registers and the status registers (INSTATUS and OUTSTATUS) containing the status control flags in bit 3 can be addressed as if they were memory locations.

Program-controlled I/O (4)

- An input operation from the keyboard.
  1. Striking a key stores a character in DATAIN.
  2. SIN is set to 1.
  3. The processor can monitor SIN as follows.

```
READ    BTST.W    #3,INSTATUS ; wait for a character to be entered in
        BEQ       READ                      ; the keyboard buffer DATAIN
        MOVE.B    DATAIN,D1   ; transfer the character from DATAIN
                                            ; into D1
```

  4. When SIN is set to 1, the processor transfers a character from DATAIN to D1.
  5. SIN is cleared to 0.
  6. If next character is entered at the keyboard, SIN is again set to 1.

Program-controlled I/O (5)

- An output operation to the display
  1. The processor monitors SOUT as follows.

```
WRITE   BTST.W  #3,OUTSTATUS    ; wait for the display to become ready
        BEQ     WRITE
        MOVE.B  D1,DATAOUT      ; move the character from D1 to the
                                ; output buffer DATAOUT
```

  2. When SOUT is set to 1, the processor transfers a character from D1 to DATAOUT to be displayed.
  3. SOUT is cleared to 0.
  4. When the display device is ready to receive the next character, SOUT is again set to 1.

Interrupts (1)

- Interrupt is a mechanism for diverting the attention of a processor when a particular event occurs, such as I/O device requests.
- Interrupts cause a break in the normal execution of a program.

Interrupts (2)

- (i) External Interrupts
  - I/O devices request transfer of data
  - a timing device indicates an elapsed time of event or exceeded time allocation for an endless looping of a program
  - external hardware circuits such as power supply failure, system reset or bus error
  - asynchronous in operation between devices
  - event driven, independent of the program in execution

Interrupts (3)

- (ii) Internal Interrupts (***Traps***)
  - arise from illegal or erroneous use of an instruction or data
    - examples: divide by zero, stack overflow
  - the interrupt service program (***trap handler***) determines the corrective action to be taken, e.g., print an error message
  - internal interrupts are initiated by some exceptional conditions caused by program
  - synchronous in operation with the program
  - if re-run program, event will occur in the same place

Interrupts (4)

- (iii) Software Interrupts
  - initiated by executing a ***privileged instruction*** as a special call instruction (behaves as an interrupt)
  - programmers often use software interrupts to initiate an interrupt procedure at a desired point in the program
    - e.g., a switch from the user mode (user program execution) to supervisor mode for input or output transfer operation by means of a *supervisor call instruction*. The user program passed data to the system program (supervisor mode) to specify the requested I/O task.

Steps taken to process Interrupt (1)

1. A device raises an interrupt request by sending a hardware signal in one of the bus control lines, called an *interrupt-request* line, to the processor.
2. The processor completes execution of the current instruction.
3. The processor *determines* the device requesting an interrupt and sends an interrupt-acknowledge signal to the device.
4. The device removes its interrupt-request signal.

Steps taken to process Interrupt (2)

5. The status of the processor (Status register) and the location of the next instruction to be executed (Program Counter) is saved on the processor stack.
   - The amount of information saved automatically by the processor should be kept to a minimum to avoid long ***interrupt latency***.
6. The processor loads the PC with the address of the first instruction of the Interrupt Service Routine (ISR) associated with the device (causing the interrupt).

Steps taken to process Interrupt (3)

7. Process the ISR.
   - Any additional information such as the contents of the registers must be saved at the beginning of the ISR and restored at the end of the ISR.
8. Restore the status register and program counter.
9. Resume the execution of the interrupted program.

Determining Source of Interrupt (1)

- If a number of devices capable of initiating interrupts are connected to the processor, how can the processor recognize the device requesting an interrupt?

Determining Source of Interrupt (2)

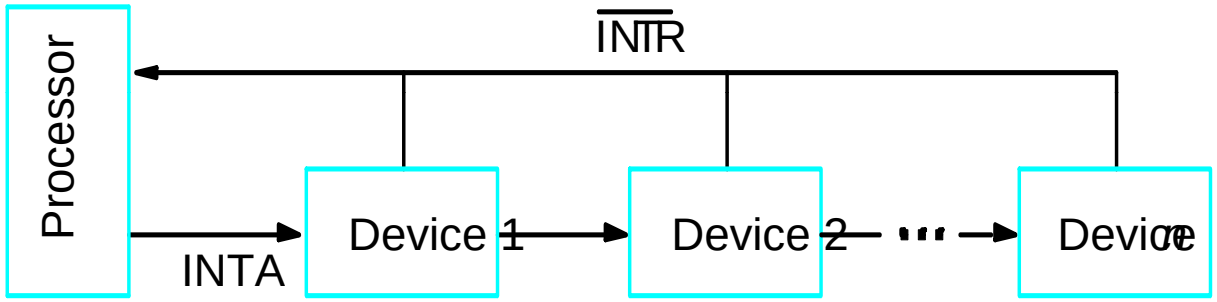
- (i) Polling
  - When a device raises an interrupt request, it sets an interrupt request bit (IRQ) in its status register to 1.
  - Whenever an interrupt is generated, all the I/O devices are polled.
  - The first device with its IRQ bit set is the device to be serviced.
  - The order in which the devices are polled determines the priorities of the devices.
  - Advantage: simple and easy to implement
  - Disadvantage: time is spent testing the IRQ bits of all the devices that may not be requesting any service.

Determining Source of Interrupt (3)

- (ii) Vectored interrupts
  - A device requesting an interrupt identifies itself by sending a vector to the processor upon the interrupt acknowledgement.
  - This vector is used as a pointer to the ISR for that device.

Determining Source of Interrupt (4)

– To ensure only one device is selected to send its vector, devices can be connected to form a daisy chain.

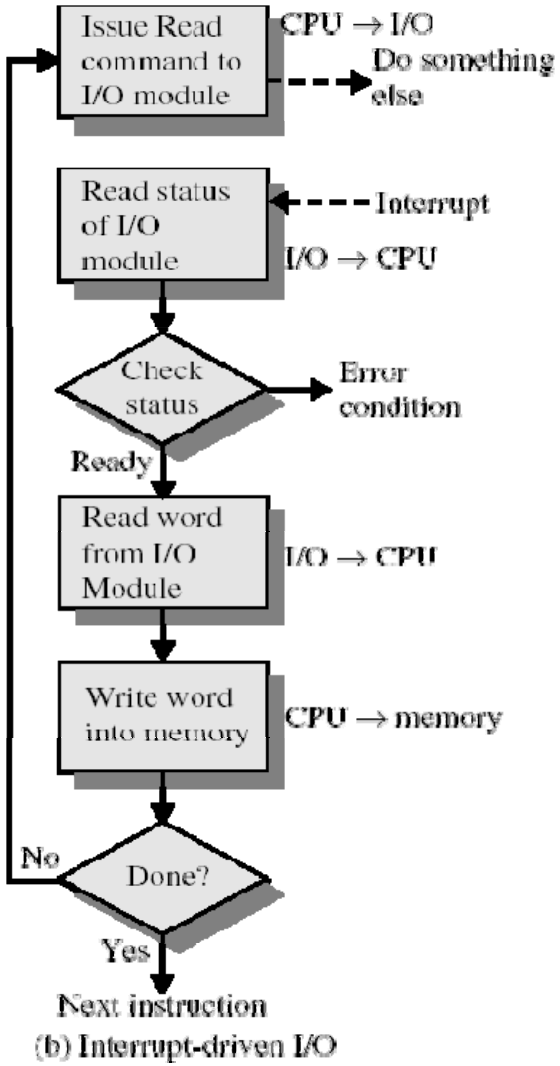


– The interrupt-acknowledge signal propagates serially through the devices.

– The device with a pending request for interrupt blocks the signal and sends its vector to the processor.

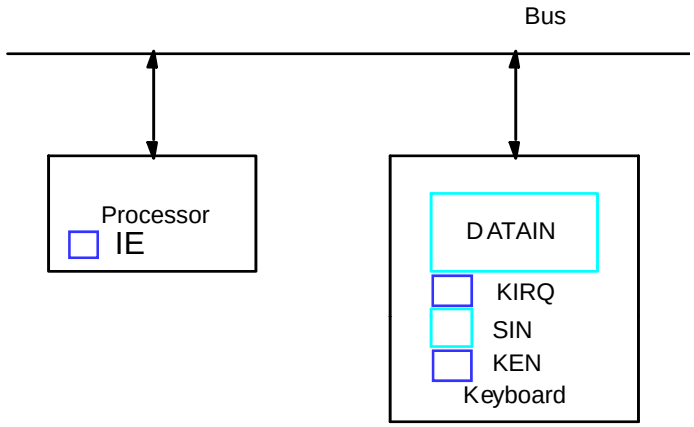
Interrupt Driven I/O (1)

- Better than programmed I/O, but too many interrupts occur for every byte of input and output.
- Processing an interrupt is expensive.



Interrupt Driven I/O (2)

- Example: In a program called MAIN, it reads a line from the keyboard and store it in the memory, starting at location LINE.



  - Refer to the simple example used in program-controlled I/O, the following information is added.
    - KEN: keyboard interrupt-enable bit
    - If KEN is set, the interface circuit generates an interrupt request whenever the status flag SIN is set.
    - KIRQ: indicates that the keyboard is requesting an interrupt
    - IE: interrupt-enable bit at the processor.

Interrupt Driven I/O (3)

- Initialization
  - Load the starting address of the ISR.
  - Load the address LINE.
  - Enable keyboard interrupts by setting KEN to 1.
  - Enable interrupts in the processor by setting IE to 1.
- When a character is typed on the keyboard, an interrupt request will be generated.
- The program being executed will be interrupted.

Interrupt Driven I/O (4)

- The ISR will be executed as follows.
  - Read the input character from DATAIN. This will cause the interface circuit to remove its interrupt request.
  - Store the character in the memory.
  - When the end of the line is reached, clear the KEN bit to disable keyboard interrupts and inform the program MAIN.
  - Return from interrupt.

Why DMA?

- Used for high-speed block transfers between a device and memory
- During the transfer, the CPU is not involved
- Typical DMA devices:
  - Disk drives, tape drives
- Remember (1st slide)
  - Keyboard data rate → 0.01 KB/s (1 byte every 100 ms)
  - Disk drive data rate → 2,000 KB/s (1 byte every 0.5 µs)

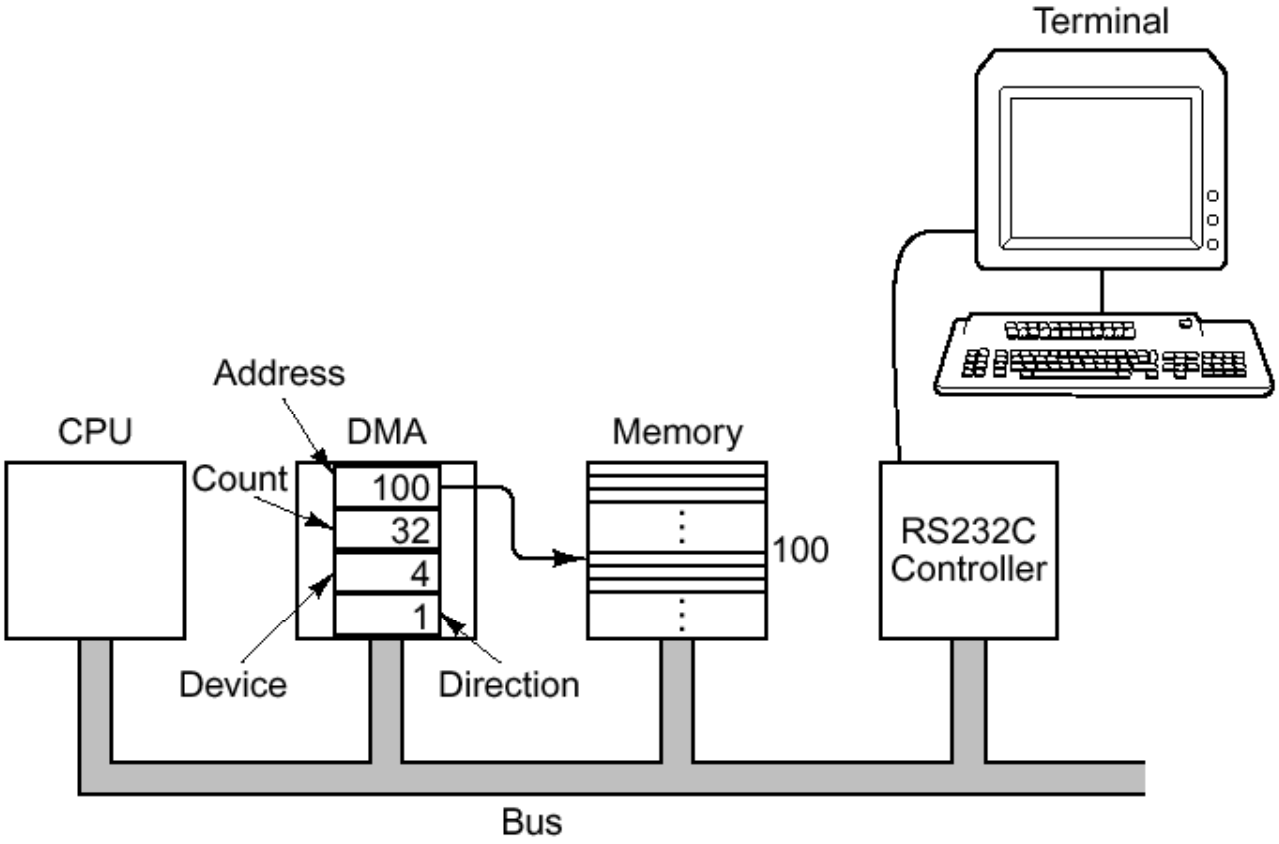
Transfer rate is too high to be controlled by software executing on the CPU

How

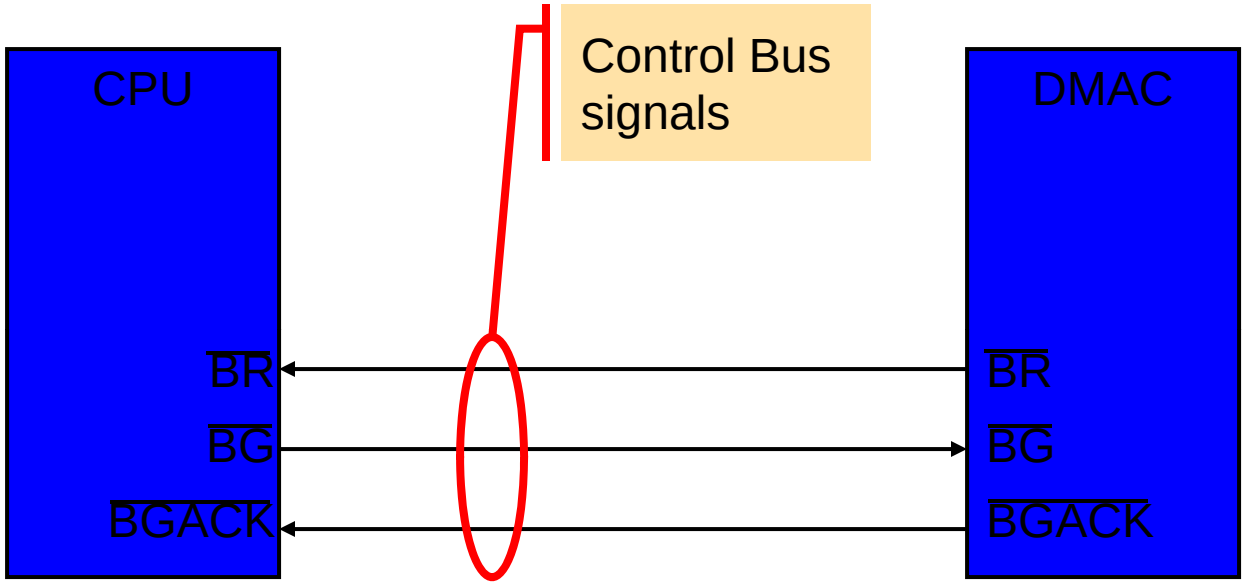
- The CPU “prepares” the DMA operation by transferring information to a DMA controller (DMAC):
  - Location of the data on the device
  - Location of the data in memory
  - Size of the block to transfer
  - Direction of the transfer
  - Mode of transfer (burst, cycle steal)
- When the device is ready to transfer data, the DMAC takes control of the system buses (next few slides)

DMA I/O

- Example:



Taking Control



$\overline{\text{BR}}$ = Bus request (DMAC: May I take control of the system buses?)

$\overline{\text{BG}}$ = Bus grant (CPU: Yes, here you go.)

$\overline{\text{BGACK}}$ = BG acknowledge (DMAC: Thanks, I've got control.)

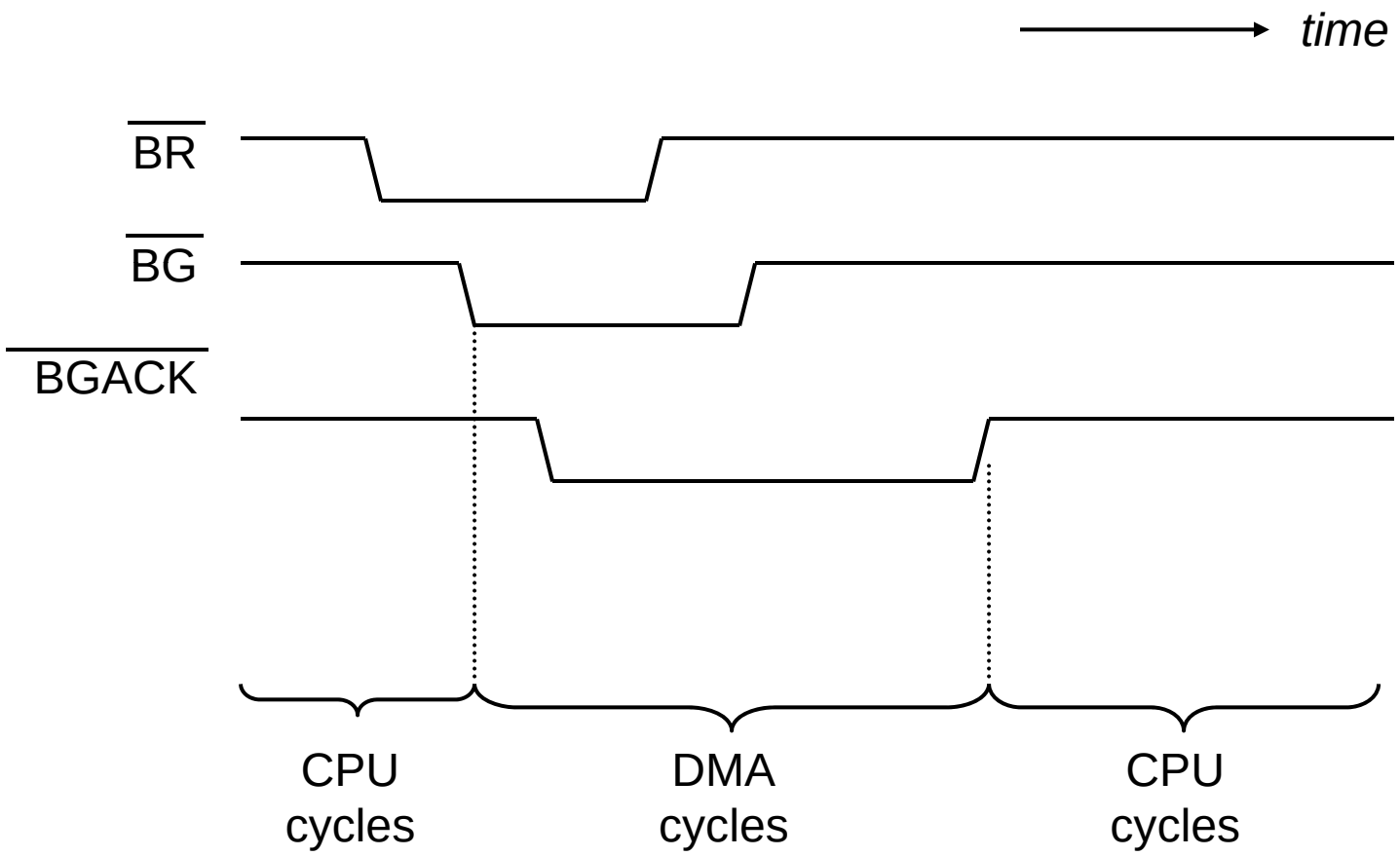
Taking Control

- DMAC issues a $\overline{\text{BR}}$ (“bus request”) signal
- CPU halts (perhaps in the middle of an instruction!) and issues a $\overline{\text{BG}}$ (“bus grant”) signal
- DMAC issues $\overline{\text{BGACK}}$ (“bus grant acknowledge”) and releases $\overline{\text{BR}}$
- DMAC has control of the system buses
- DMAC “acts like the CPU” and generates the bus signals (e.g., address, control) for one transfer to take place
- Then…

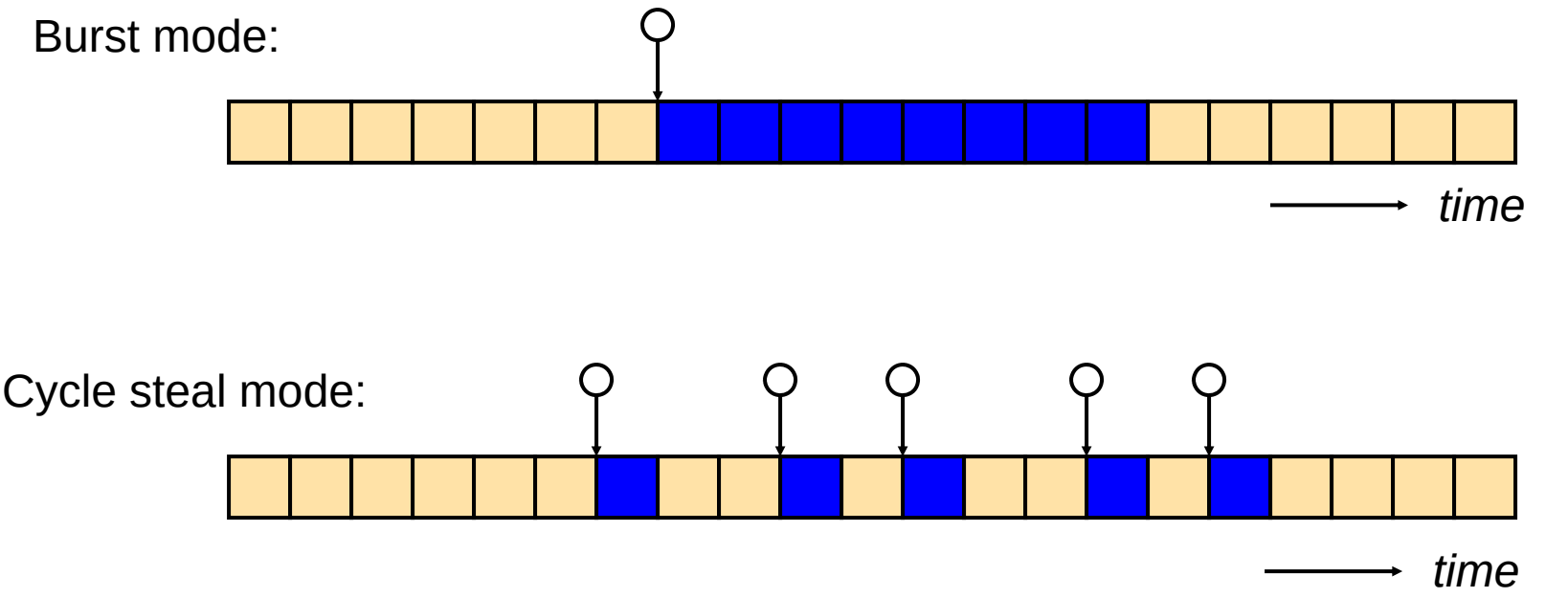
DMA Transfers

- **Burst mode**
  - This transfer is repeated until complete
  - $\overline{\text{DMAC}}$ relinquishes control of the system buses by releasing $\overline{\text{BGACK}}$
- **Cycle steal mode**
  - $\overline{\text{DMAC}}$ relinquishes control of the system buses by releasing $\overline{\text{BGACK}}$
  - A $\overline{\text{BR}}$-$\overline{\text{BG}}$-$\overline{\text{BGACK}}$ sequence occurs for every transfer, until the block is completely transferred
- DMAC interrupts the CPU when the transfer is complete
  - This is an example of a “completion signal” interrupt

$\overline{\text{BR}}$-$\overline{\text{BG}}$-$\overline{\text{BGACK}}$ Timing



Burst Mode vs. Cycle Steal Mode

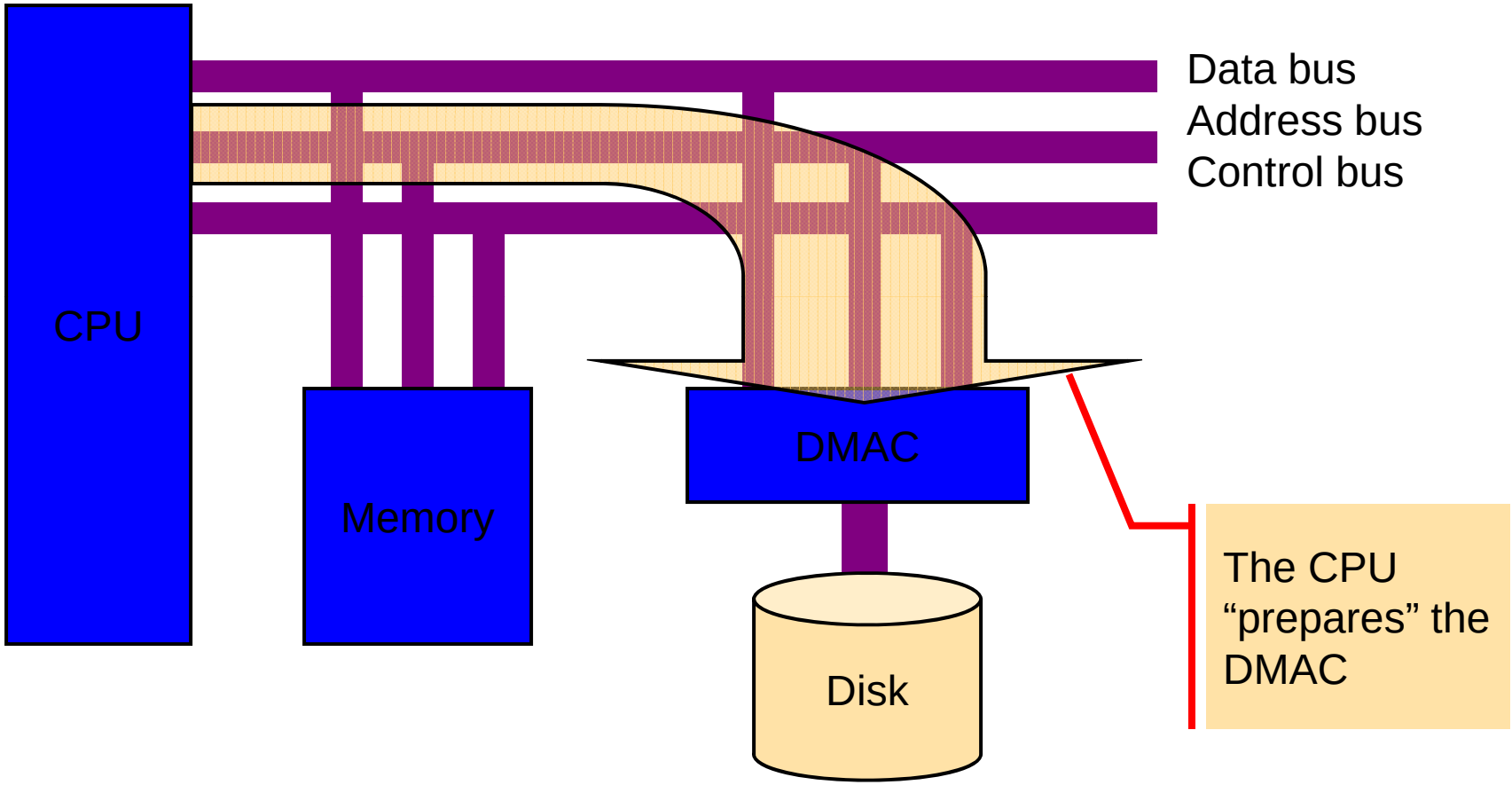


Types of I/O

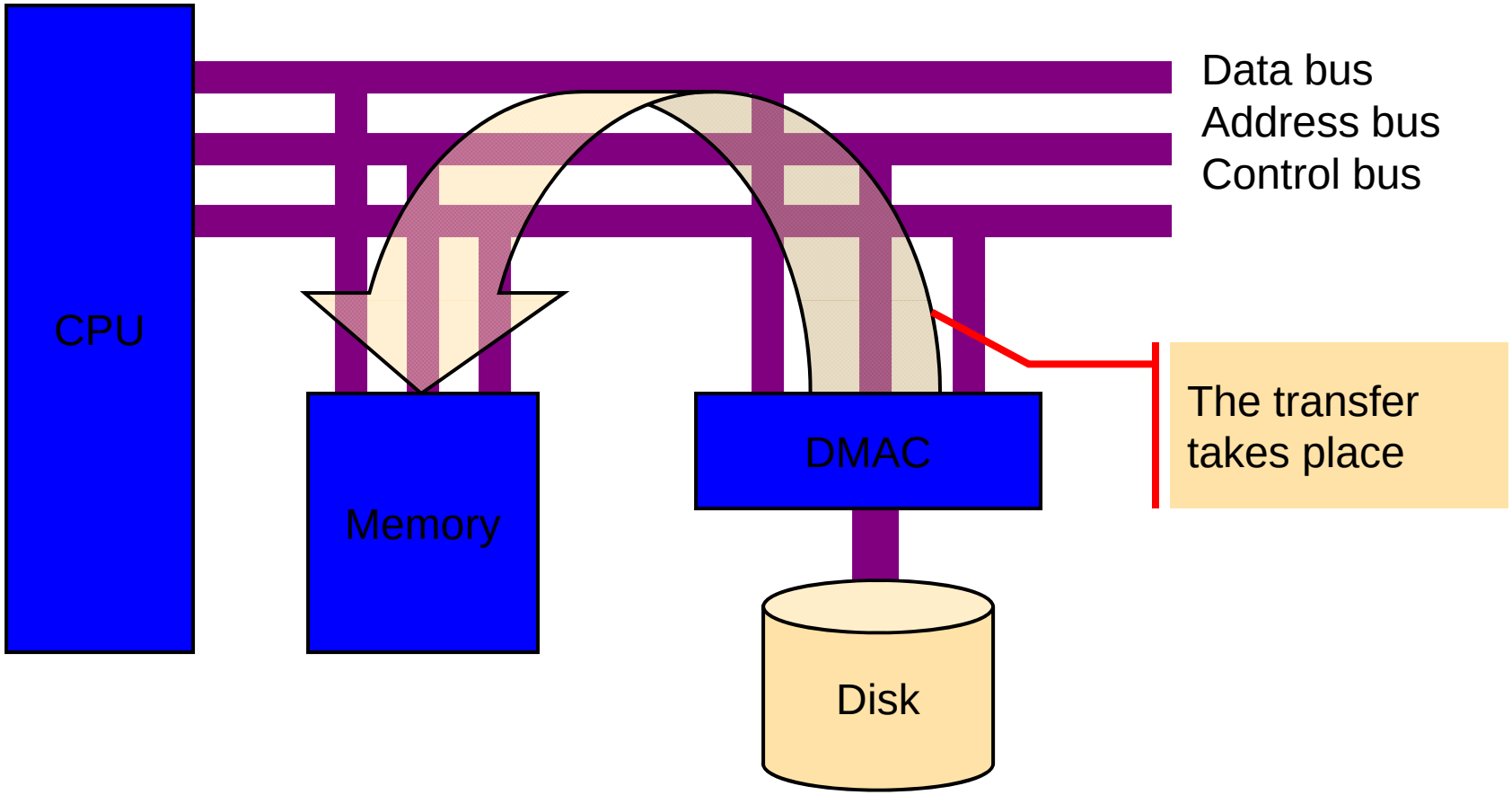
- Programmed I/O
- Interrupt-driven I/O
- Direct memory access (DMA)

DMA includes all three types of I/O.
Let's see…

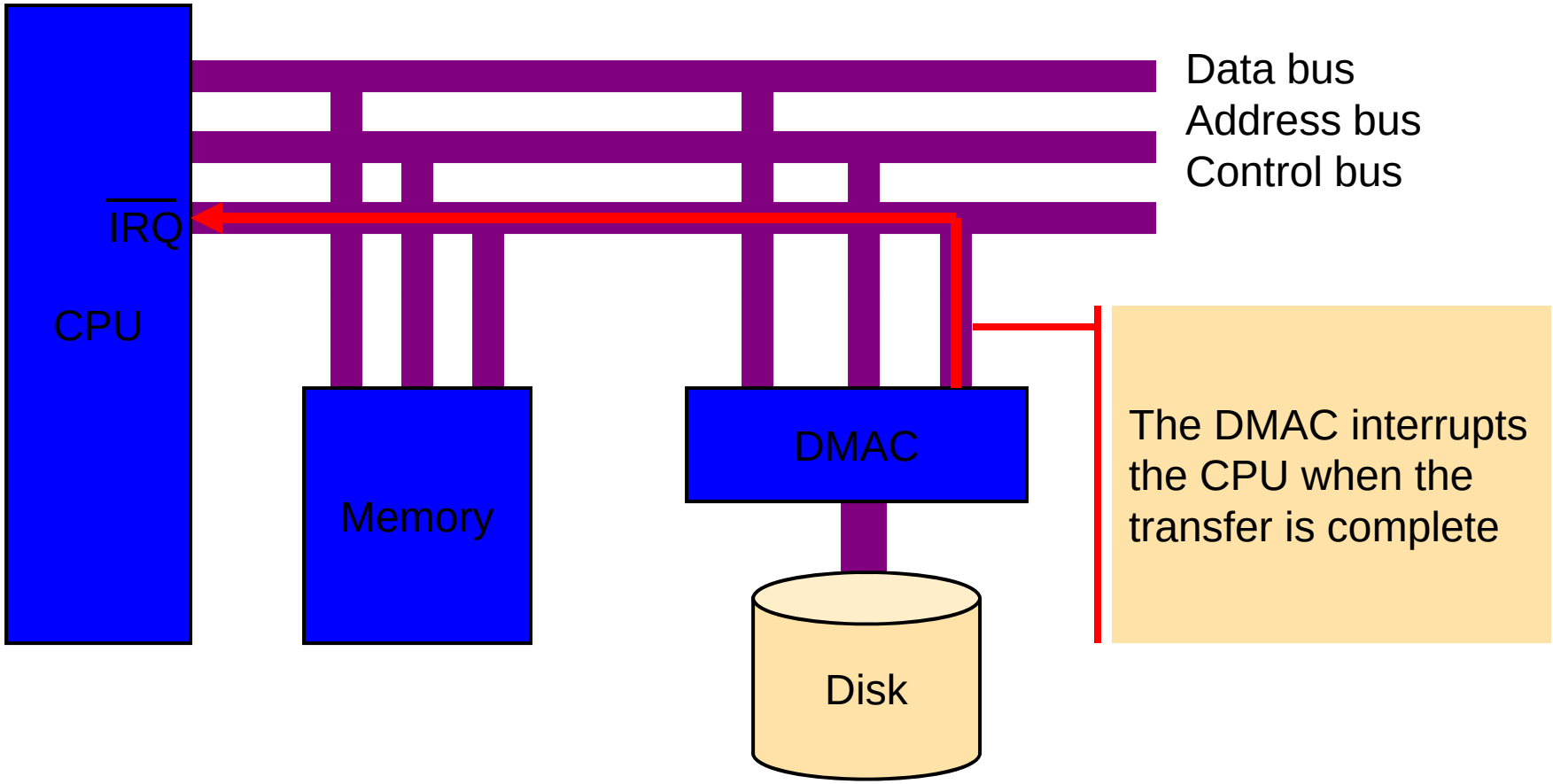
Program-Controlled I/O (in DMA)



DMA


CPU
Memory
DMAC
Disk
Data bus
Address bus
Control bus
The transfer takes place

Interrupt-driven I/O (in DMA)



Input/Output (I/O) Interfaces (1)

- Circuitry to transfer data between computer bus and an I/O device. The interface connects to the bus signals (address, data, control) on one side, with the data path and associated control for the I/O device on the other side.
- Types of I/O interfaces
  - Parallel Interface
  - Serial Interface

Input/Output (I/O) Interfaces (2)

- Parallel interface is suitable for devices that are physically close to the computer because the problem of timing skew limits the data rates for longer distances.
- On the other hand, serial transmission is convenient for connecting devices that are physically far away from the computer because it requires fewer wires.

Input/Output (I/O) Interfaces (3)

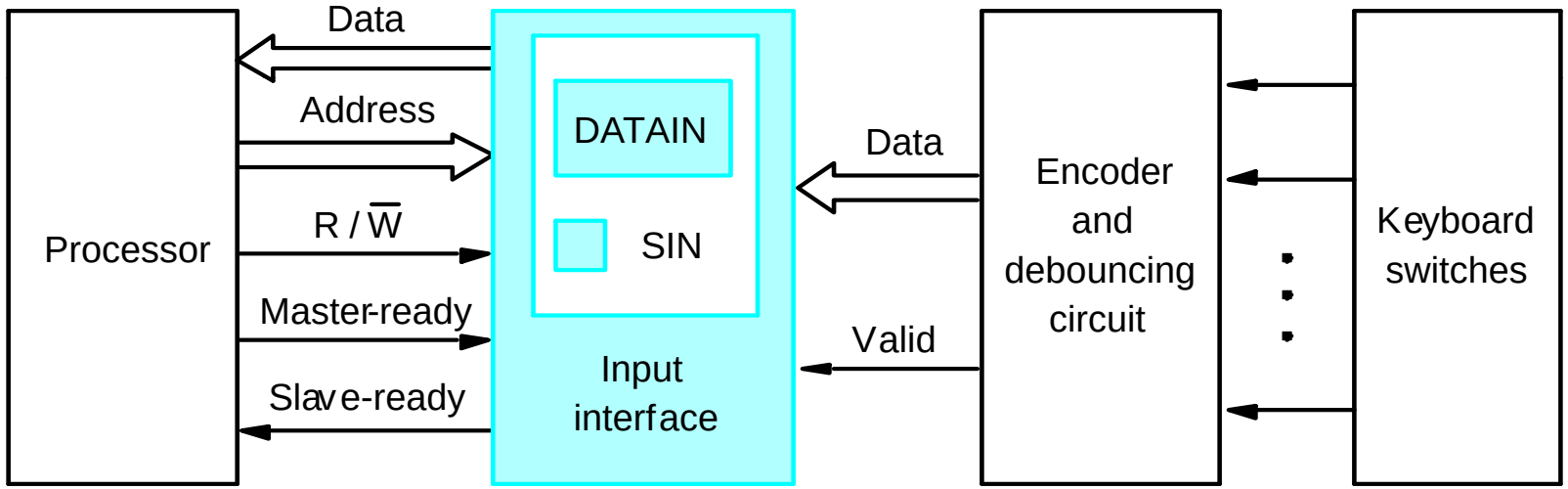
- I/O Interface functions :
  - (1) Provides a storage
  - (2) Contains status flags to be accessed by the processor
  - (3) Decodes addresses sent by the processor
  - (4) Generates timing signals for bus control
  - (5) Performs format conversion for data transfer between bus and I/O device

Parallel Interface (1)

- A parallel interface transfers data in the form of one or more bytes simultaneously to or from the device.
- The interface allows data input (such as keyboard) or output (such as printer) connected to the CPU through the bus.

Parallel Interface (2)

- Parallel Input (Example : Keyboard input)



Parallel input from keyboard to processor

Parallel Interface (3)

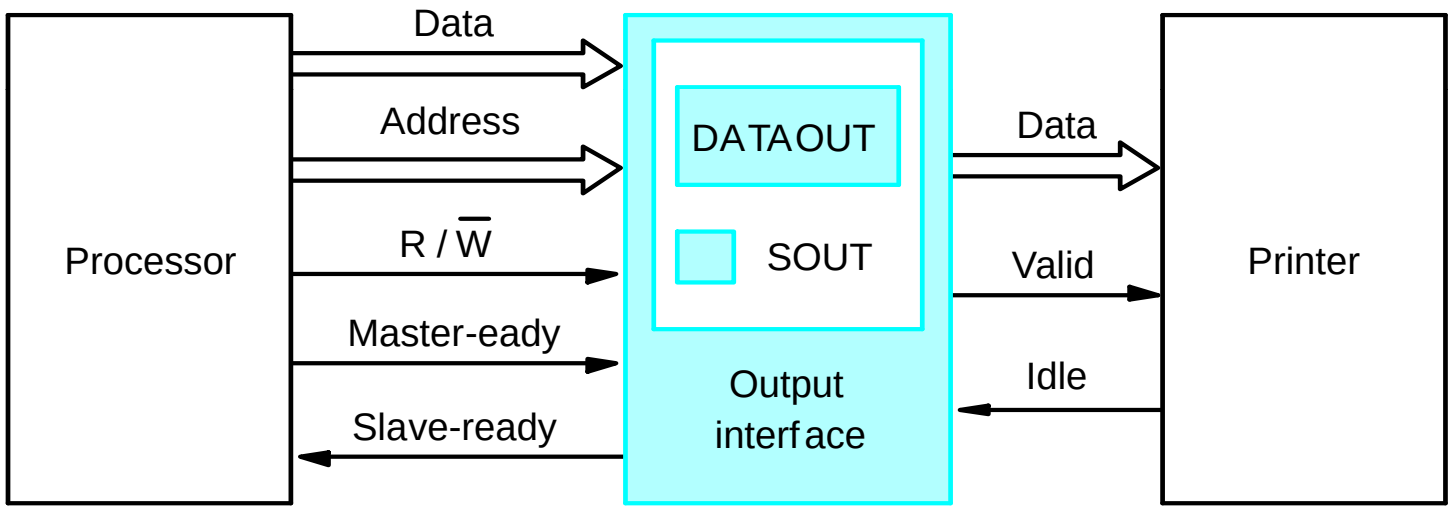
- (1) When a key is pressed, an electrical signal is detected by an encoder circuit that generates the ASCII code for the corresponding character and one control signal, called Valid, which indicates that a key is being pressed.
- (2) The input interface contains a data register DATAIN and a status flag SIN. The ASCII code is loaded into DATAIN and SIN is set to 1.
  - The output lines of DATAIN and SIN are connected to the data bus.

Parallel Interface (4)

- (3) The processor reads the contents of DATAIN and SIN is cleared to 0.
- (4) The I/O interface is connected to the asynchronous bus on which transfers are controlled using handshake signals (Master-ready and Slave-ready).

Parallel Interface (5)

- Parallel Output (Example : Printer output)



Parallel output from processor to printer

Parallel Interface (6)

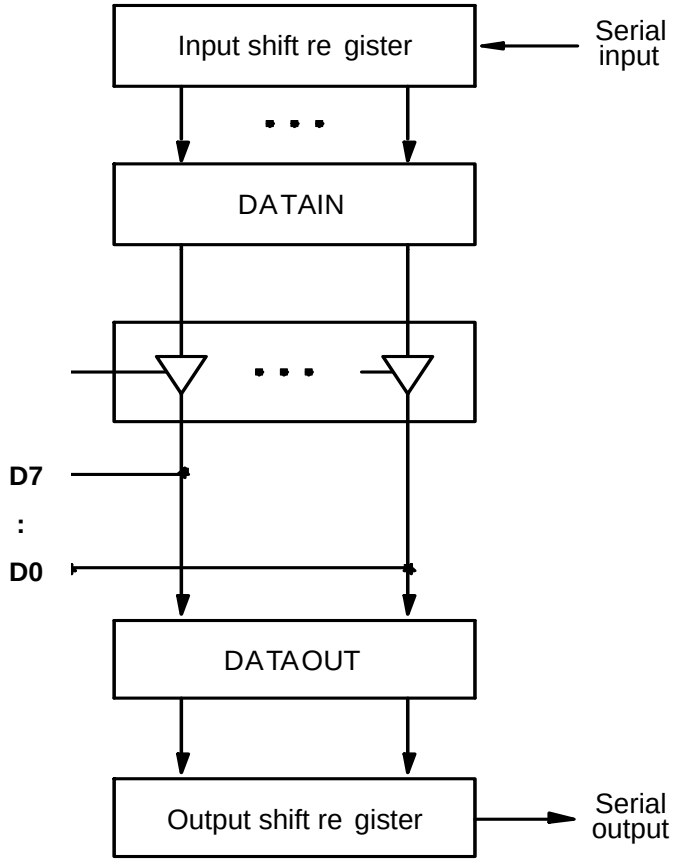
- (1) When the printer is ready to accept a character, it asserts the Idle signal.
- (2) The I/O interface can then place a new character on the data lines and activate the Valid signal.
- (3) The printer starts printing the character and negates the Idle signal, that causes the interface to deactivate the Valid signal.

Parallel Interface (7)

- (4) The interface contains a DATAOUT register and a status flag, SOUT. The SOUT flag is set to 1 when the printer is ready to accept another character, it is cleared to 0 when a new character is loaded into DATAOUT by the processor.

Serial Interface (1)

- Serial interface circuit communicates in a serial fashion on the device side and in a parallel fashion on the bus side.
- The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.



Serial Interface (2)

- The input shift register accepts bit-serial input from the I/O device. When all bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register.
- Output data in the DATAOUT register are loaded into the output shift register, from which the bits are shifted out and sent to the I/O device.