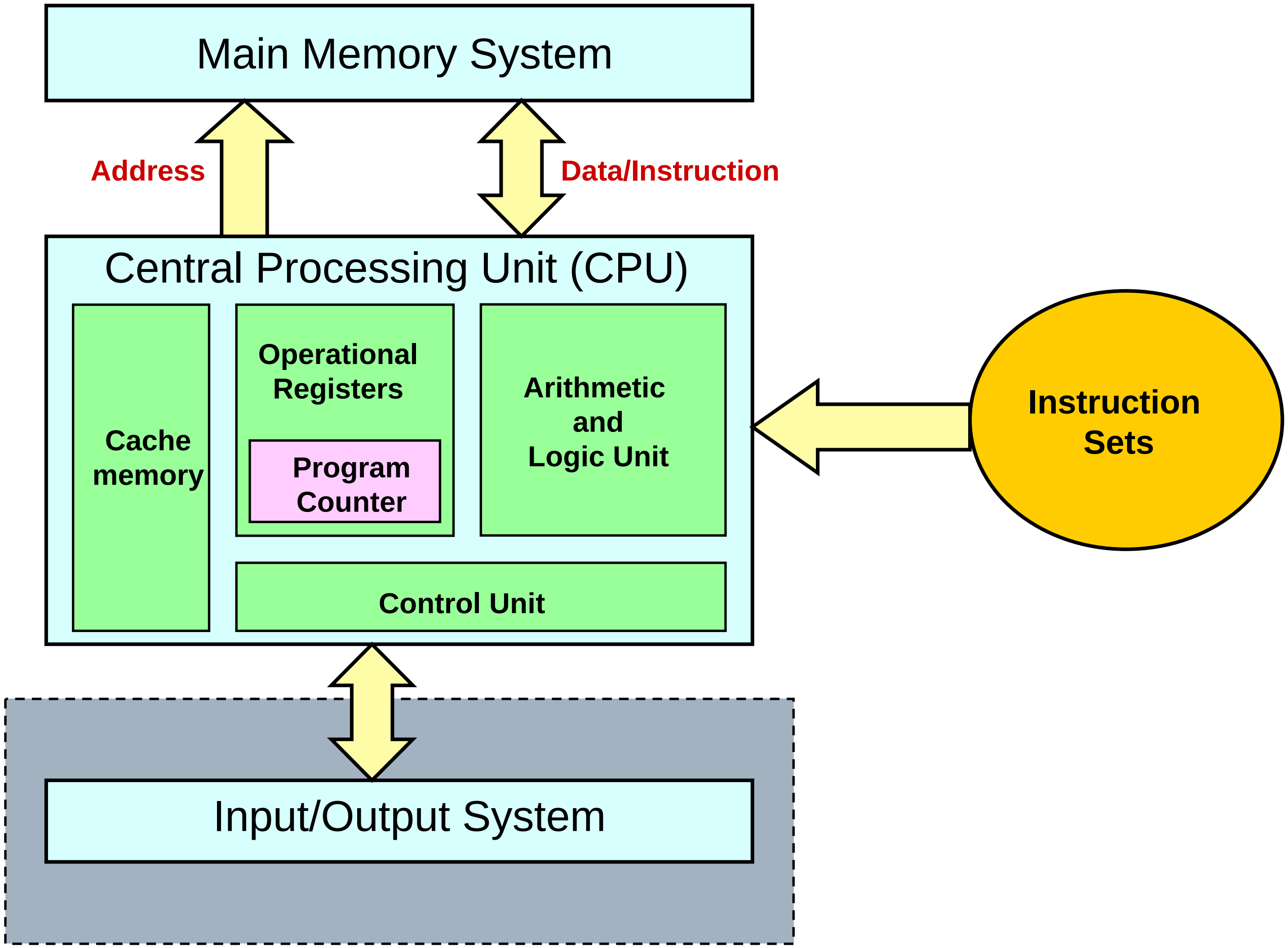
Chapter 5
Input/Output Organization

Jin-Fu Li
Department of Electrical Engineering
National Central University
Jungli, Taiwan

Outline

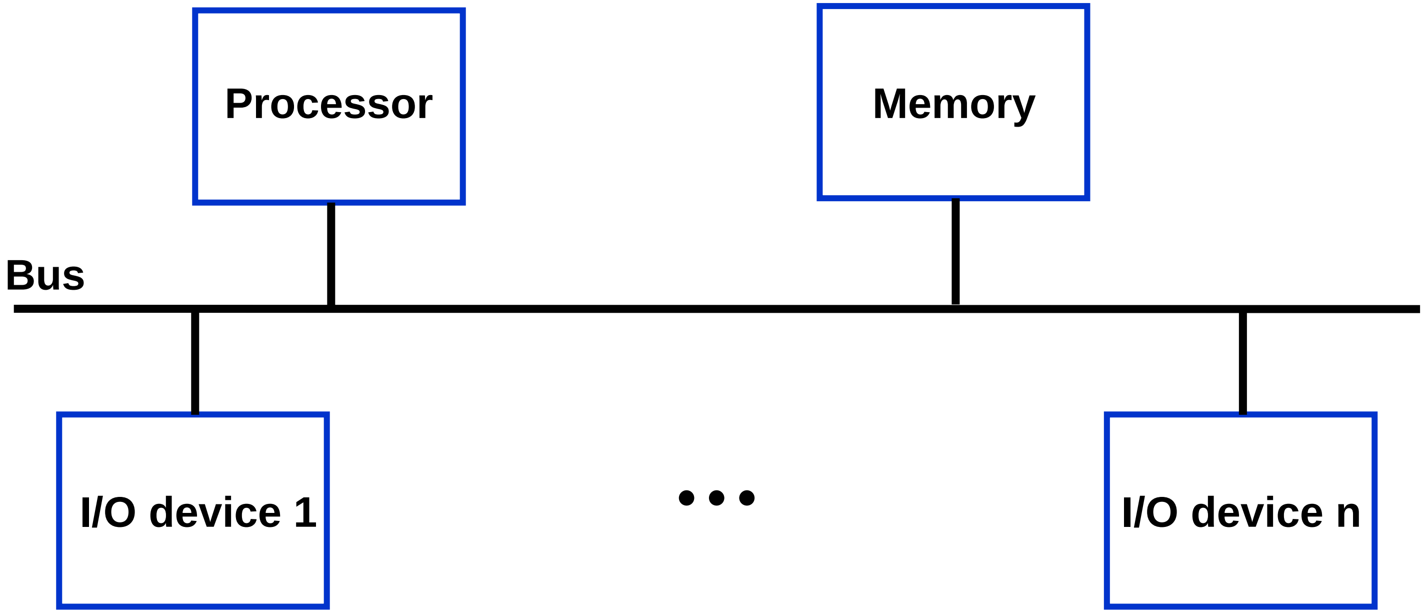
- Accessing I/O Devices
- Interrupts
- Direct Memory Access
- Buses
- Interface Circuits
- Standard I/O Interfaces

Content Coverage



Accessing I/O Devices

- Single-bus structure
  - The bus enables all the devices connected to it to exchange information
  - Typically, the bus consists of three sets of lines used to carry address, data, and control signals
  - Each I/O device is assigned a unique set of addresses



I/O Mapping

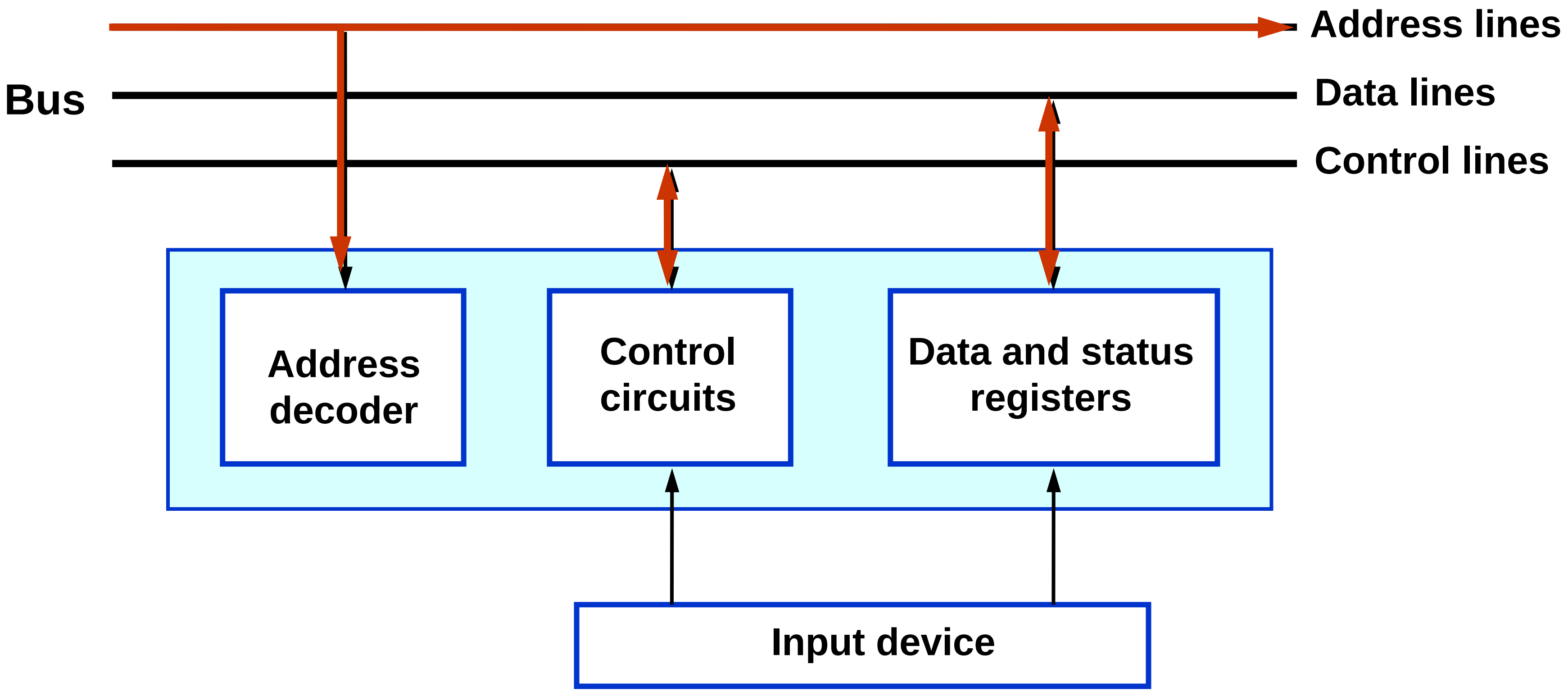
- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for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

Memory-Mapped I/O

- When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O
- With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device
- Most computer systems use memory-mapped I/O.
- Some processors have special IN and OUT instructions to perform I/O transfers
  - When building a computer system based on these processors, the designer has the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space

I/O Interface for an Input Device

- The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's *interface circuit*

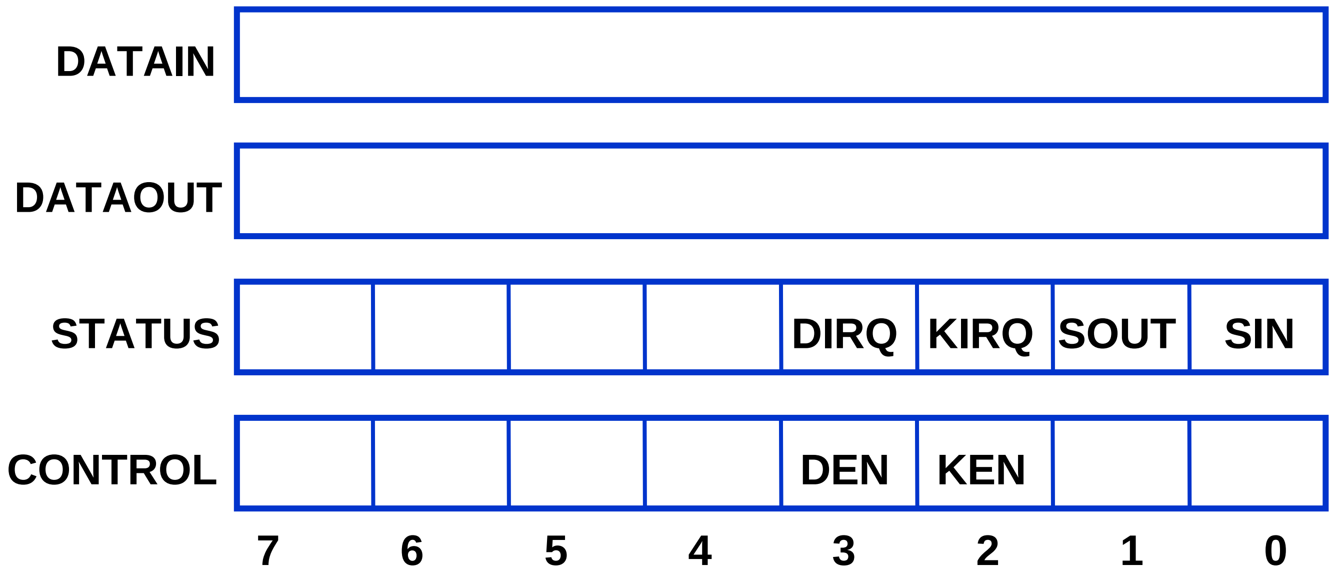


I/O Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

Program-Controlled I/O

- Consider a simple example of I/O operations involving a keyboard and a display device in a computer system. The four registers shown below are used in the data transfer operations
  - The two flags KIRQ and DIRQ in STATUS register are used in conjunction with interrupts



An Example

- A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display

```
         Move       #LINE, R0       Initialize memory pointer
WAITK    TestBit    #0,STATUS       Test SIN
         Branch=0   WAITK           Wait for character to be entered
         Move       DATAIN,R1       Read character
WAITD    TestBit    #1,STATUS       Test SOUT
         Branch=0   WAITD           Wait for display to become ready
         Move       R1,DATAOUT      Send character to display
         Move       R1,(R0)+        Store character and advance pointer
         Compare    #$0D,R1         Check if Carriage Return
         Branch≠0   WAITK           If not, get another character
         Move       #$0A,DATAOUT    Otherwise, send Line Feed
         Call       PROCESS         Call a subroutine to process the
                                    input line
```

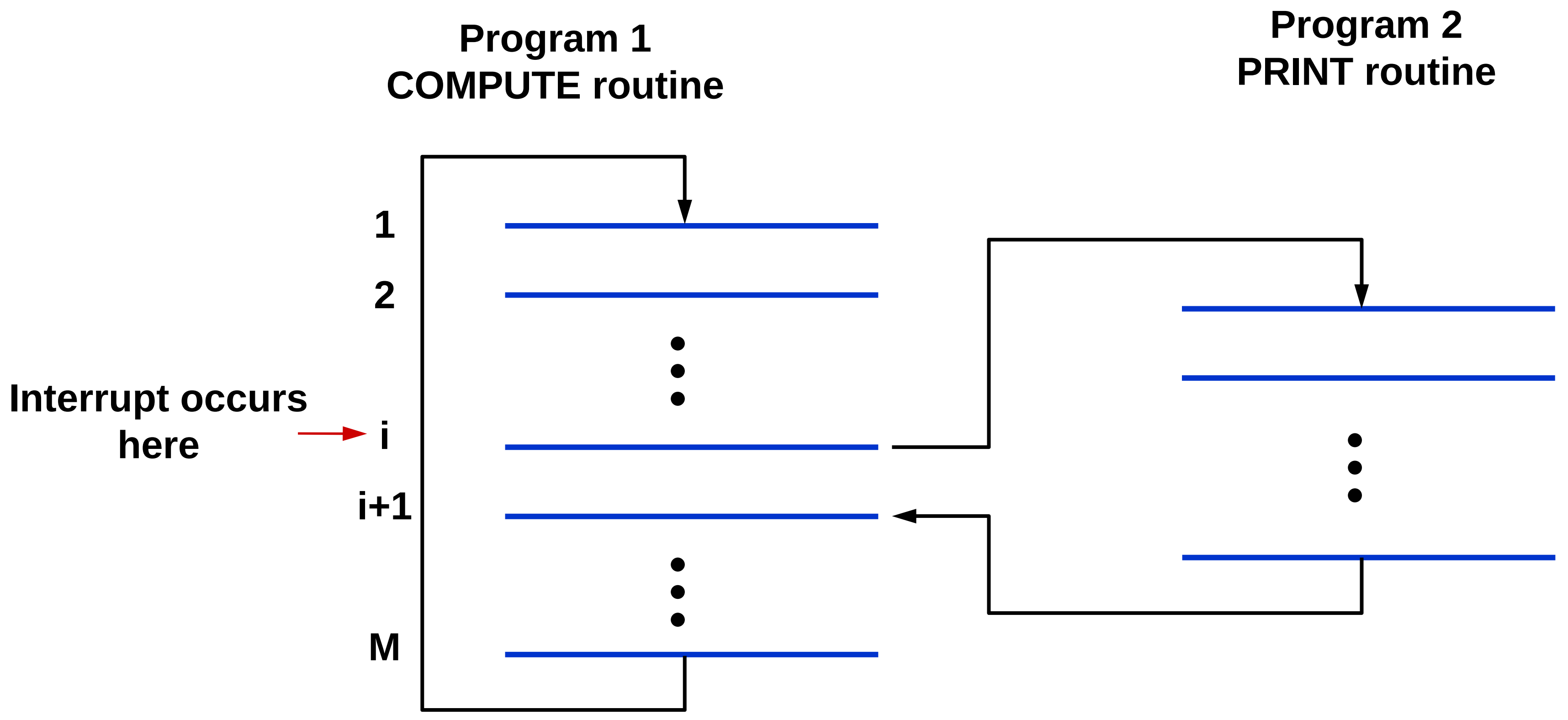
Program-Controlled I/O

- The example described above illustrates program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor *polls* the devices
- There are two other commonly used mechanisms for implementing I/O operations: *interrupts* and *direct memory access*
  - Interrupts: synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation
  - Direct memory access: it involves having the device interface transfer data directly to or from the memory

Interrupts

- To avoid the processor being not performing any useful computation, a hardware signal called an *interrupt* to the processor can do it. At least one of the bus control lines, called an *interrupt-request* line, is usually dedicated for this purpose
- An *interrupt-service routine* usually is needed and is executed when an interrupt request is issued
- On the other hand, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. An *interrupt-acknowledge* signal serves this function

Example


Program 1
COMPUTE routine
Program 2
PRINT routine
1
2
Interrupt occurs here
i
i+1
M

Interrupt-Service Routine & Subroutine

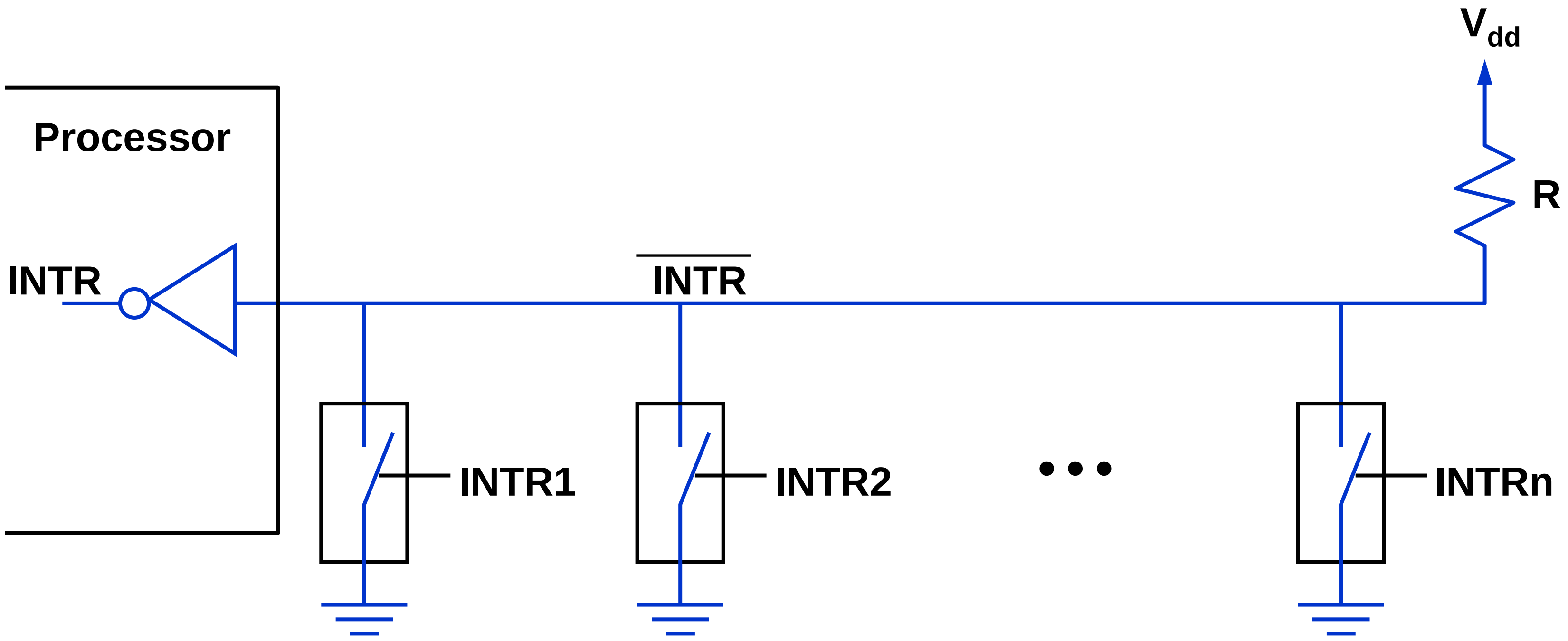
- Treatment of an interrupt-service routine is very similar to that of a subroutine
- An important departure from the similarity should be noted
  - A subroutine performs a function required by the program from which it is called.
  - The interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received. In fact, the two programs often belong to different users
- Before executing the interrupt-service routine, any information that may be altered during the execution of that routine must be saved. This information must be restored before the interrupted program is resumed

Interrupt Latency

- The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine
- Saving registers also increases the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. The delay is called *interrupt latency*
- Typically, the processor saves only the contents of the program counter and the processor status register. Any additional information that needs to be saved must be saved by program instruction at the beginning of the interrupt-service routine and restored at the end of the routine

Interrupt Hardware

- An equivalent circuit for an open-drain bus used to implement a common interrupt-request line



INTR=INTR1+INTR2+…+INTRn

Handling Multiple Devices

- Handling multiple devices gives rise to a number of questions:
  - How can the processor recognize the device requesting an interrupt?
  - Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
  - Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  - How should two or more simultaneous interrupt request be handled?
- The information needed to determine whether a device is requesting an interrupt is available in its status register
  - When a device raises an interrupt request, it sets to 1 one of the bits in its status register, which we will call the IRQ bit

Identify the Interrupting Device

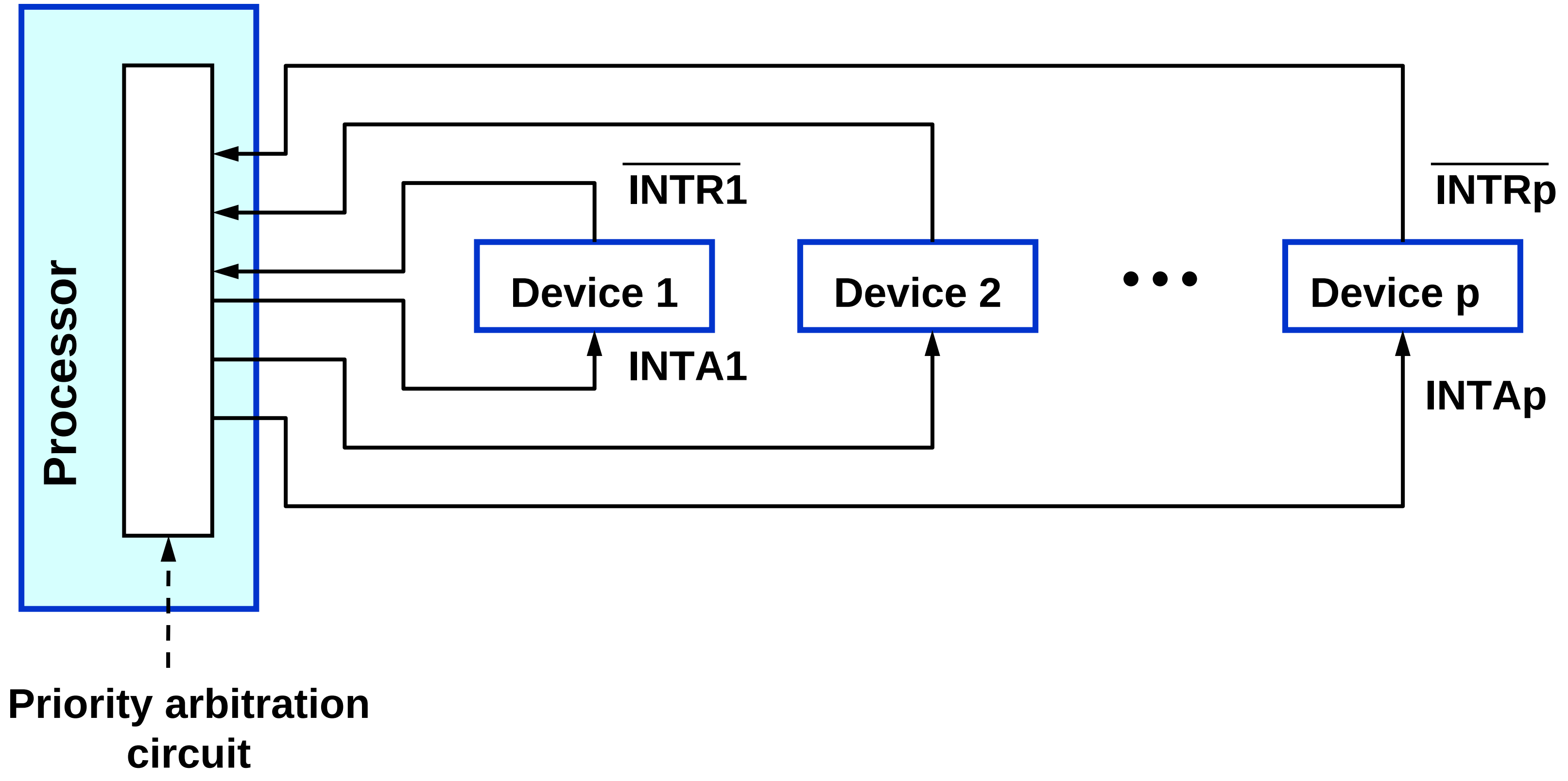
- The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the I/O devices connected to the bus
  - The polling scheme is easy to implement. Its main disadvantage is the time spent interrogating all the devices
- A device requesting an interrupt may identify itself directly to the processor. Then, the processor can immediately start executing the corresponding interrupt-service routine. This is called vectored interrupts
- An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device

Interrupt Priority

- The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the program status register (PS). These are privileged instructions, which can be executed only while the processor is running in the supervisor mode
- The processor is in the supervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application program
- An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privilege exception

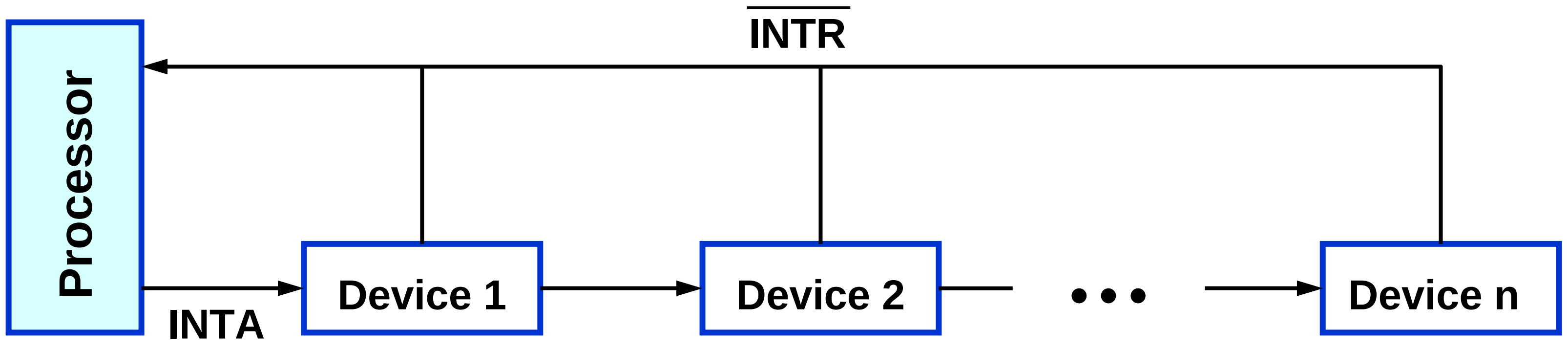
Implementation of Interrupt Priority

- An example of the implementation of a multiple-priority scheme


Processor
$\overline{\text{INTR1}}$
INTA1
Device 1
Device 2
• • •
Device p
$\overline{\text{INTRp}}$
INTAp
Priority arbitration circuit

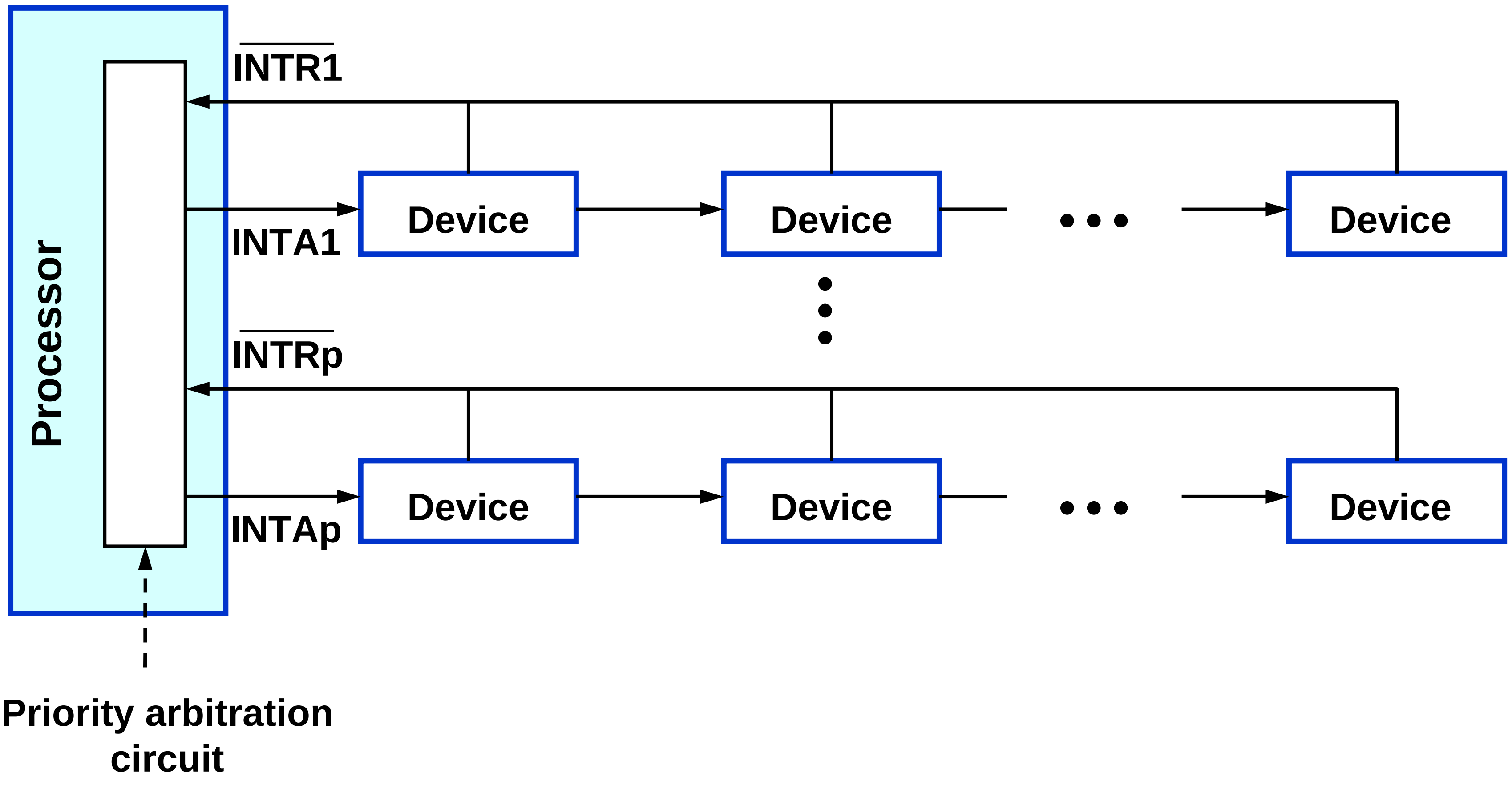
Simultaneous Requests

- Consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor must have some means of deciding which request to service first
- Interrupt priority scheme with daisy chain



Priority Group

- Combination of the interrupt priority scheme with daisy chain and with individual interrupt-request and interrupt-acknowledge lines



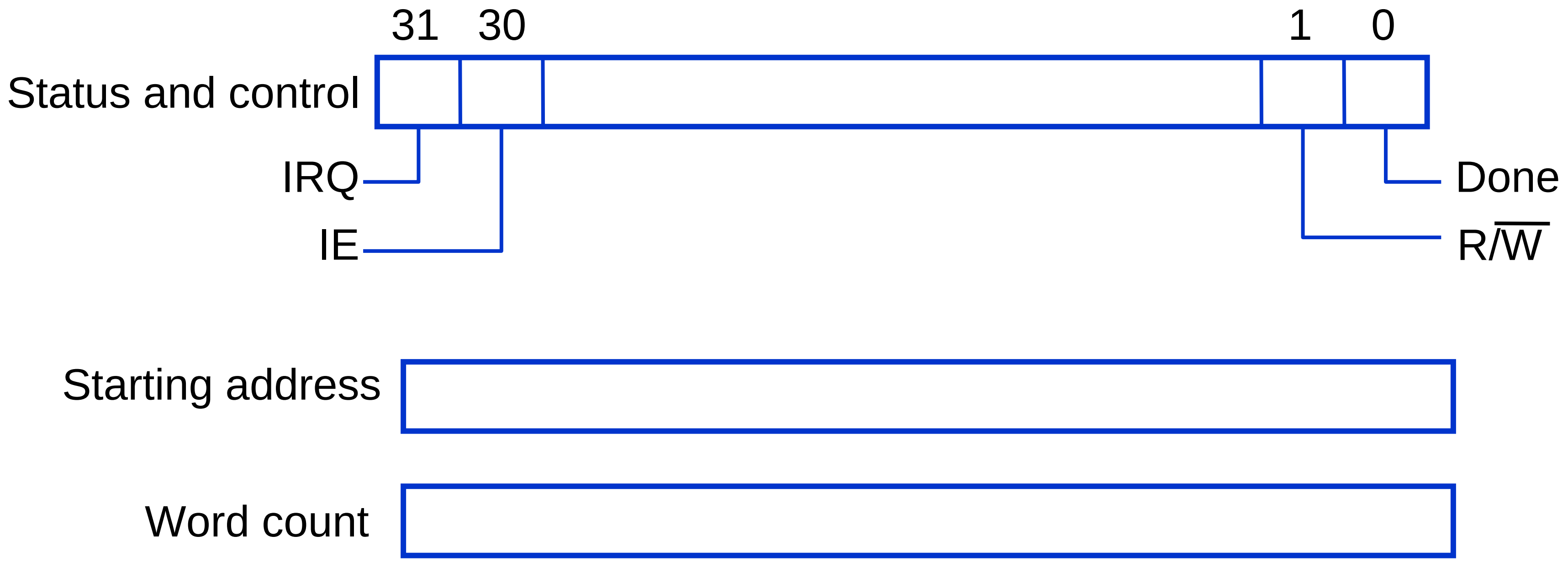
Processor
$\overline{\text{INTR1}}$
INTA1
$\overline{\text{INTRp}}$
INTAp
Device
Device
Device
Device
Device
Device
Priority arbitration circuit

Direct Memory Access

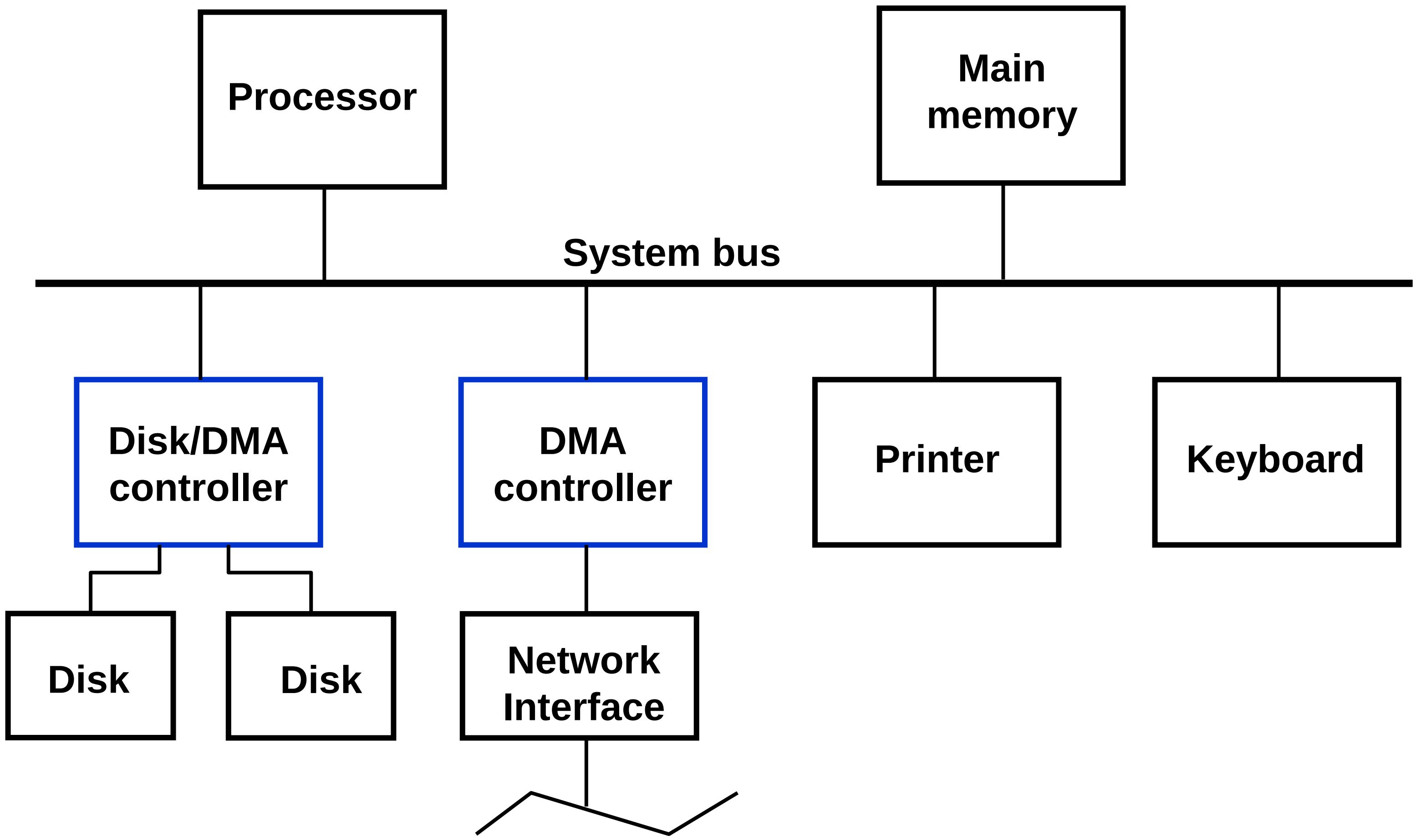
- To transfer large blocks of data at high speed, a special control unit may be provided between an external device and the main memory, without continuous intervention by the processor. This approach is called *direct memory access* (DMA)
- DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA controller.
- Since it has to transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of the number of transfers

DMA Controller

- Although a DMA controller can transfer data without intervention by the processor, its operation must be under the control of a program executed by the processor
- An example



DMA Controller in a Computer System



Memory Access Priority

- Memory accesses by the processor and the DMA controllers are interwoven. Request by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, etc.
- Since the processor originates most memory access cycles, the DMA controller can be said to “steal” memory cycles from the processor. Hence, this interweaving technique is usually called *cycle stealing*
- The DMA controller may transfer a block of data without interruption. This is called *block/burst* mode

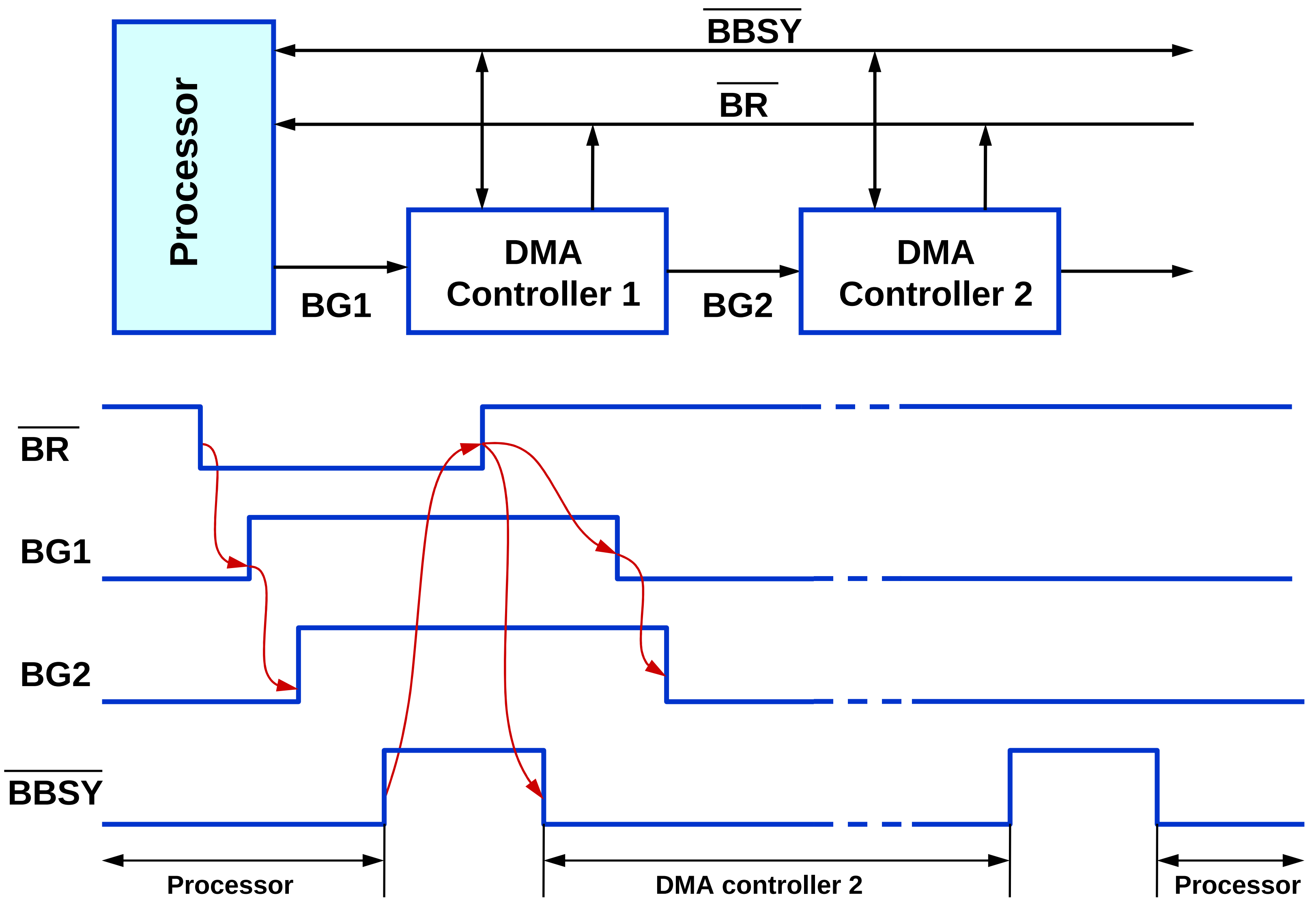
Bus Arbitration

- A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this problem, an arbitration procedure on bus is needed
- The device that is allowed to initiate data transfer on the bus at any given time is called the bus master. When the current master relinquishes control of the bus, another device can acquire this status
- Bus arbitration is the process by which the next device to become the bus master take into account the needs of various devices by establishing a priority system for gaining access to the bus

Bus Arbitration

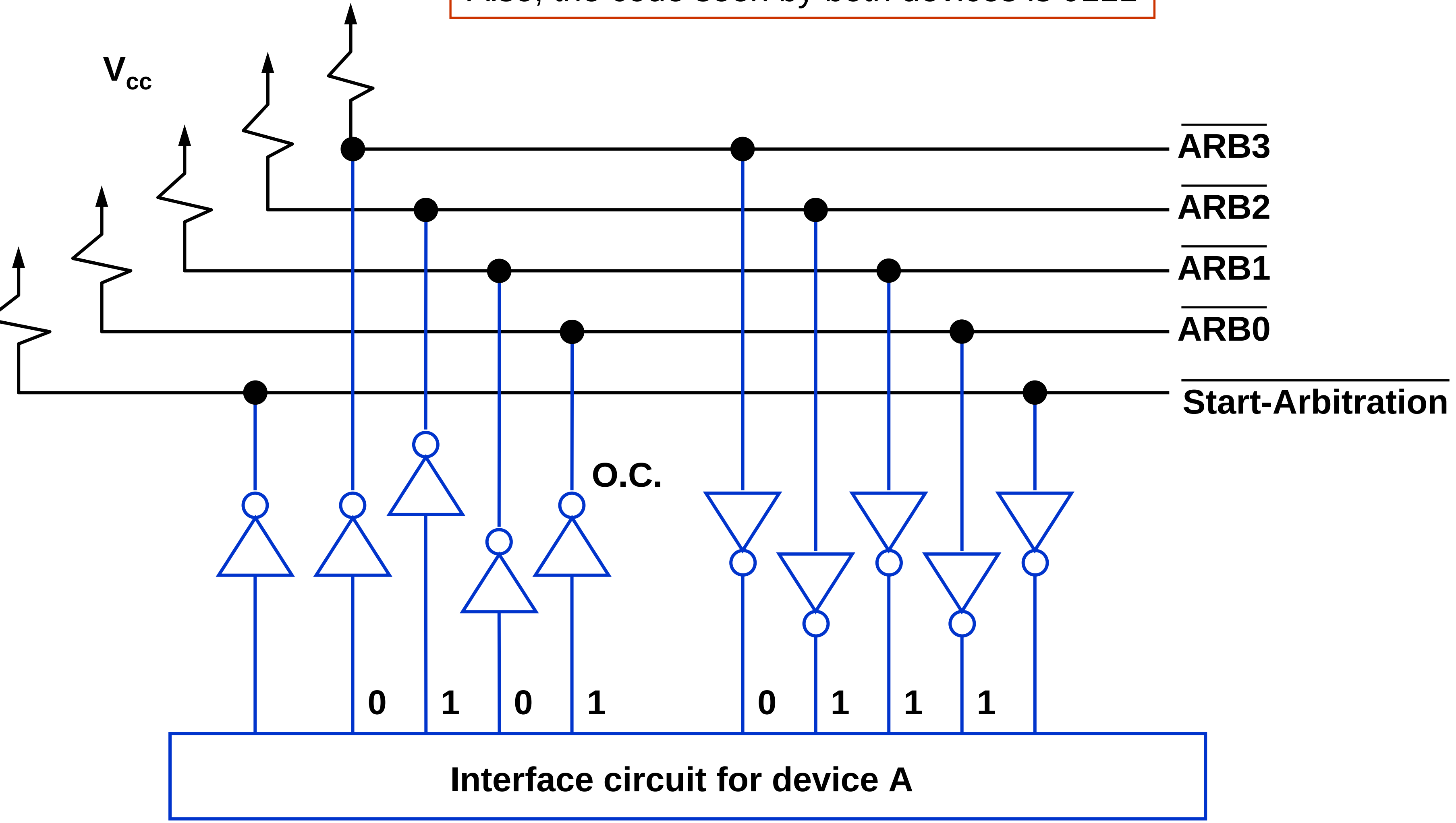
- There are two approaches to bus arbitration
  - Centralized and distributed
- In centralized arbitration, a single bus arbiter performs the required arbitration
- In distributed arbitration, all devices participate in the selection of the next bus master

Centralized Arbitration



Distributed Arbitration

Assume that IDs of A and B are 5 and 6.
Also, the code seen by both devices is 0111

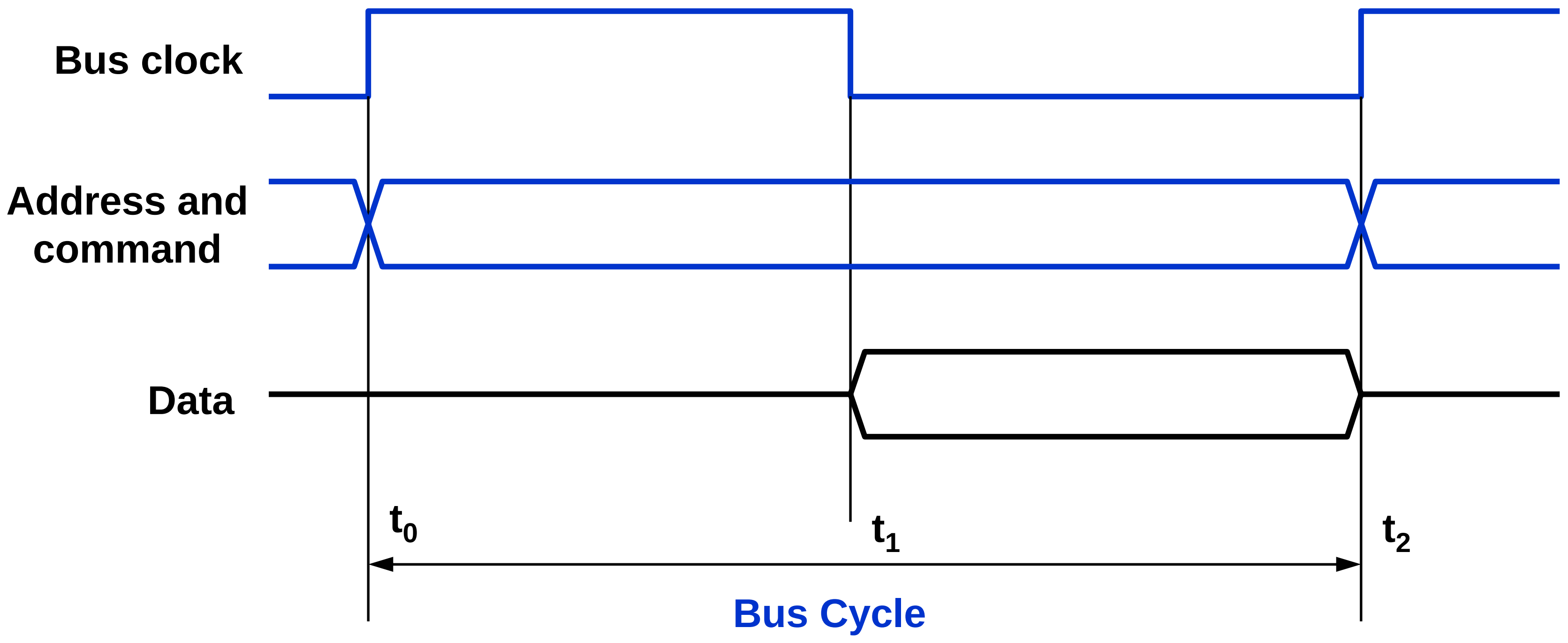


Buses

- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on
- In a synchronous bus, all devices derive timing information from a common clock line. Equal spaced pulses on this line define equal time intervals
- In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place

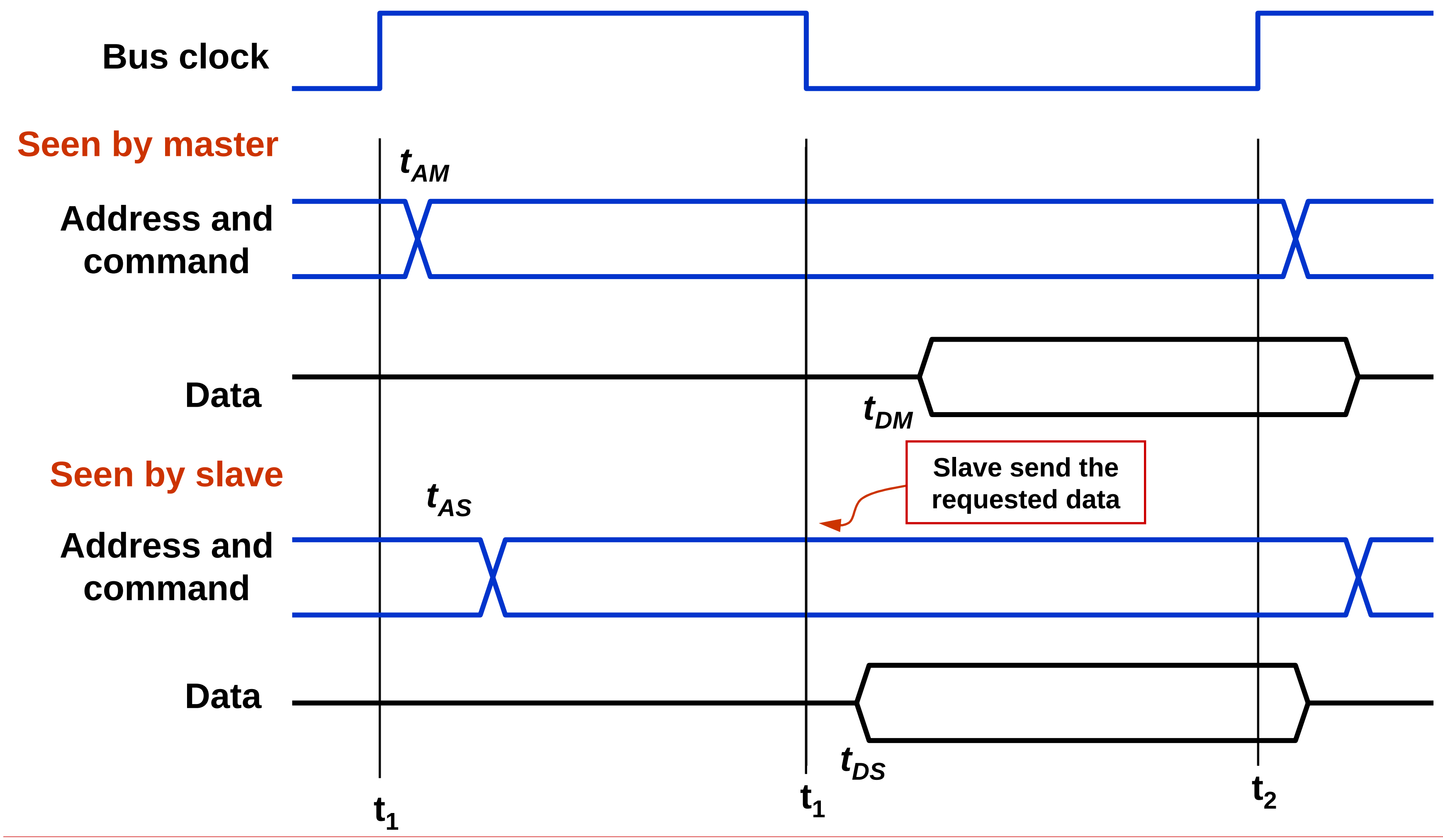
A Synchronous Bus Example

- Timing of an input transfer on a synchronous bus

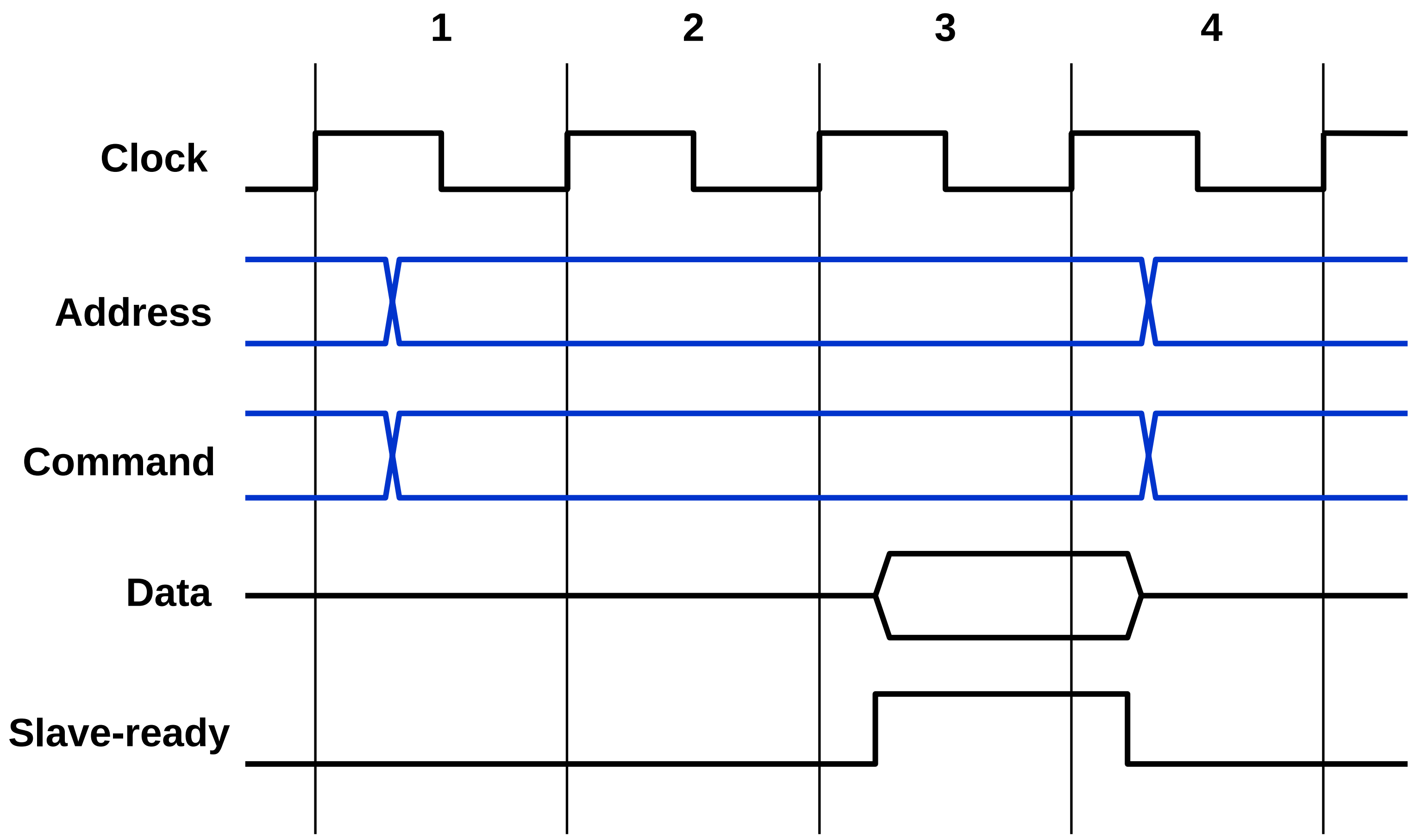


A Synchronous Bus Example

- Detail timing diagram

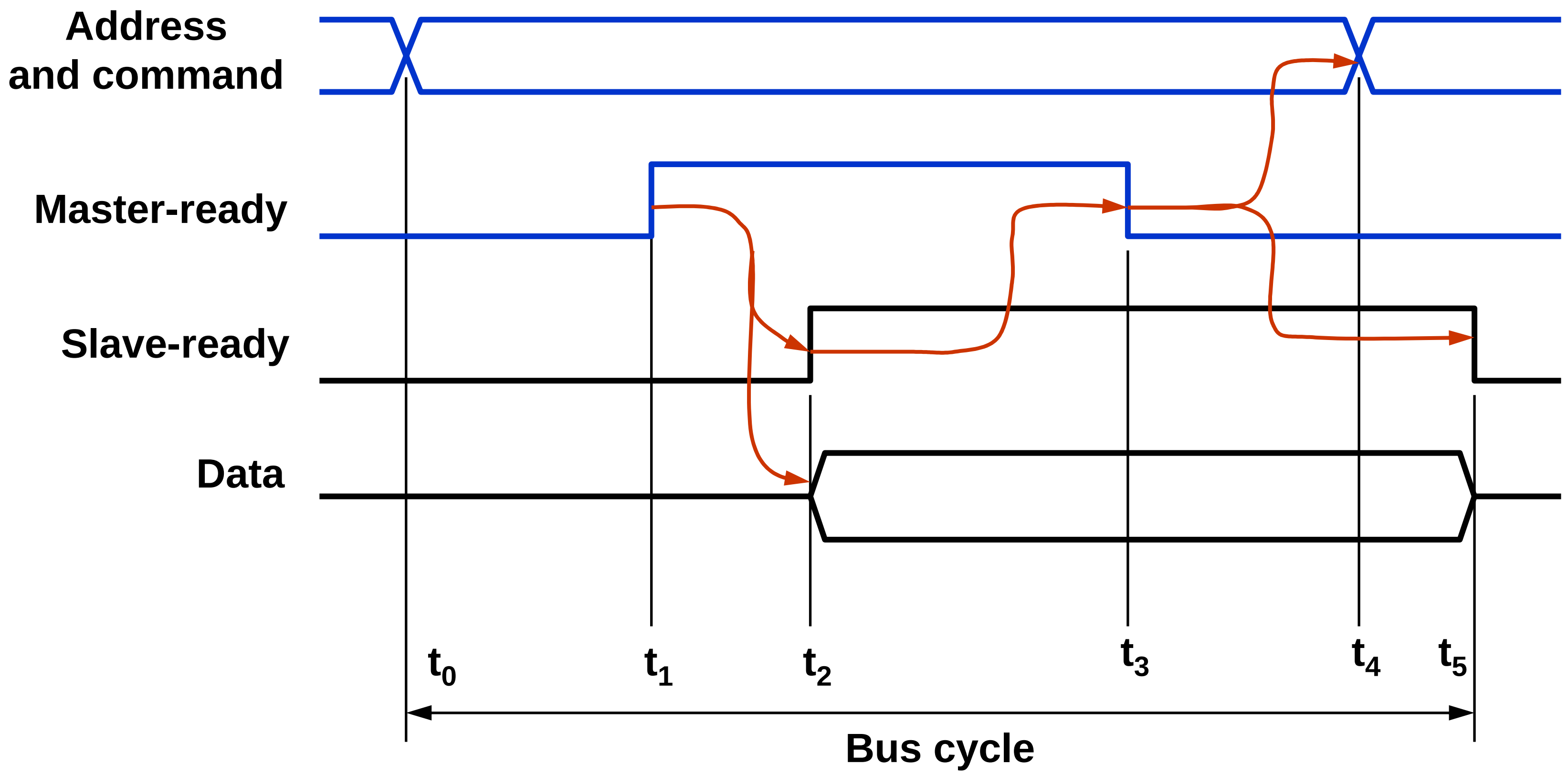


Input Transfer Using Multiple Clock Cycles



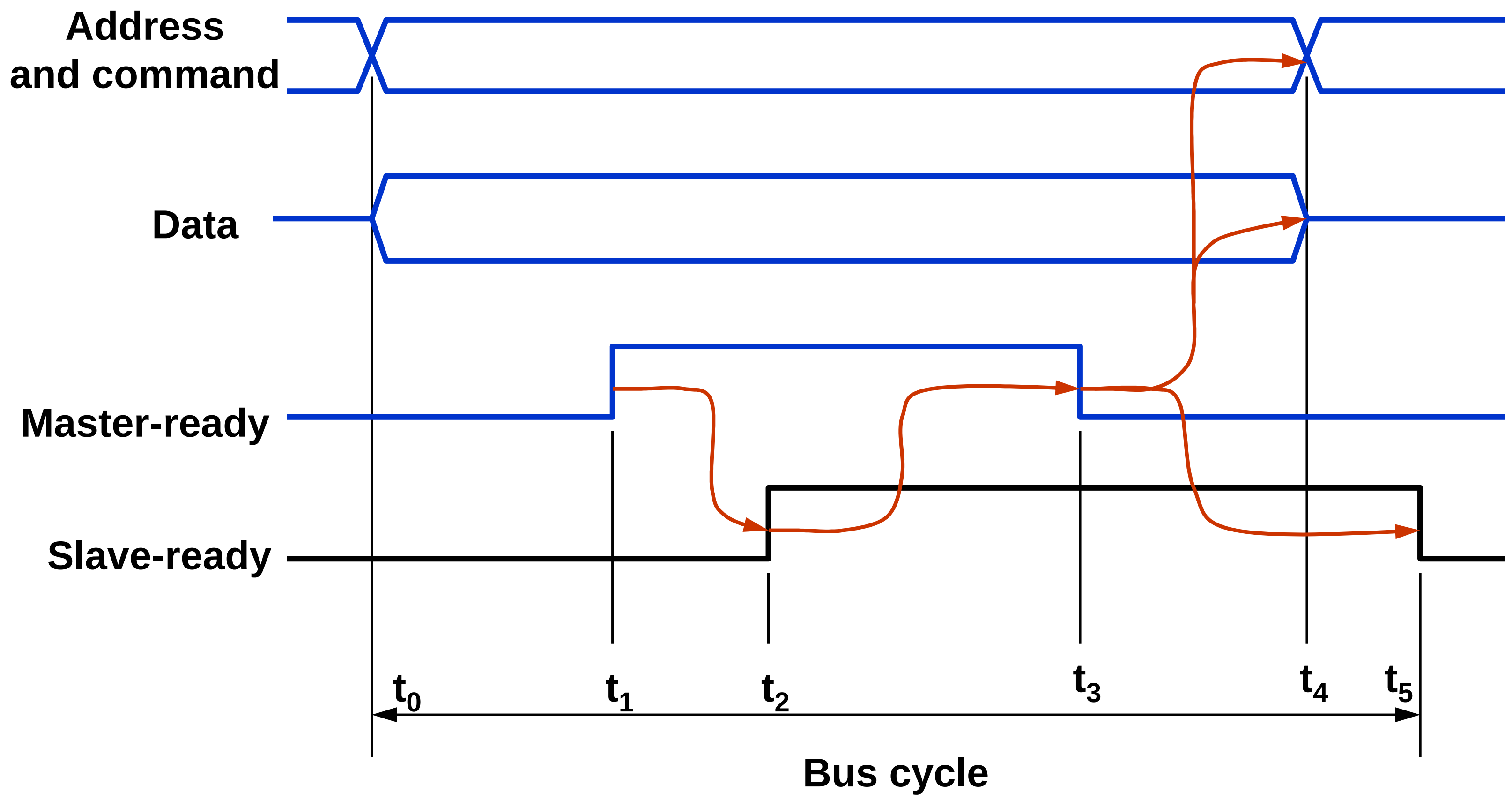
Asynchronous Bus

- An alternative scheme for controlling data transfers on the bus is based on the use of a handshake between the master and slave



Asynchronous Bus

- Handshake control of data transfer during an output operation

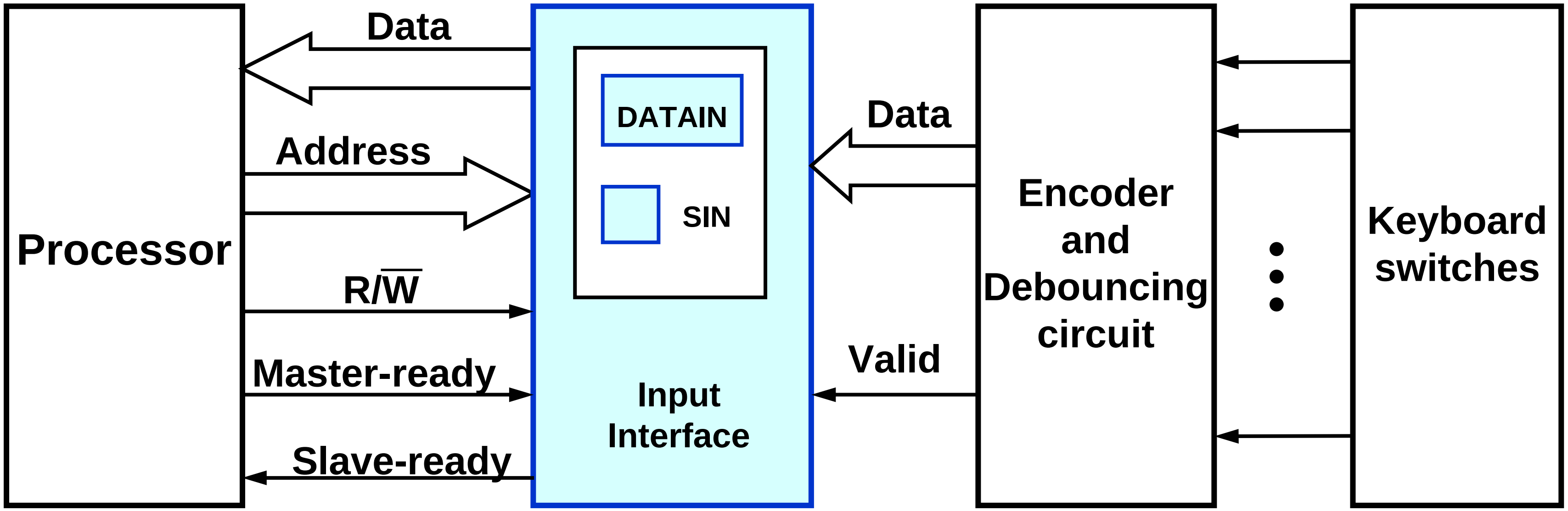


Discussion

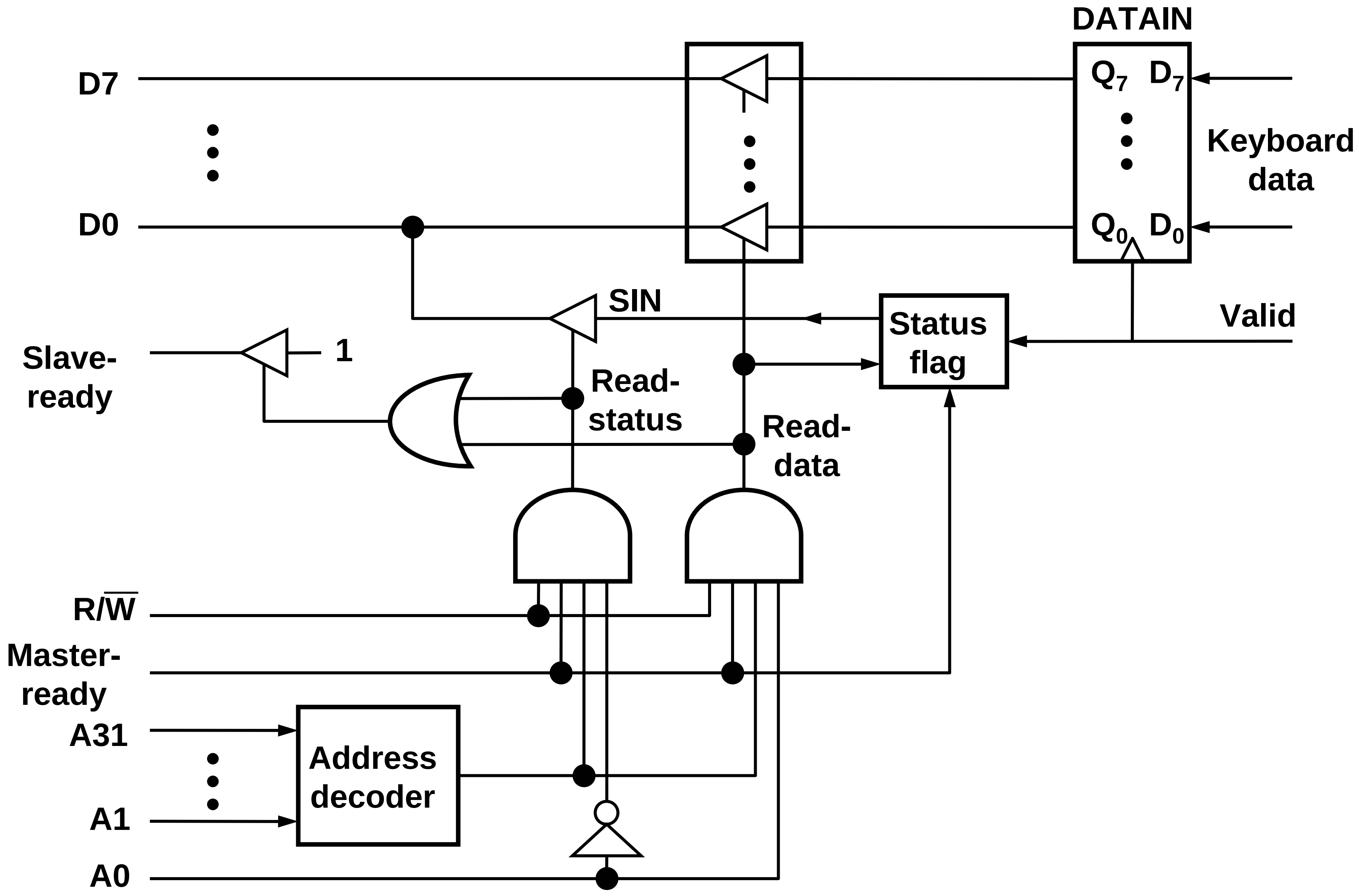
- The choice of a particular design involves trade-offs among factors such as
  - Simplicity of the device interface
  - Ability to accommodate device interfaces that introduce different amounts of delay
  - Total time required for bus transfer
  - Ability to detect errors results from addressing a nonexistent device or from an interface malfunction
- Asynchronous bus
  - The handshake process eliminates the need for synchronization of the sender and receiver clock, thus simplifying timing design
- Synchronous bus
  - Clock circuitry must be designed carefully to ensure proper synchronization, and delays must be kept within strict bounds

Interface Circuits

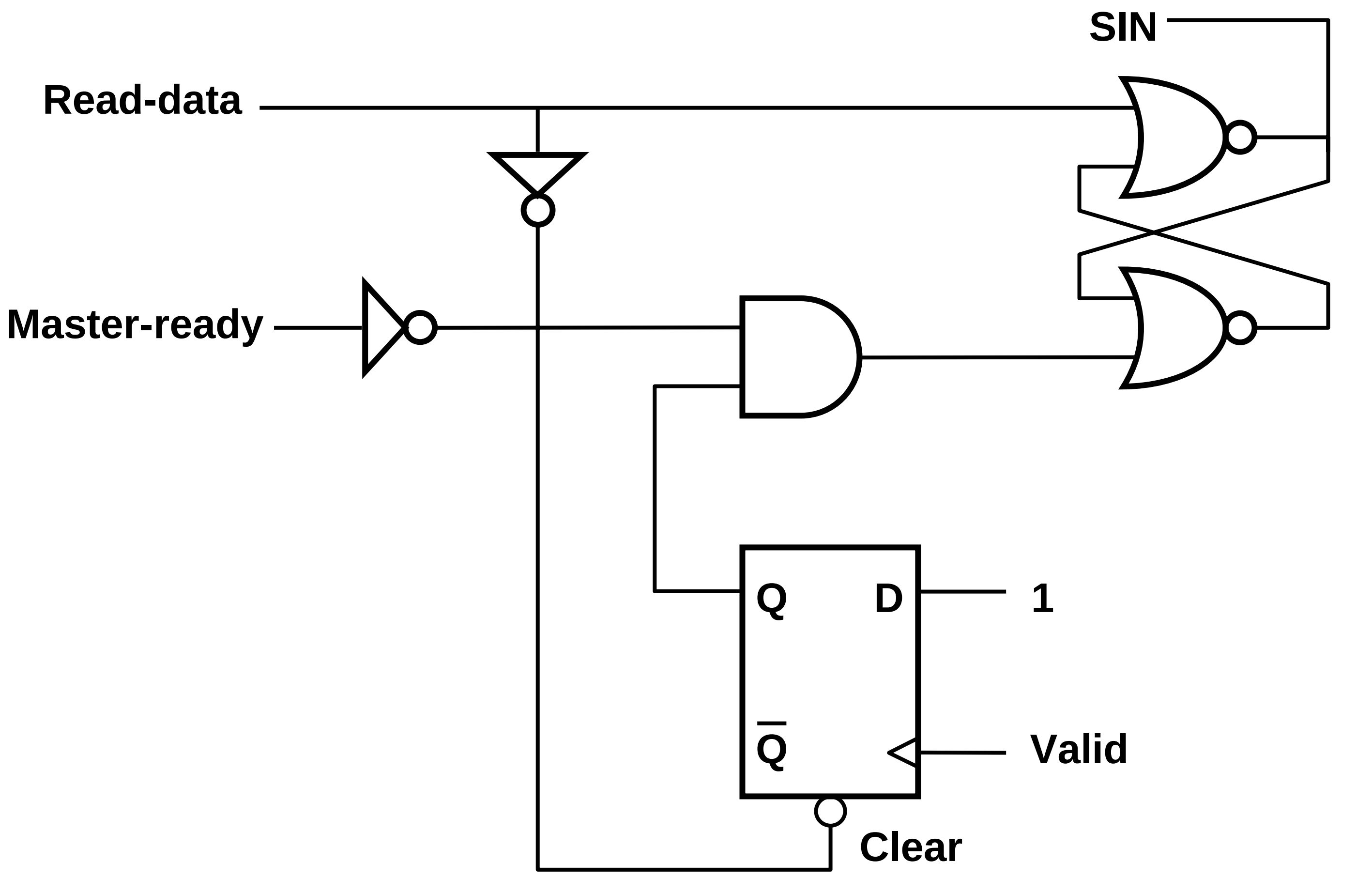
- Keyboard to processor connection
  - When a key is pressed, the Valid signal changes from 0 o 1, causing the ASCII code to be loaded into DATAIN and SIN to be set to 1
  - The status flag SIN is cleared to 0 when the processor reads the contents of the DATAIN register



Input Interface Circuit

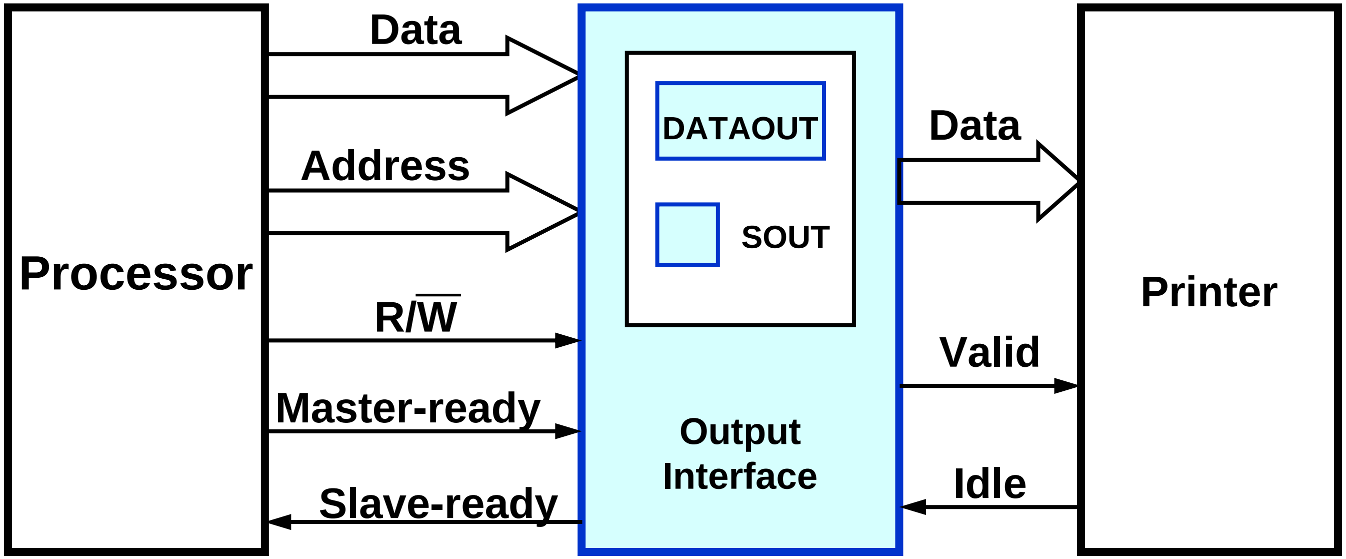


Circuit for the Status Flag Block

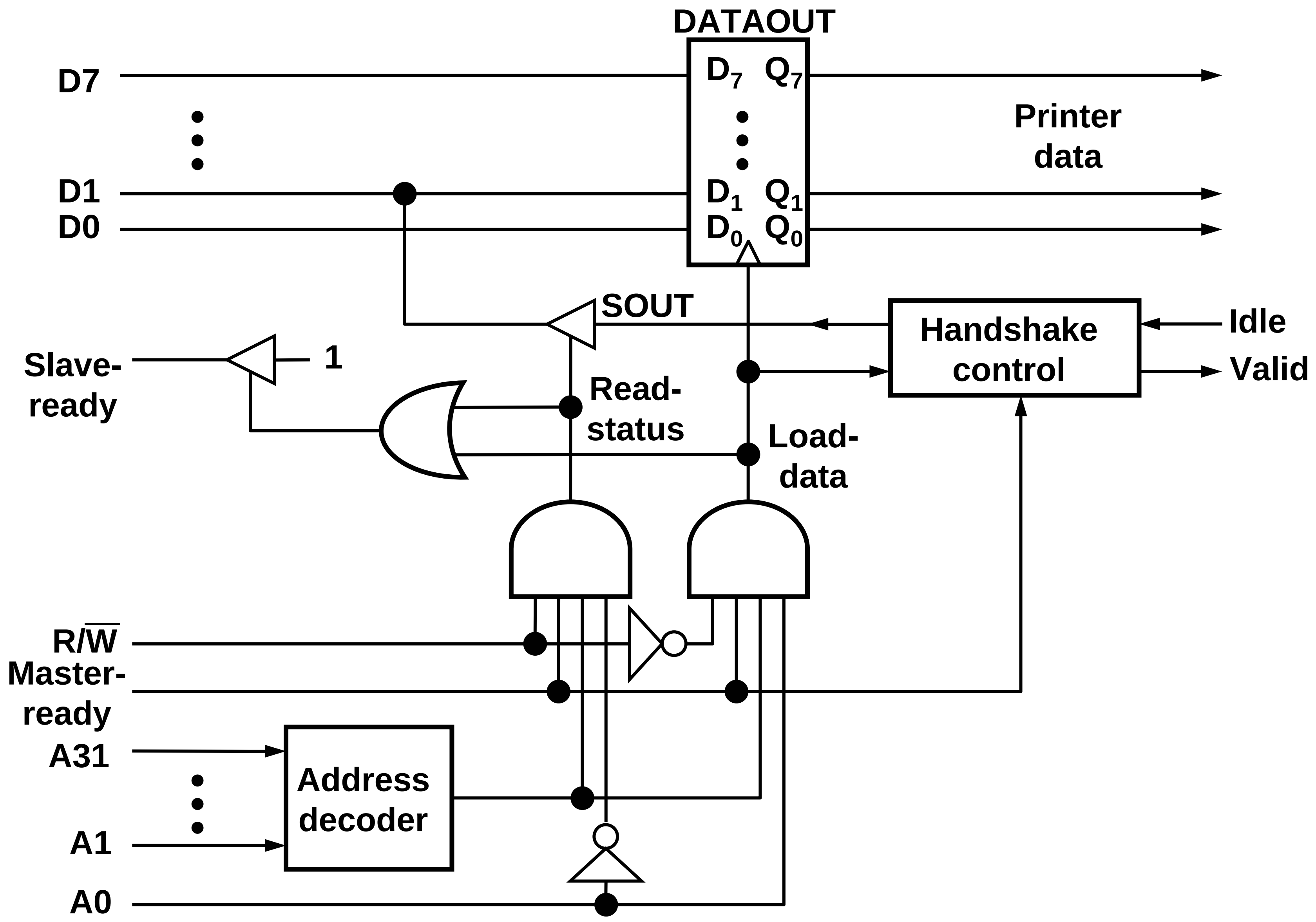


Printer to Processor Connection

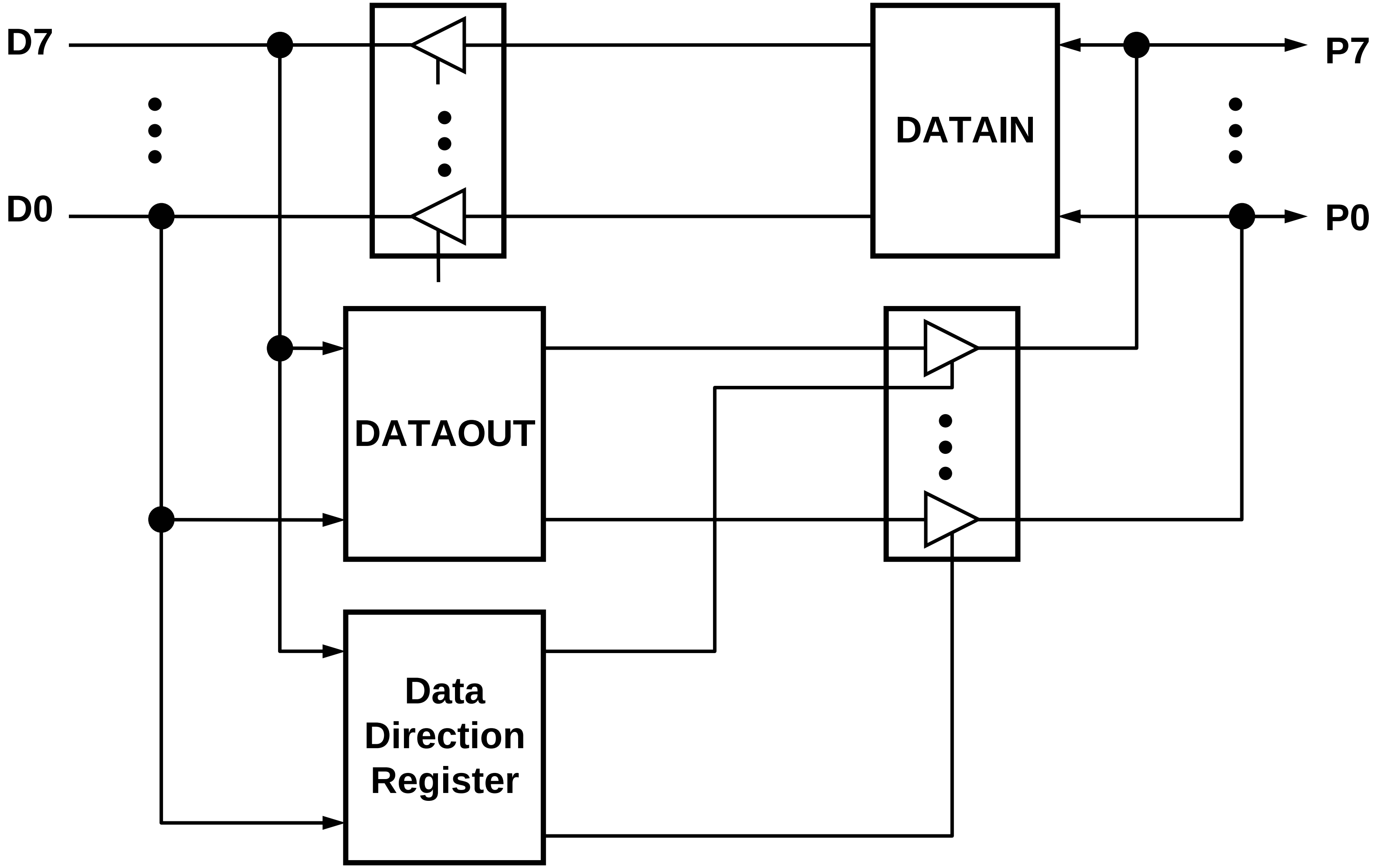
- The interface contains a data register, DATAOUT, and a status flag, SOUT
  - The SOUT flag is set to 1 when the printer is ready to accept another character, and it is cleared to 0 when a new character is loaded into DATAOUT by the processor
  - When the printer is ready to accept a character, it asserts its idle signal



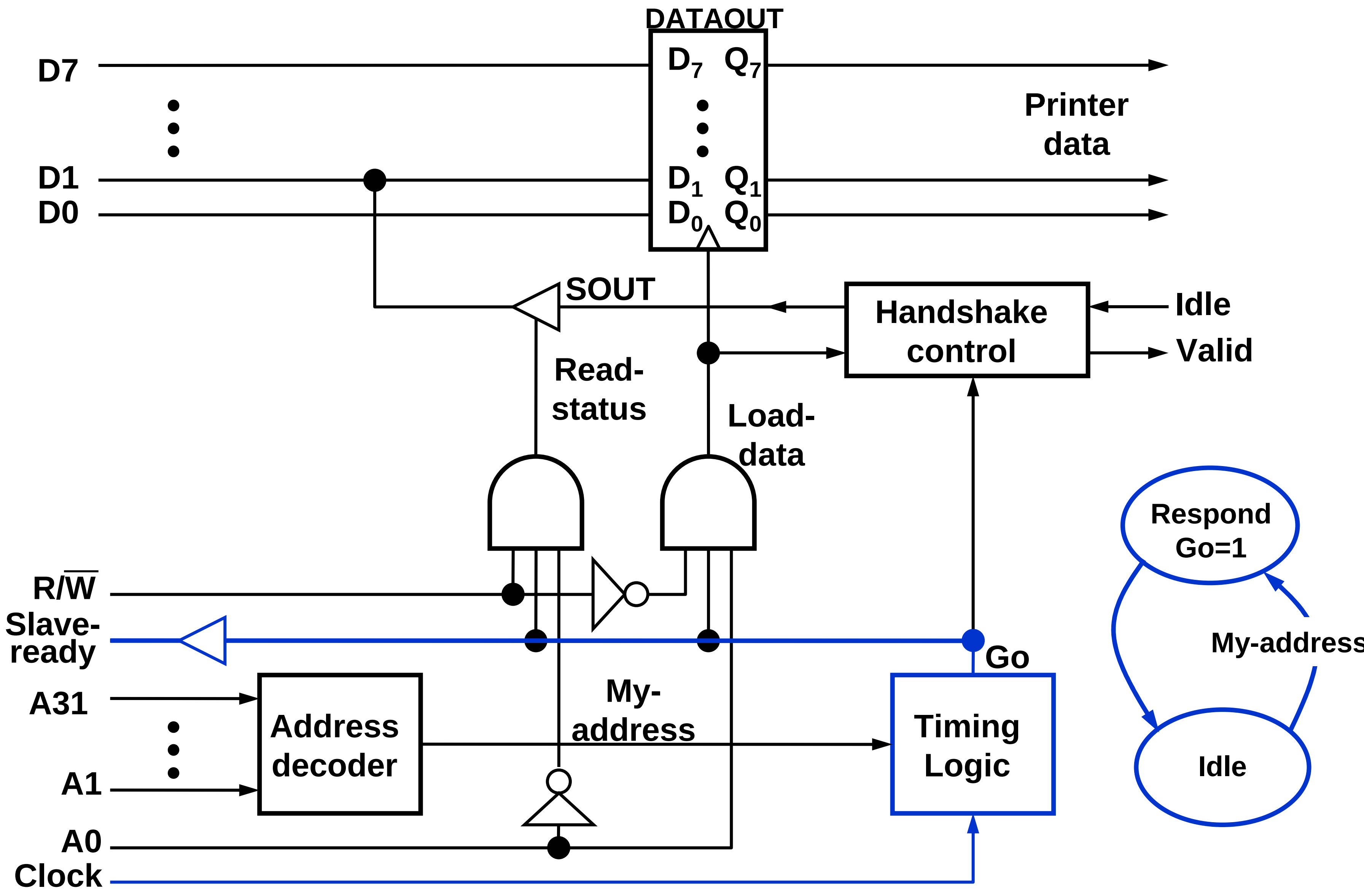
Output Interface Circuit



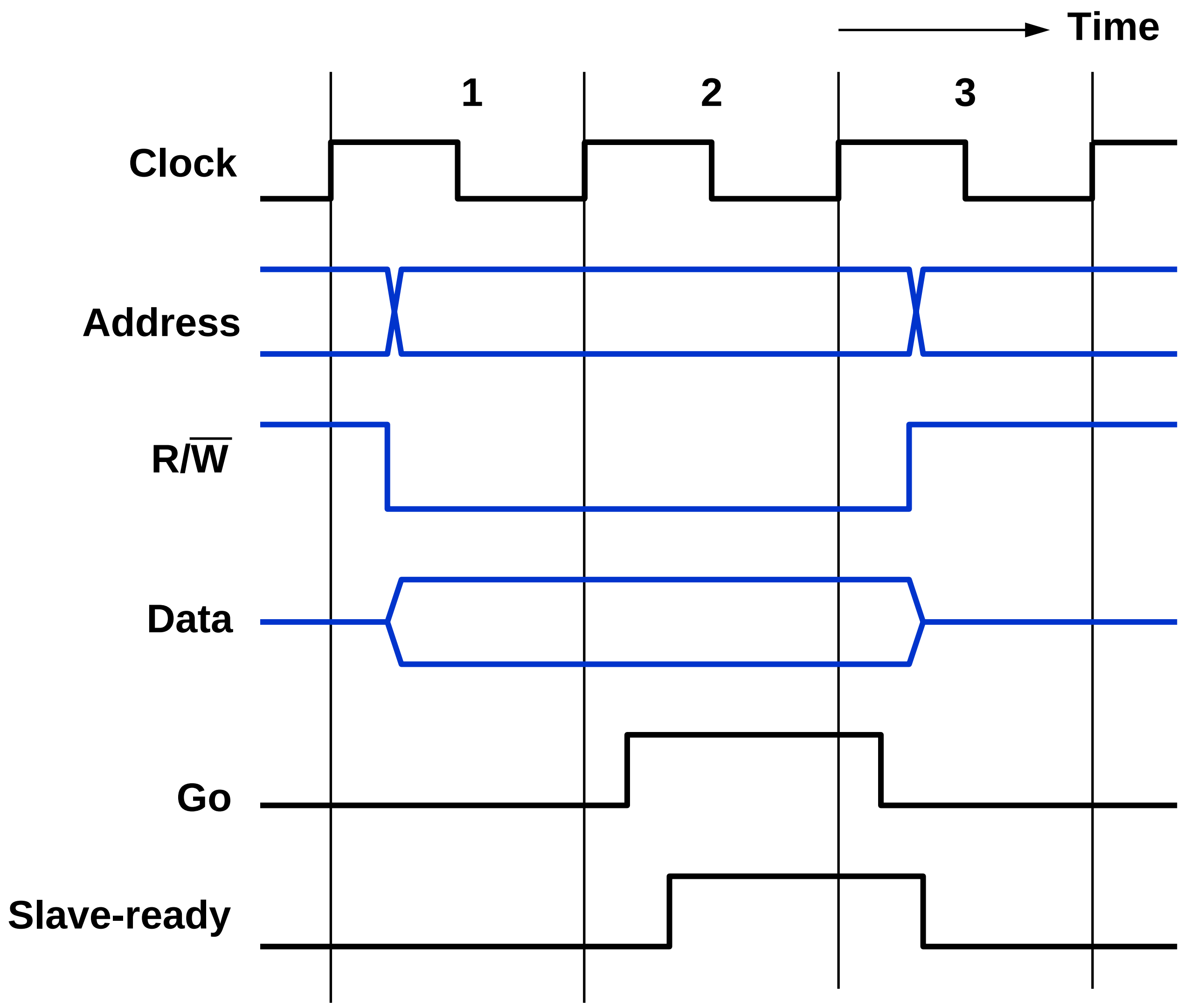
A General 8-Bit Parallel Interface



Output Interface Circuit for a Bus Protocol



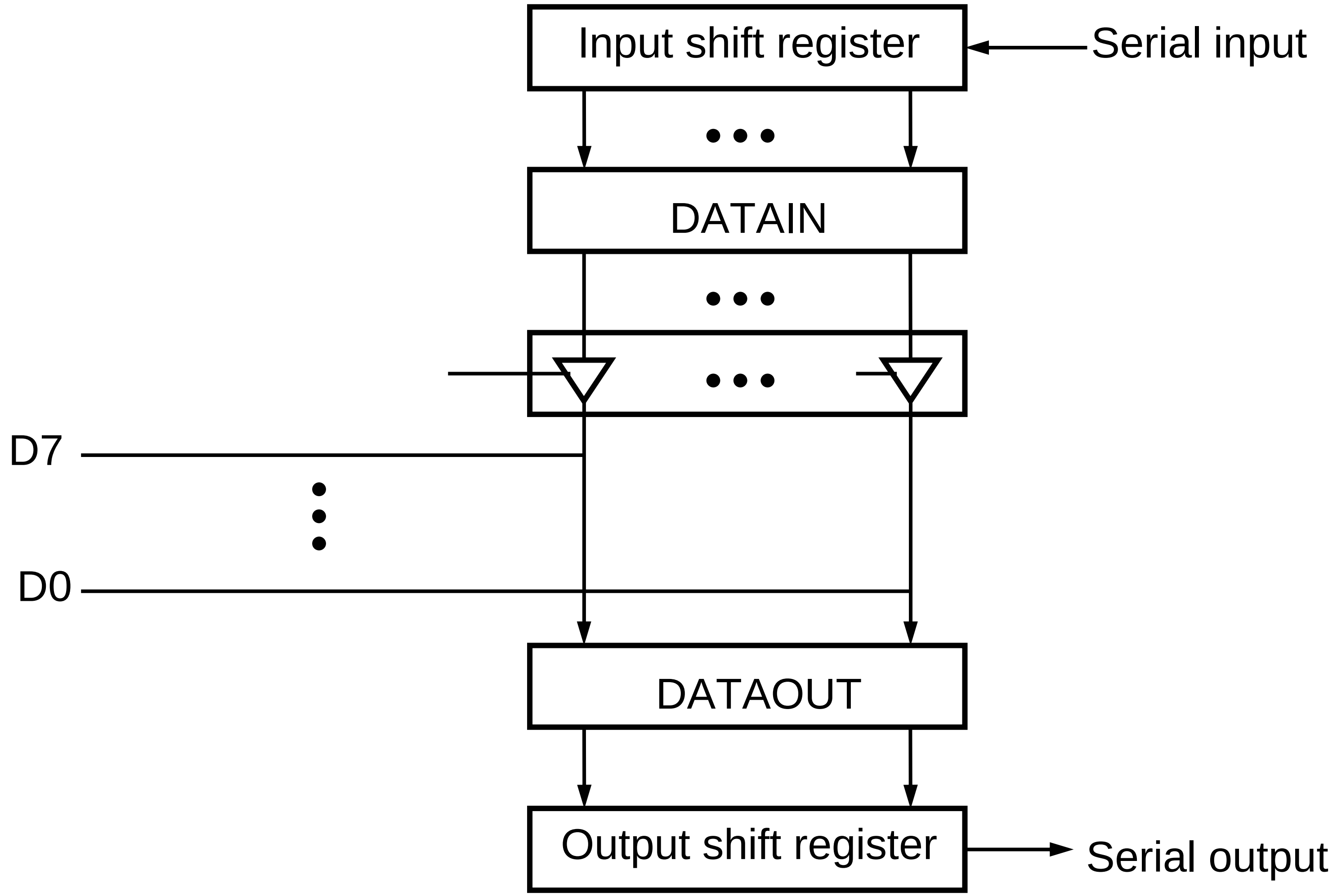
Timing Diagram for an Output Operation



Serial Port

- A serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time
- The key feature of an interface circuit for a serial port is that it is capable of communicating in a bit-serial fashion on the device side and in a bit-parallel fashion on the bus side
- The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability

A Serial Interface

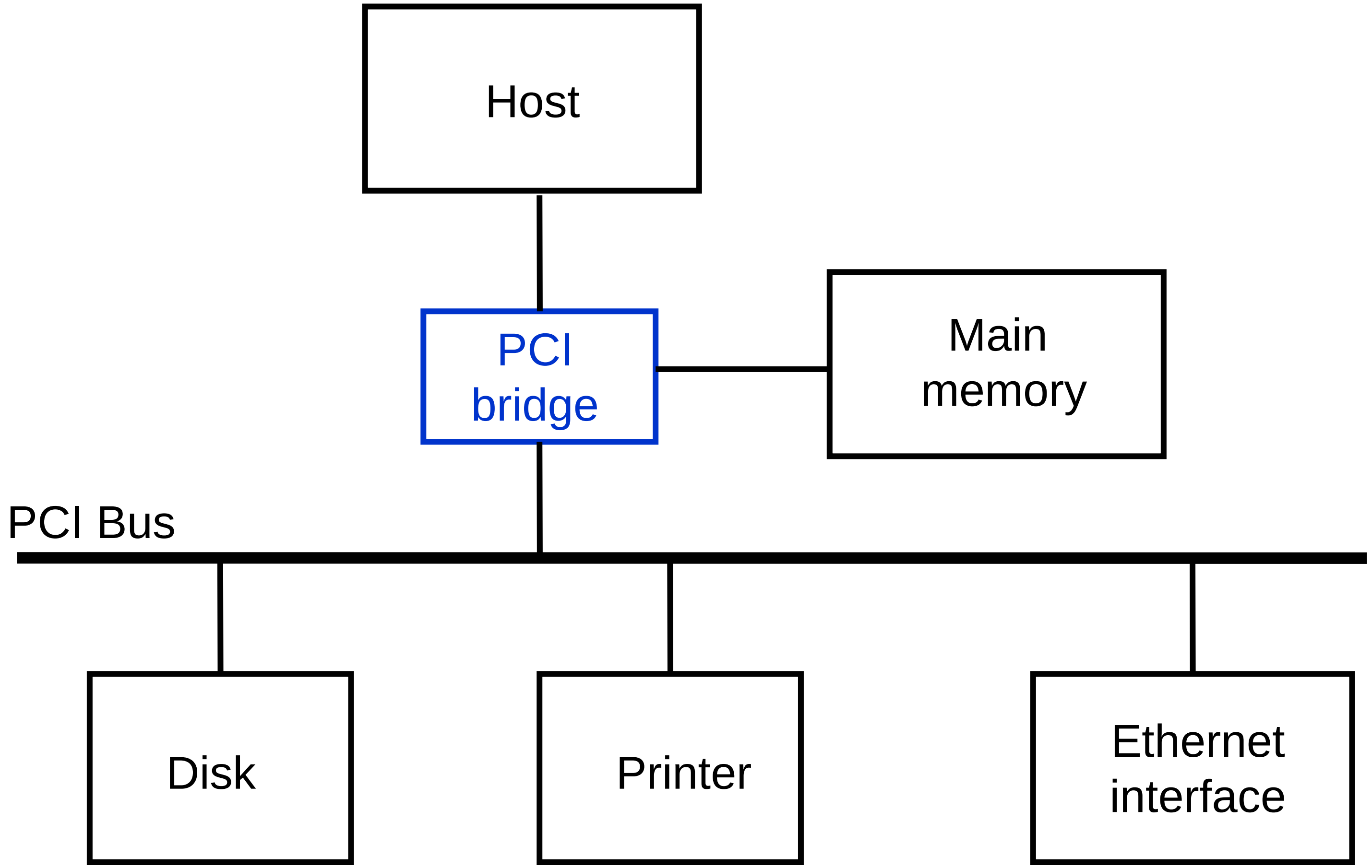


Standard I/O Interfaces

- The processor bus is the bus defined by the signals on the processor chip itself. Devices that require a very high speed connection to the processor, such as the main memory, may be connected directly to this bus
- The motherboard usually provides another bus that can support more devices.
- The two buses are interconnected by a circuit, which we called a bridge, that translates the signals and protocols of one bus into those of the other
- It is impossible to define a uniform standards for the processor bus. The structure of this bus is closely tied to the architecture of the processor
- The expansion bus is not subject to these limitations, and therefore it can use a standardized signaling structure

Peripheral Component Interconnect Bus

- Use of a PCI bus in a computer system



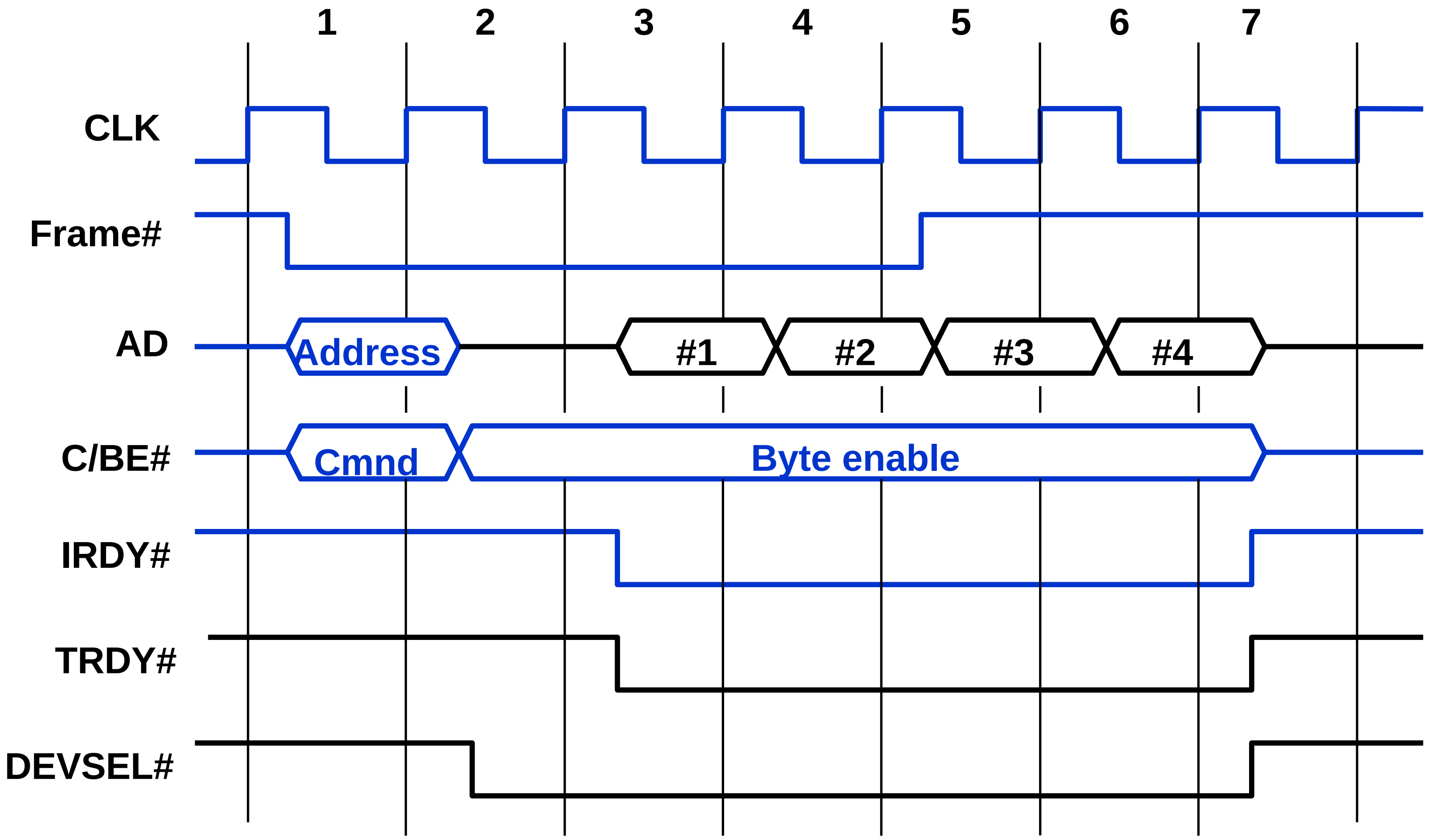
PCI Bus

- The bus support three independent address spaces: memory, I/O, and configuration.
- The I/O address space is intended for use with processors, such Pentium, that have a separate I/O address space.
- However, the system designer may choose to use memory-mapped I/O even when a separate I/O address space is available
- The configuration space is intended to give the PCI its plug-and-play capability.
  - A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation

Data Transfer Signals on the PCI Bus

Name	Function
CLK	A 33-MHz or 66MHz clock
FRAME#	Sent by the initiator to indicate the duration of a transaction
AD	32 address/data lines, which may be optionally increased to 64
C/BE#	4 command/byte-enable lines (8 for 64-bit bus)
IRDY#, TRDY#	Initiator-ready and Target-ready signals
DEVSEL#	A response from the device indicating that it has recognized its Address and is ready for a data transfer transaction
IDSEL#	Initialization Device Select

A Read Operation on the PCI Bus



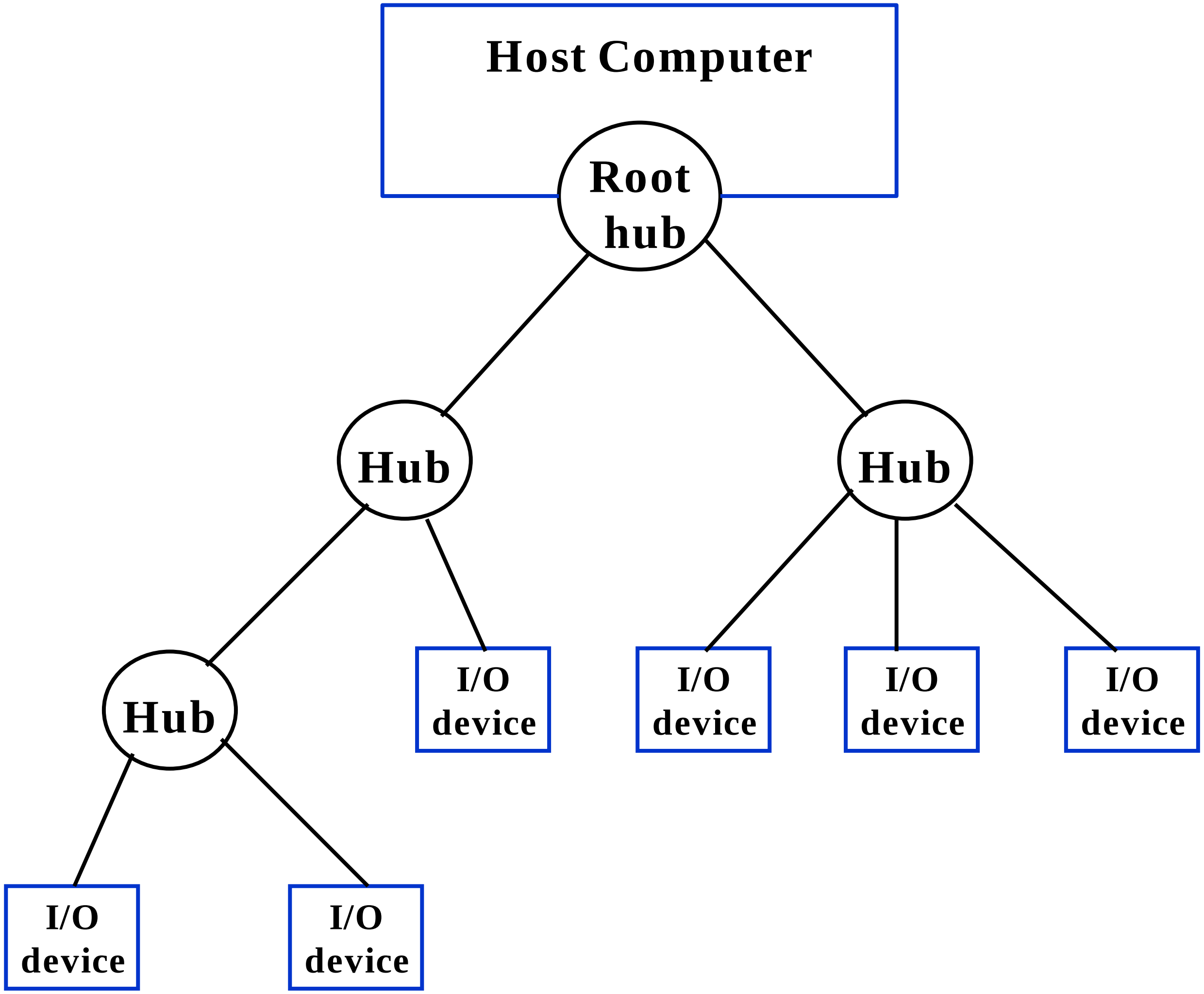
Universal Serial Bus (USB)

- The USB has been designed to meet several key objectives
  - Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer
  - Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections
  - Enhance user convenience through a “plug-and-play” mode of operation

USB Structure

- A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements
- Clock and data information are encoded together and transmitted as a single signal
  - Hence, there are no limitations on clock frequency or distance arising from data skew
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure
  - Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O device
  - At the root of the tree, a root hub connects the entire tree to the host computer

USB Tree Structure



USB Tree Structure

- The tree structure enables many devices to be connected while using only simple point-to-point serial links
- Each hub has a number of ports where devices may be connected, including other hubs
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports
  - As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message
- A message sent from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices
  - Hence, USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other

USB Protocols

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information
- The information transferred on the USB can be divided into two broad categories: control and data
  - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error
  - Data packets carry information that is delivered to a device. For example, input and output data are transferred inside data packets