Contents of the Lecture

- 1. Introduction
- 2. Methods for I/O Operations
- 3. Computer Buses
- 4. Expansion Modules for Embedded Systems
- 5. Computer Displays
- 6. Graphics Adapters
- 7. Optical Discs

2. Methods for I/O Operations

- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access (DMA)
- I/O Processors

Programmed I/O

- Principle of Programmed I/O
- I/O Device Addressing
- I/O Instructions
- Disadvantages of Programmed I/O

Principle of Programmed I/O (1)

- Data are transferred between the CPU and I/O module under direct control of the CPU
  - Each transfer operation requires the execution of an instruction sequence by the CPU
  - The transfer is carried out between a CPU register and a register of the I/O device
  - The I/O device does not have direct access to main memory

Principle of Programmed I/O (2)

- Execution of an I/O operation:
  - The CPU sends a command to the I/O module
  - The I/O module performs the requested action and sets the appropriate bits in the status register
  - The CPU must periodically check the status of the I/O module to detect that the operation is complete

Programmed I/O

- Principle of Programmed I/O
- I/O Device Addressing
- I/O Instructions
- Disadvantages of Programmed I/O

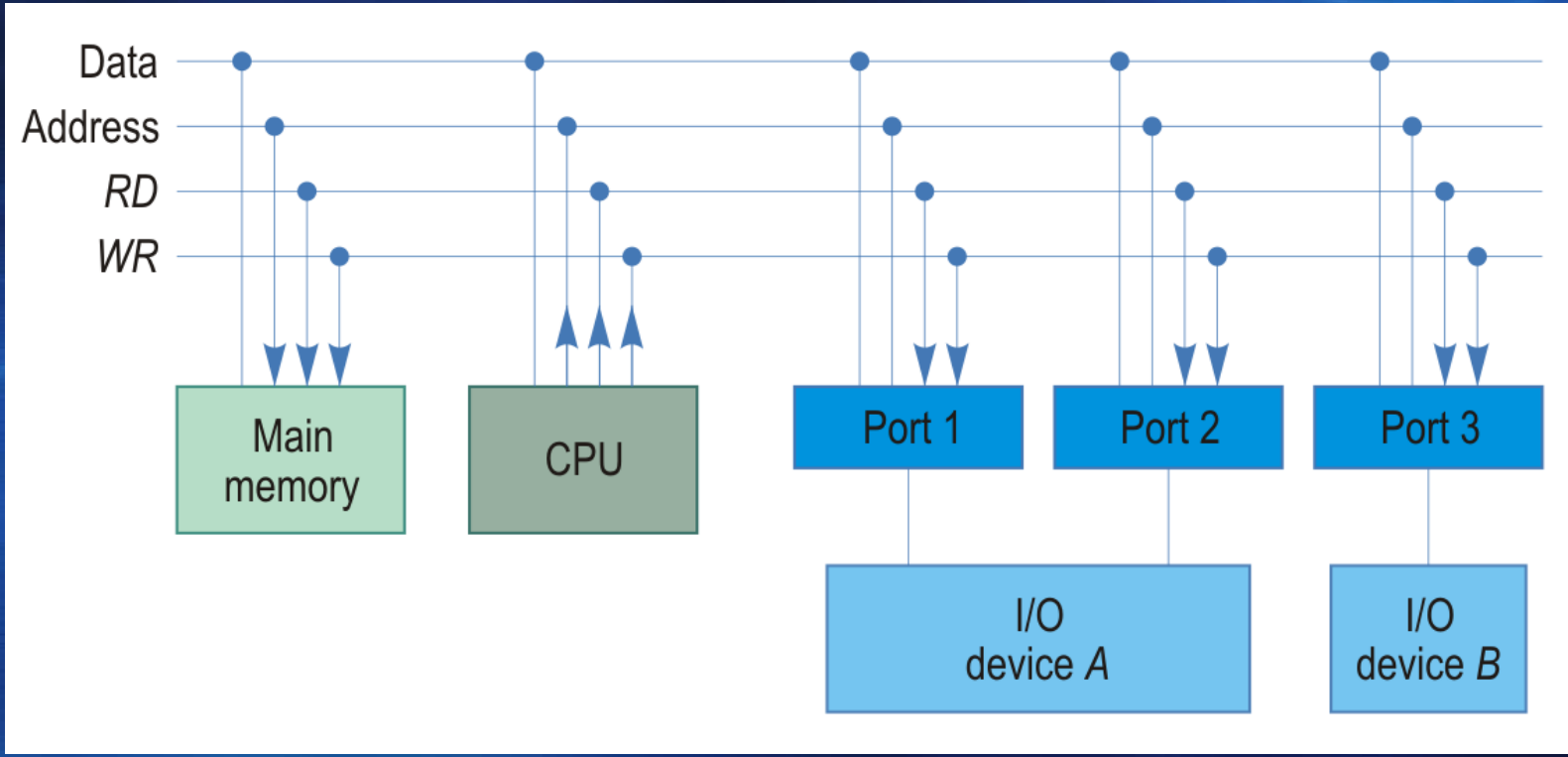
I/O Device Addressing (1)

- The CPU, memory, and I/O devices usually communicate via the system bus
- An I/O device is connected to the bus via an I/O port → addressable register
- When the CPU, main memory, and I/O system share a common bus, two addressing techniques are possible:
  - *Memory-mapped* addressing
  - *Isolated* addressing

I/O Device Addressing (2)

- Memory-mapped addressing
  - There is a single address space for memory locations and I/O devices
  - The CPU treats the status and data registers of I/O modules as memory locations
    - The same instructions are used to access both memory and I/O devices
    - No special I/O instructions are needed → memory load and store instructions

I/O Device Addressing (3)

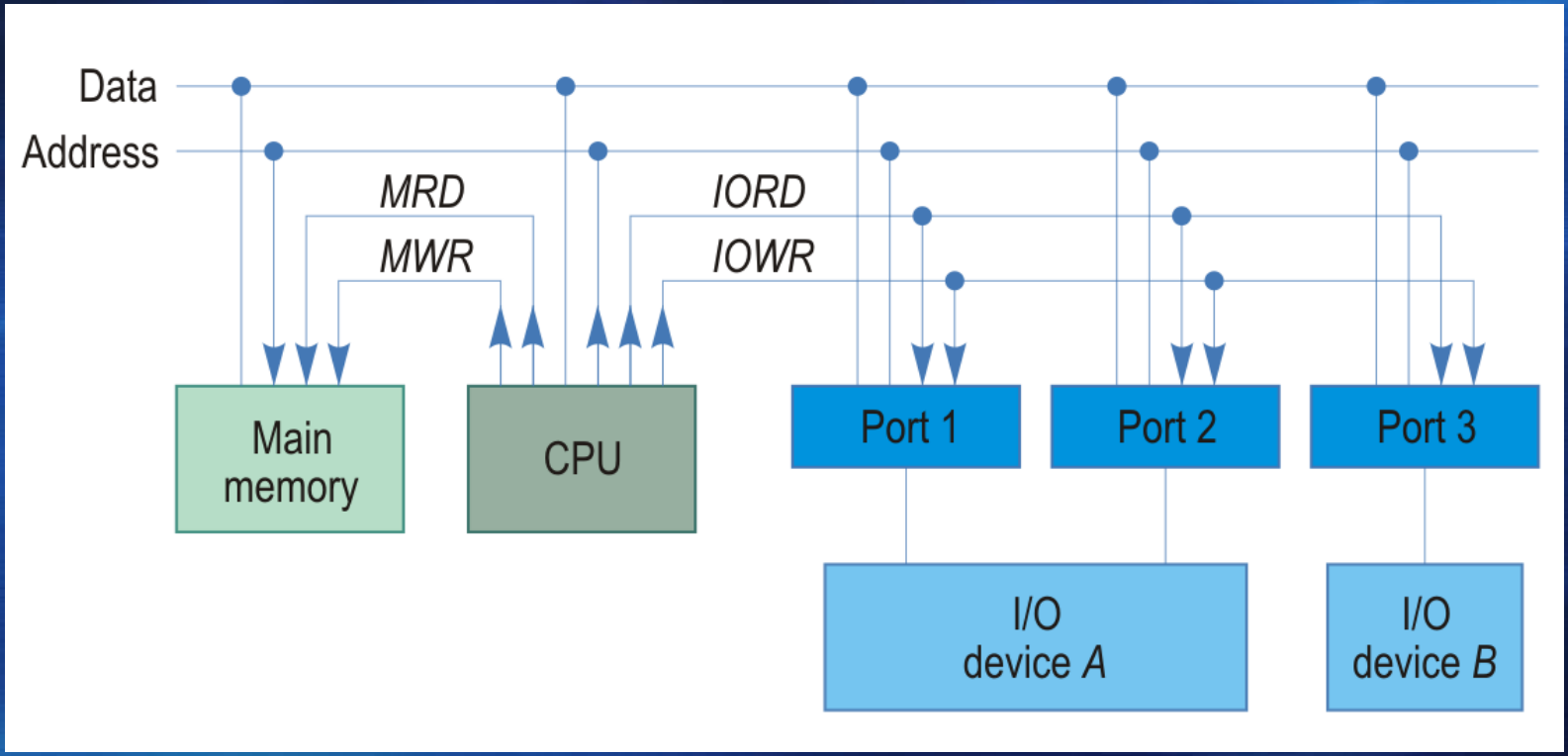


- The *RD* and *WR* control lines are used to initiate either a memory access cycle or an I/O transfer

I/O Device Addressing (4)

- Isolated Addressing
  - The I/O address space is isolated from that for memory
  - The bus must contain:
    - Read and write lines for the memory
    - Command lines for input and output
  - A memory-referencing instruction asserts the *MRD* or *MWR* control line
  - The CPU must execute separate I/O instructions to assert the *IORD* and *IOWR* lines

I/O Device Addressing (5)



Programmed I/O

- Principle of Programmed I/O
- I/O Device Addressing
- I/O Instructions
- Disadvantages of Programmed I/O

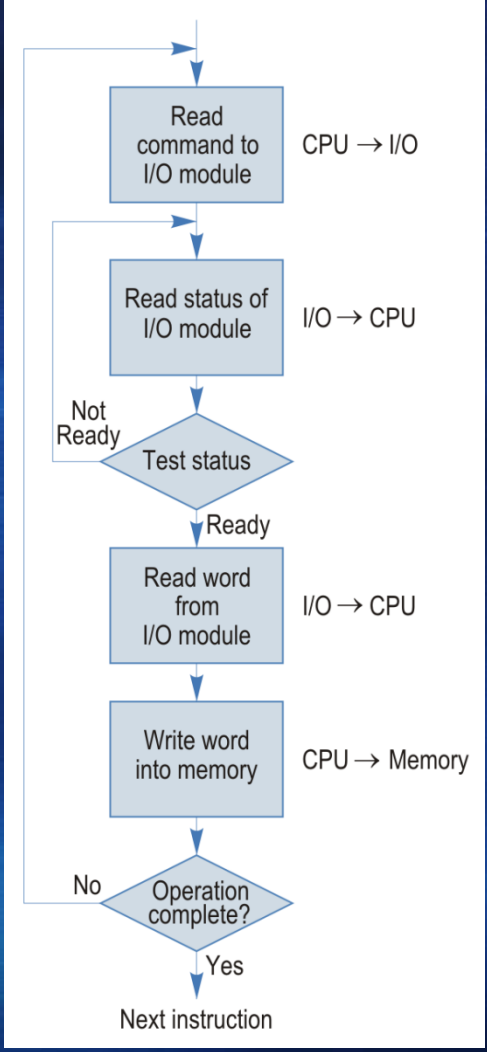
I/O Instructions (1)

- Programmed I/O can be implemented by at least two I/O instructions
  - IN, OUT (Intel)
- To prevent the loss of information or an indefinite execution time, the CPU must test the status of the I/O device
- To execute an I/O instruction, the CPU sends:
  - An address: the I/O module and the external device
  - An I/O command

I/O Instructions (2)

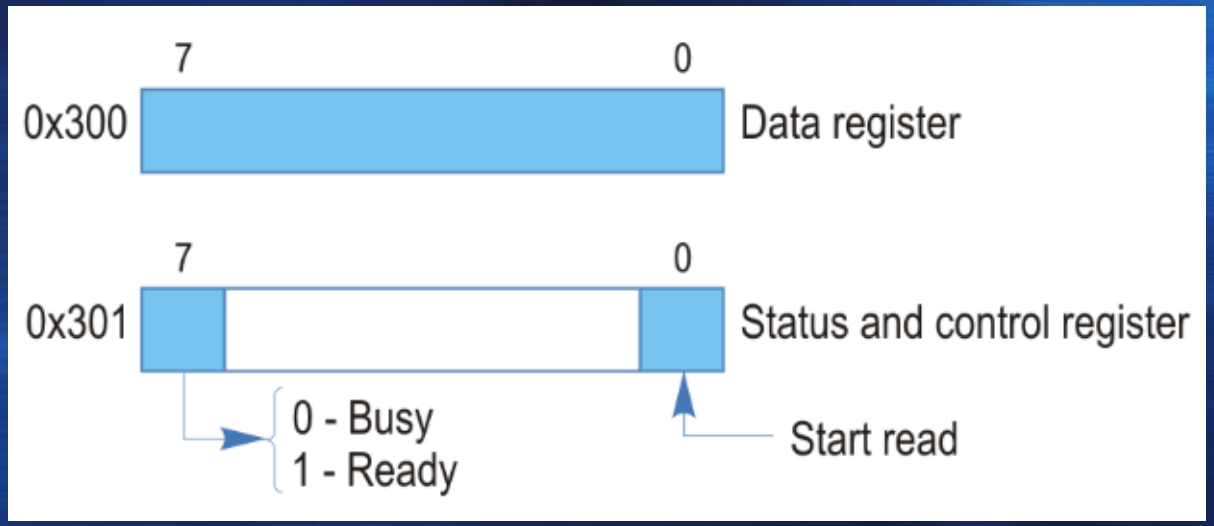
- Types of I/O commands
  - Control: used to activate a peripheral and to specify the operation to be executed
  - Test: used to test status conditions associated with an I/O module and its peripherals
  - Read: used to obtain a byte or word from the peripheral
  - Write: used to send a byte or word to the peripheral

I/O Instructions (3)



- Reading a block of data from a peripheral device into memory
- For each word that is read in, the CPU must remain in a cycle to test the status

I/O Instructions (4)



- The programmer's interface for a terminal keyboard
- Exemplifying the programmed I/O for memory-mapped addressing and isolated addressing

I/O Instructions (5)

- Memory-mapped addressing
  - 512 memory locations (0x000 – 0x1FF)
  - 512 I/O addresses (0x200 – 0x3FF)
  - Reading a byte from the keyboard

```
        LD    AC, 0x01
        ST    0x301, AC     ; start keyboard read
WAIT:   LD    AC, 0x301     ; read status byte
        AND   AC, 0x80      ; isolate bit 7
        BZ    WAIT          ; wait for the byte
        LD    AC, 0x300     ; read data byte
```

I/O Instructions (6)

- Isolated addressing
  - The I/O ports have the same addresses as in the previous example
  - Reading a byte from the keyboard

```
        LD      AC, 0x01
        OUT     0x301, AC       ; start keyboard read
WAIT:   IN      AC, 0x301       ; read status byte
        AND     AC, 0x80        ; isolate bit 7
        BZ      WAIT            ; wait for the byte
        IN      AC, 0x300       ; read data byte
```

Programmed I/O

- Principle of Programmed I/O
- I/O Device Addressing
- I/O Instructions
- Disadvantages of Programmed I/O

Disadvantages of Programmed I/O

- Performance of the system is significantly reduced, because:
  - The CPU has to wait until the peripheral becomes available, and then to execute the transfer by a program sequence
  - The transfer rate is limited by the speed with which the CPU can test and serve the I/O devices

2. Methods for I/O Operations

- Programmed I/O
- Interrupt-Driven I/O
- Direct Memory Access (DMA)
- I/O Processors

Interrupt-Driven I/O

- Principle of Interrupt-Driven I/O
- Multiple-Interrupt Systems
- Priority Interrupt Systems
  - Parallel Priority Interrupts
  - Daisy-Chain Priority Interrupts

Principle of Interrupt-Driven I/O (1)

- Interrupt: suspension of program execution by an external signal or by an internal event of the CPU
- Program suspension occurs at the end of the current instruction's execution
- The CPU is relieved from the task of testing the status of I/O devices
- Interrupt sources can be external or internal to the CPU

Principle of Interrupt-Driven I/O (2)

- Examples of interrupt sources:
  - Peripheral devices → data transfers
  - Virtual memory → page transfers
  - Hardware circuits for supervising normal operation of the system: detecting memory errors, power-supply failures
  - Internal software events: overflows, divisions by zero, non-existent or privileged instructions

Principle of Interrupt-Driven I/O (3)

- For interrupting the CPU, a control line is asserted → *IREQ* (*Interrupt Request*)
  - An interrupt flag is set
- When recognizing the interrupt request, the CPU:
  - Asserts an interrupt acknowledge signal → *IACK* (*Interrupt Acknowledge*)
  - Executes an interrupt handler routine, associated with the interrupt source

Principle of Interrupt-Driven I/O (4)

- For transferring control to the interrupt handler routine:
  - The CPU identifies the interrupt source
  - The CPU determines the address of the interrupt handler corresponding to the interrupt source
  - The CPU saves the program counter (PC) and other status information
  - The CPU loads the address of the interrupt handler into the program counter

Principle of Interrupt-Driven I/O (5)

- The CPU has to determine the address of the interrupt handler
- Methods for choosing the address of the interrupt handler:
  - Non-vectored interrupts: the interrupt handler is located at a fixed address in memory
  - Vectored interrupts: the address is supplied by the interrupt source, in the form of an *interrupt vector*

Interrupt-Driven I/O

- Principle of Interrupt-Driven I/O
- Multiple-Interrupt Systems
- Priority Interrupt Systems
  - Parallel Priority Interrupts
  - Daisy-Chain Priority Interrupts

Multiple-Interrupt Systems (1)

- For registering the interrupt requests, an *interrupt request register* is used
- For an individual control of interrupt sources, mask flip-flops are used → *interrupt mask register*
- Main problems:
  - Identifying the source of interrupt
  - Choosing the interrupt to service in case of several simultaneous requests

Multiple-Interrupt Systems (2)

- Techniques for identifying the source of interrupt:
  - Multiple interrupt lines
  - Software polling
  - Connecting the devices in a daisy chain (hardware polling)
  - Bus arbitration

Multiple-Interrupt Systems (3)

- Multiple interrupt lines between the CPU and the I/O modules
  - The simplest solution
  - It is impractical to dedicate a large number of bus lines or processor pins to interrupt lines
  - Usually, multiple I/O modules will be attached to each line

Multiple-Interrupt Systems (4)

- Software polling
  - When the CPU detects an interrupt, it executes an interrupt-service routine
  - The I/O modules are interrogated (polled) to determine which module generated the interrupt
  - For polling, a separate command line may be used (e.g., *TEST I/O*)
  - Each I/O module may contain an addressable status register

Multiple-Interrupt Systems (5)

- Hardware polling
  - A daisy-chain of devices is used
  - All I/O modules share a common interrupt request line
  - When detects an interrupt request, the CPU asserts an interrupt acknowledge signal
  - The interrupt acknowledge signal is daisy-chained through the I/O modules

Multiple-Interrupt Systems (6)

- The acknowledge signal propagates through the I/O modules until it reaches a requesting module
- This module responds by placing an interrupt vector on the data bus
- The CPU uses the vector as a pointer to the service routine for the module
- Advantage: there is no need to execute a general interrupt service routine

Multiple-Interrupt Systems (7)

- Bus arbitration
  - Uses vectored interrupts
  - An I/O module must first gain control of the bus before it can assert the interrupt request signal
  - When detects the interrupt, the CPU asserts the interrupt acknowledge signal
  - The requesting module places its vector on the data lines

Summary (1)

- Programmed I/O: the CPU executes a sequence of instructions for each transfer
- Addressing techniques
  - Memory-mapped addressing: the registers of I/O modules are treated as memory locations
  - Isolated addressing: the registers of I/O modules have addresses in an address space separated from that of the memory
- Programmed I/O has major disadvantages

Summary (2)

- Interrupts relieve the CPU from testing the status of I/O devices
- Two methods for choosing the address of the interrupt handler: non-vectored or vectored interrupts
- Techniques for identifying the source of interrupt: multiple interrupt lines; software polling; hardware polling; bus arbitration

Concepts, Knowledge (1)

- Principle of programmed I/O
- Execution of an I/O operation
- Memory-mapped addressing
- Isolated addressing
- Disadvantages of programmed I/O
- Operations executed by the CPU when detecting an interrupt request
- Methods for choosing the address of the interrupt handler

Concepts, Knowledge (2)

- Non-vectored interrupts
- Vectored interrupts
- Techniques for identifying the source of interrupt in a multiple-interrupt system
- Software polling technique
- Hardware polling technique
- Bus arbitration technique