Estructura de Computadores

Grado de Ingeniería Informática

ETSINF

Unit 7: Input/Output Interfaces

Course 2020/2021

DISCA

Unit goals

- To meet the concept of I/O interface
- To understand the mechanisms for I/O addressing
- To design simple selection circuits for I/O interfaces according to a given I/O address map
- To acquire a programmer's view of I/O devices
- To write assembly programs that handle devices with simple I/O interfaces

Contents

Bibliography

- D. Patterson, J. Hennessy
  - ✓ *Computer organization and design. The hardware/software interface*. 4th edition. 2009. Elsevier
    - Chapter 6
- W. Stallings
  - ✓ *Computer Organization and Architecture. Designing for Performance*. 7th edition. 2006. Prentice Hall
    - Chapter 7
- C. Hamacher, Z. Vranesic, S. Zaky
  - ✓ *Computer Organization*. 5th edition. 2001. McGraw-Hill
    - Chapter 4

I – The Input/Output System

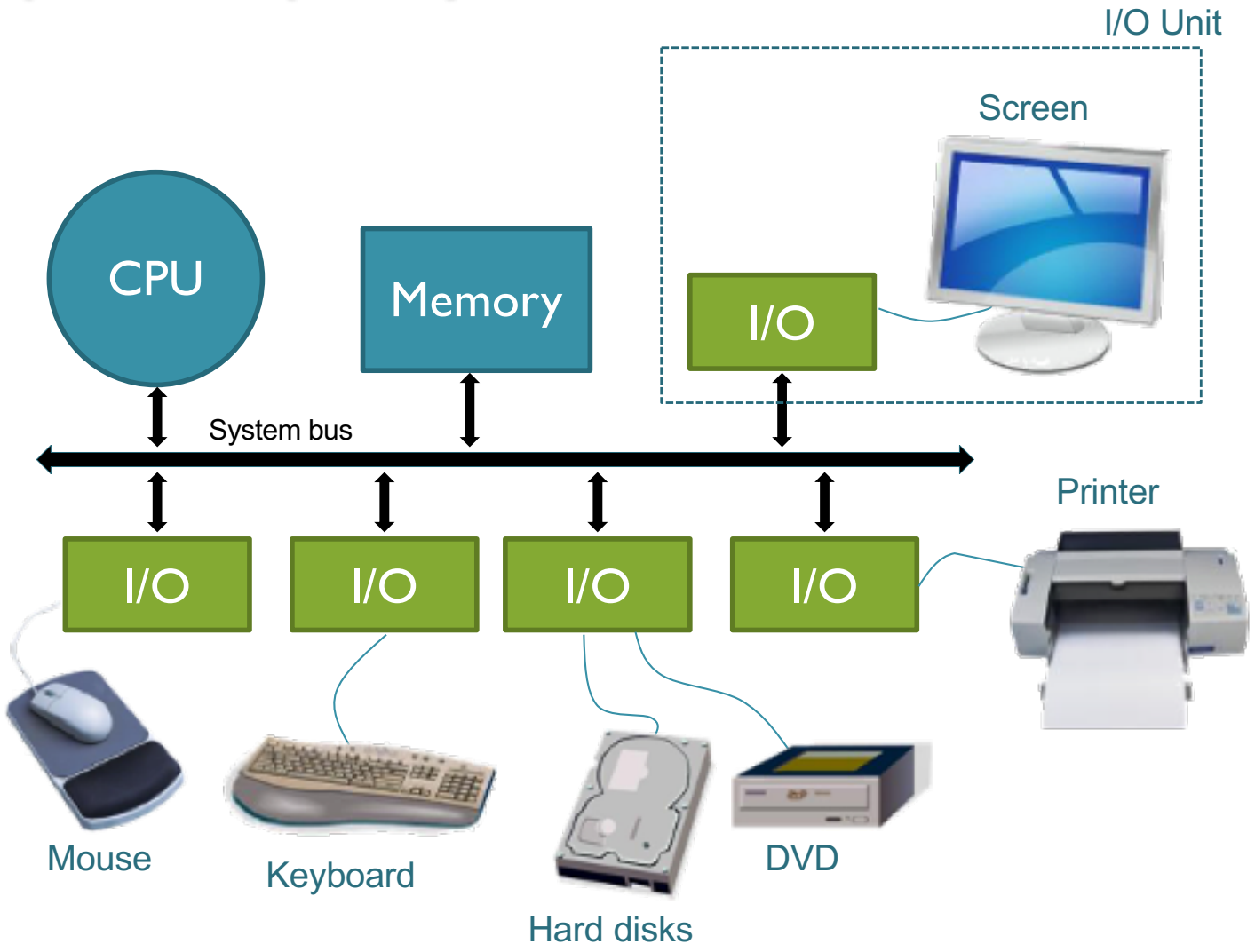
The Big Picture

Input/Output system

- The I/O system communicates CPU and memory with the *external world* of peripheral devices
  - ✓ How do programs and data arrive in main memory?
  - ✓ How do programs display results and accept user inputs?
- Diversity of *peripheral* devices and physical support, e.g.:

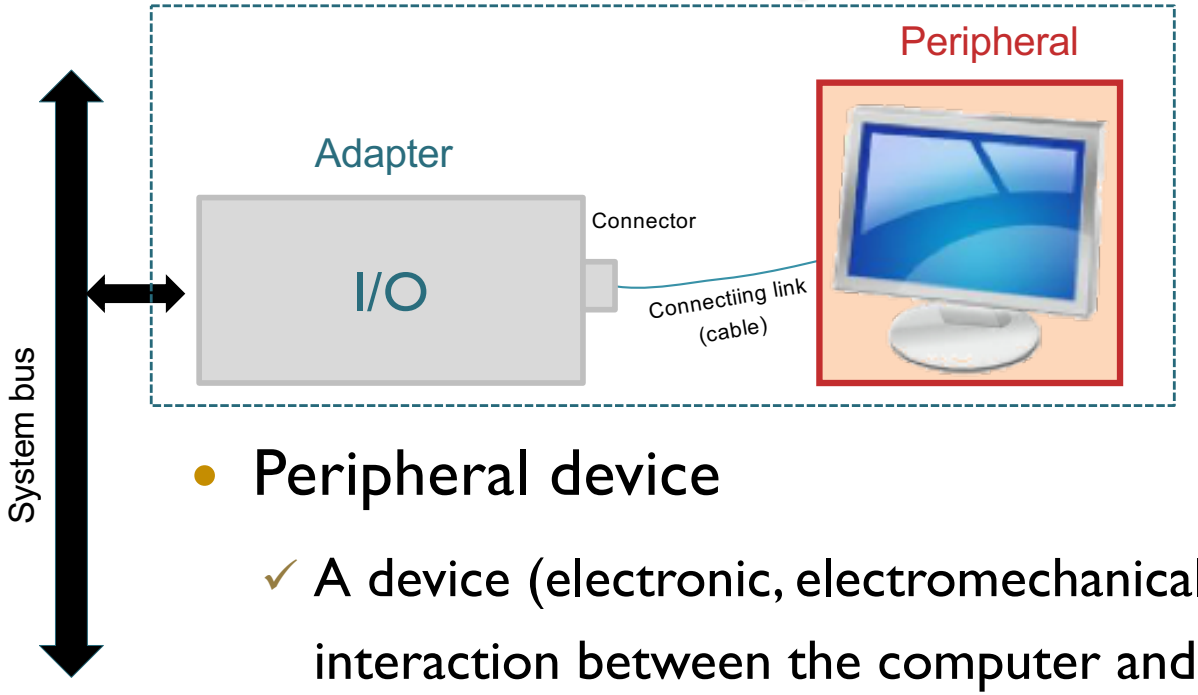
Device	Use	Physical support
Keyboard	Text input	Electro-mechanical
Mouse	Graphical input	Optical
Screen	Display	CRT, TFT array, ...
Network	Communication	Ethernet, WiFi, ADSL, DSL,…
Printer	Print on paper	Electrostatic, ink injection,…
3D Printer	Build objects	Plastic injection, electro-mech.
Hard disk	Storage	Magnetic
DVD	Storage	Optical

The Input/Output system



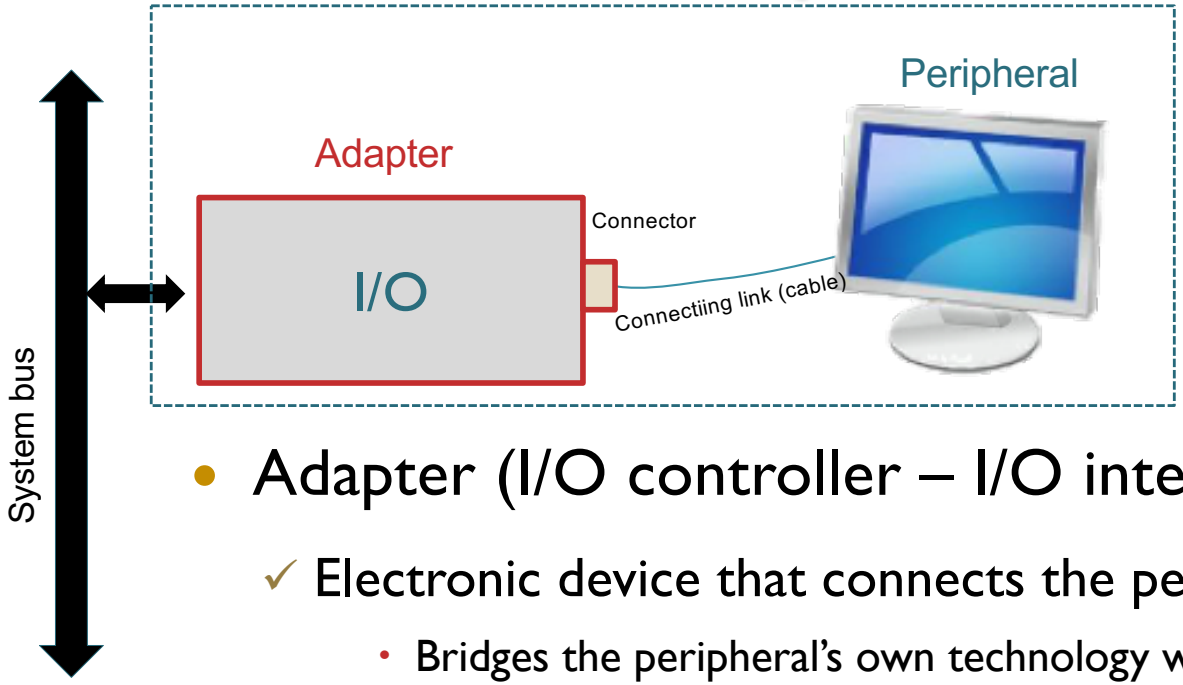
Peripherals require specific interfaces

Elements of the I/O System



- Peripheral device
  - ✓ A device (electronic, electromechanical,…) that enables interaction between the computer and its environment
    - Human-Machine Interaction (HMI): keyboard, mouse, screen, …
    - Interaction with other devices: motors, actuators, sensors,…
    - Storage: disks, CD, DVD,…
    - Communication: network adapters, Bluetooth,…
  - ✓ Peripherals cannot be directly connected to the system bus. They need specific adapters

Elements of the I/O System



- Adapter (I/O controller – I/O interface)
  - ✓ Electronic device that connects the peripheral to the system bus
    - Bridges the peripheral's own technology with the system bus technology and signals
    - Every peripheral needs its own adapter
  - ✓ Implements support for a programmer's view of the peripheral
    - Translates interface commands into peripheral actions (output)
    - Translates peripheral state into information readable from programs (input)
    - Enables data transfers between peripherals and programs

I/O adapter functions

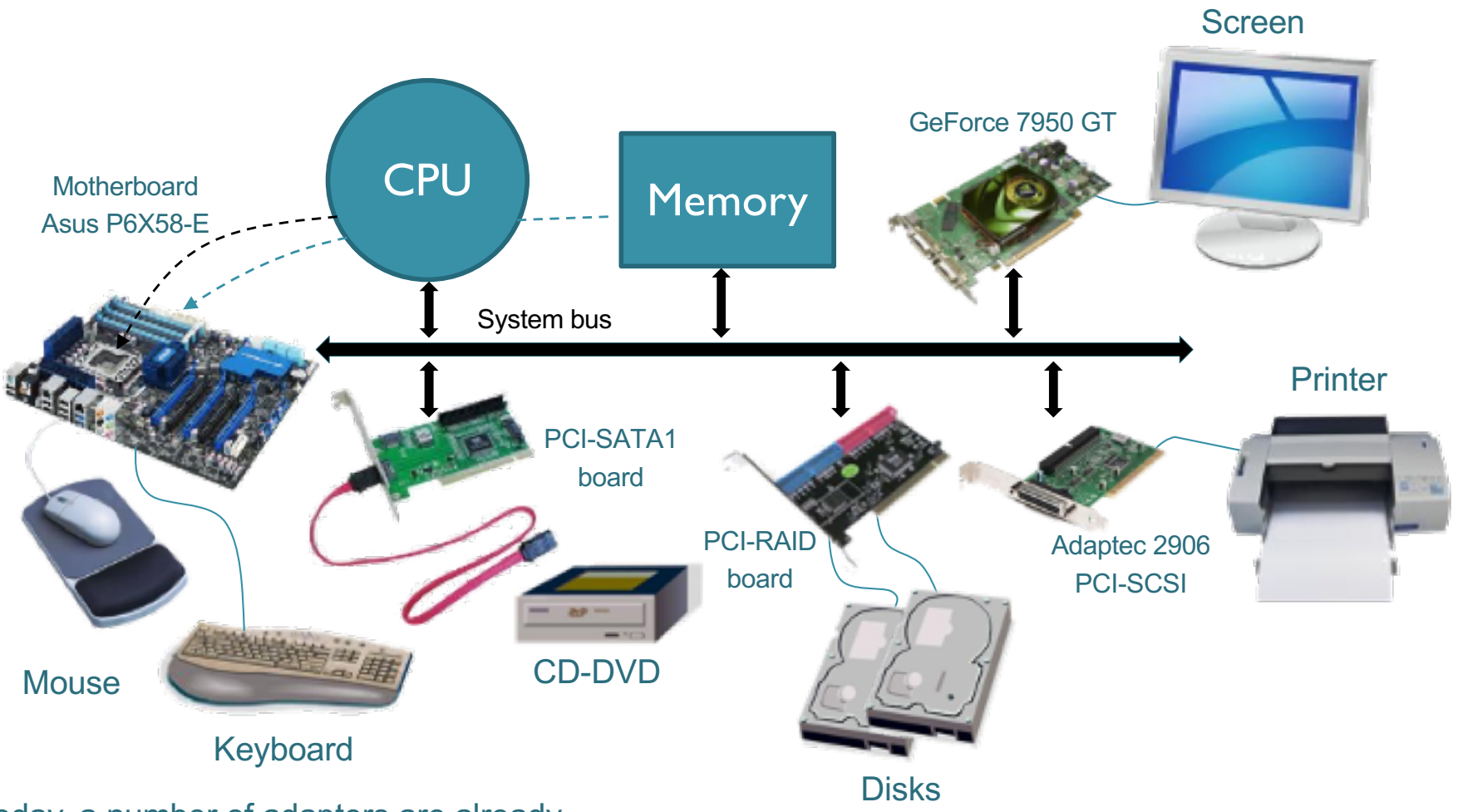
- The main functions of all I/O adapters are:
  - ✓ Connect with the CPU
    - Via the system bus
    - Offer interface registers that can be addressed by programmers

    } Unit 7
  - ✓ Connect with the peripheral
    - External interface: set of connections to the peripheral (cables and connectors)
  - ✓ Timely handling of operations and peripheral events
    - Synchronizing CPU and peripherals: polling vs. interrupts

    } Unit 8
  - ✓ Data transfer
    - Programmed vs. Direct Memory Access (DMA)
    - Data buffering

    } Unit 9
  - ✓ Error control

Examples of adapters



Today, a number of adapters are already integrated in the same *motherboard* as the CPU and memory

All peripherals are completely different!

We need an *abstract view* of peripherals

2 – The concept of I/O adapter

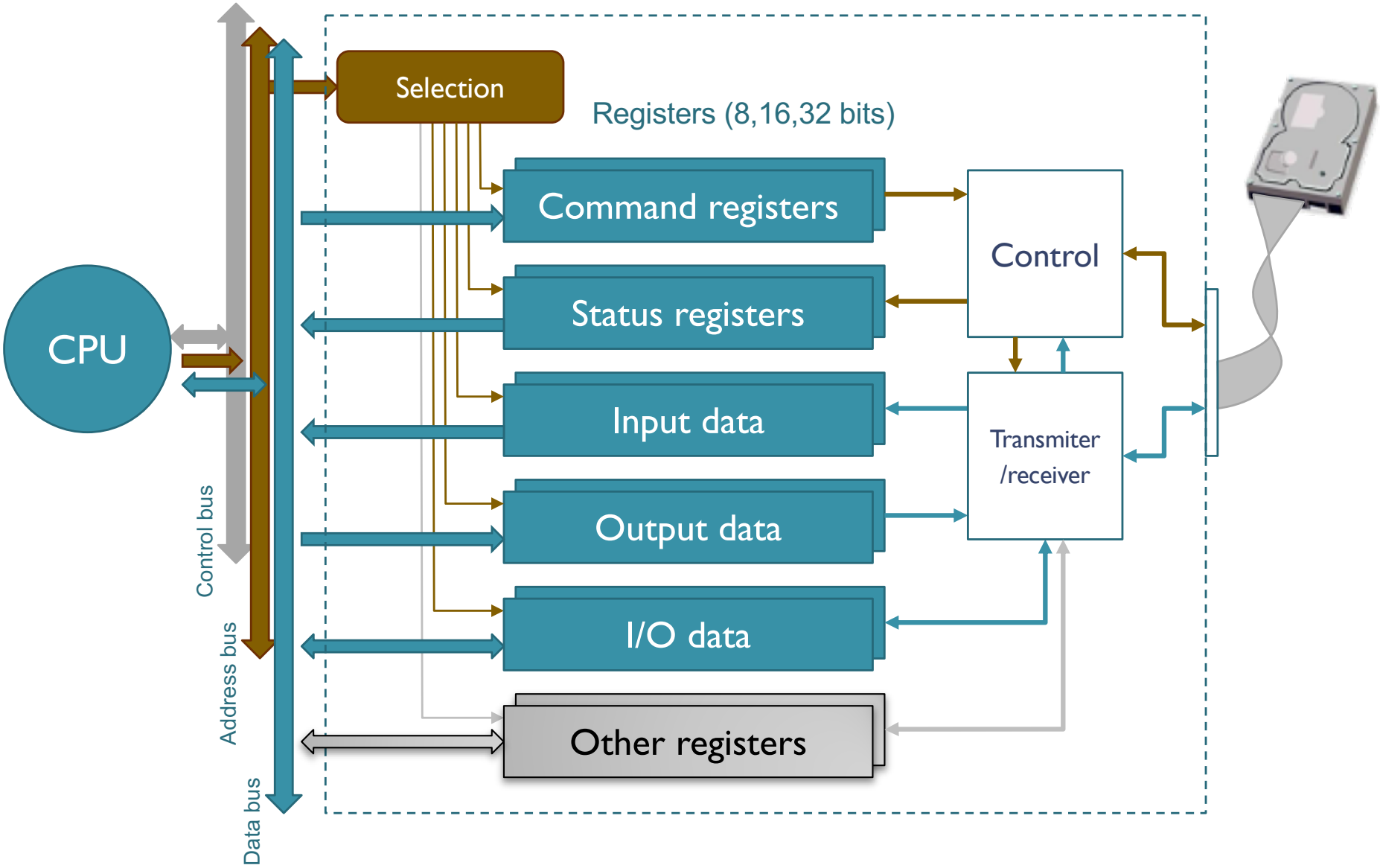
The programmer's view

Concept of I/O adapter

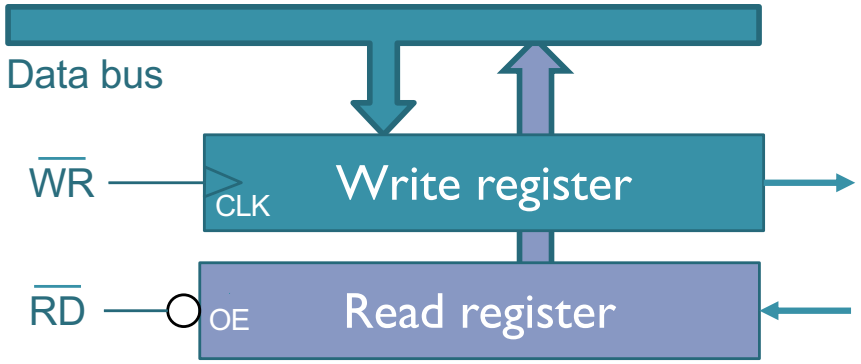
- A peripheral adapter (or I/O interface) is viewed as a heterogeneous set of registers that enable communication between programs and the peripheral

- Different peripherals have different adapters, in terms of number, size, use and meaning of those registers
- Each register in the adapter serves a particular function
  - ✓ Command: (or *control*) to send commands and configure the peripheral
  - ✓ Status: to get information about the current device state (busy, ready,…)
  - ✓ Data: to transfer data, either reading an input from, or writing an output to the peripheral

Simplified diagram of an I/O interface



Interface registers

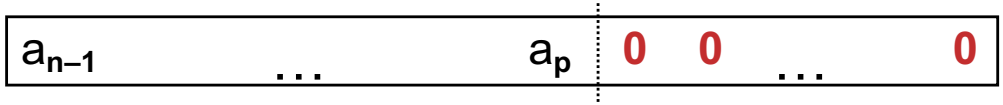


- Sizes:
  - Typically: 8,16 or 32 bits
- Access mode:
  - A register can be read-only, write-only or read/write
  - Writing a read-only register has no effect; reading from a write-only register may yield an undefined or a constant value
  - Two registers may share address if one is read-only an the other write-only
- Contents:
  - A register may be structured or not. If structured, each bit or group of bits has its own specific meaning
  - Some bits may be undefined or reserved by the manufacturer
- Types:
  - Command, status and data, as described before

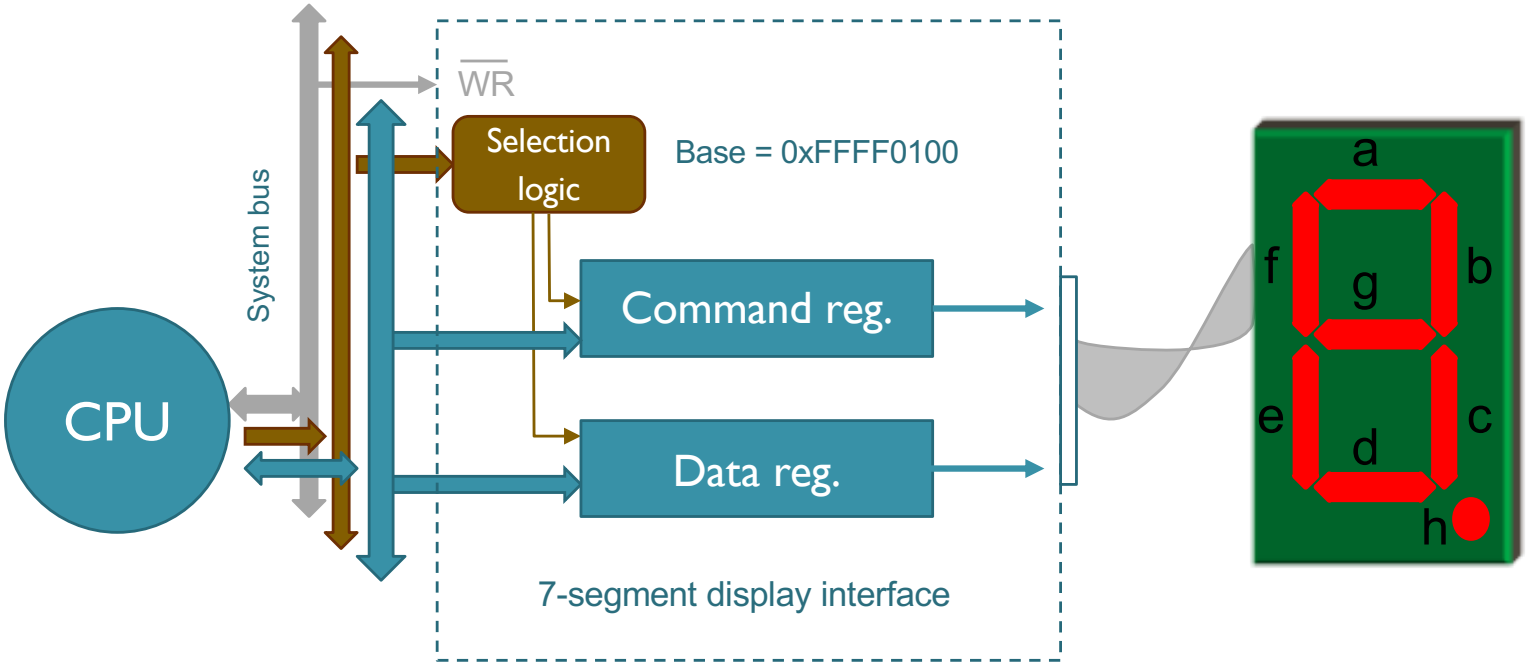
Interface addressing

- ✓ Interface registers have their own addresses in one of two possible I/O addressing schemes (we'll see later):
  - *Memory-mapped I/O* → Most processors, including MIPS
  - *I/O-mapped I/O* → Intel
- ✓ Registers of an interface take a range of addresses:

- ✓ The starting address is called the base address of the interface



Interface example: 7-segment display



Command register

8 bits – write-only – Address: 0xFFFF0100 (BA)

7	6	5	4	3	2	1	0
	Frequency						ON

Activates display and blinking

- ON (bit 0): 1 for setting ON; 0 for setting OFF
- Frequency (bits 6..4): blinking frequency in Hz
- Frequency = 0: continuous

Data register

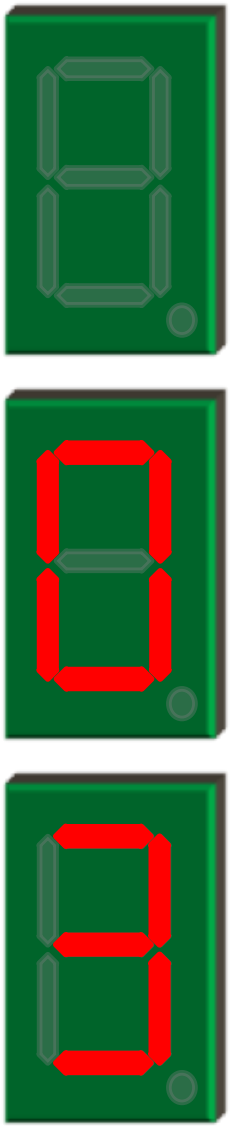
8 bits – write-only – Address: 0xFFFF0104 (BA+4)

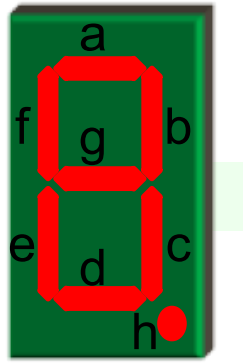
7	6	5	4	3	2	1	0
h	g	f	e	d	c	b	a

bits *a*…*h*: when set to 1, switch the corresponding segment on

Programming the 7-segment display

```
        la $t0,0xFFFF0100
# Switch off display
        sw $zero,0($t0)
        …
# Display continuous zero
        li $t1,0x3F
        sw $t1,4($t0)
        li $t1,1
        sw $t1,0($t0)
        …
# Display "3" at 1 Hz
        li $t1,0x4F
        sw $t1,4($t0)
        li $t1,0x11
        sw $t1,0($t0)
```





0 0 0 1 0 0 0 1

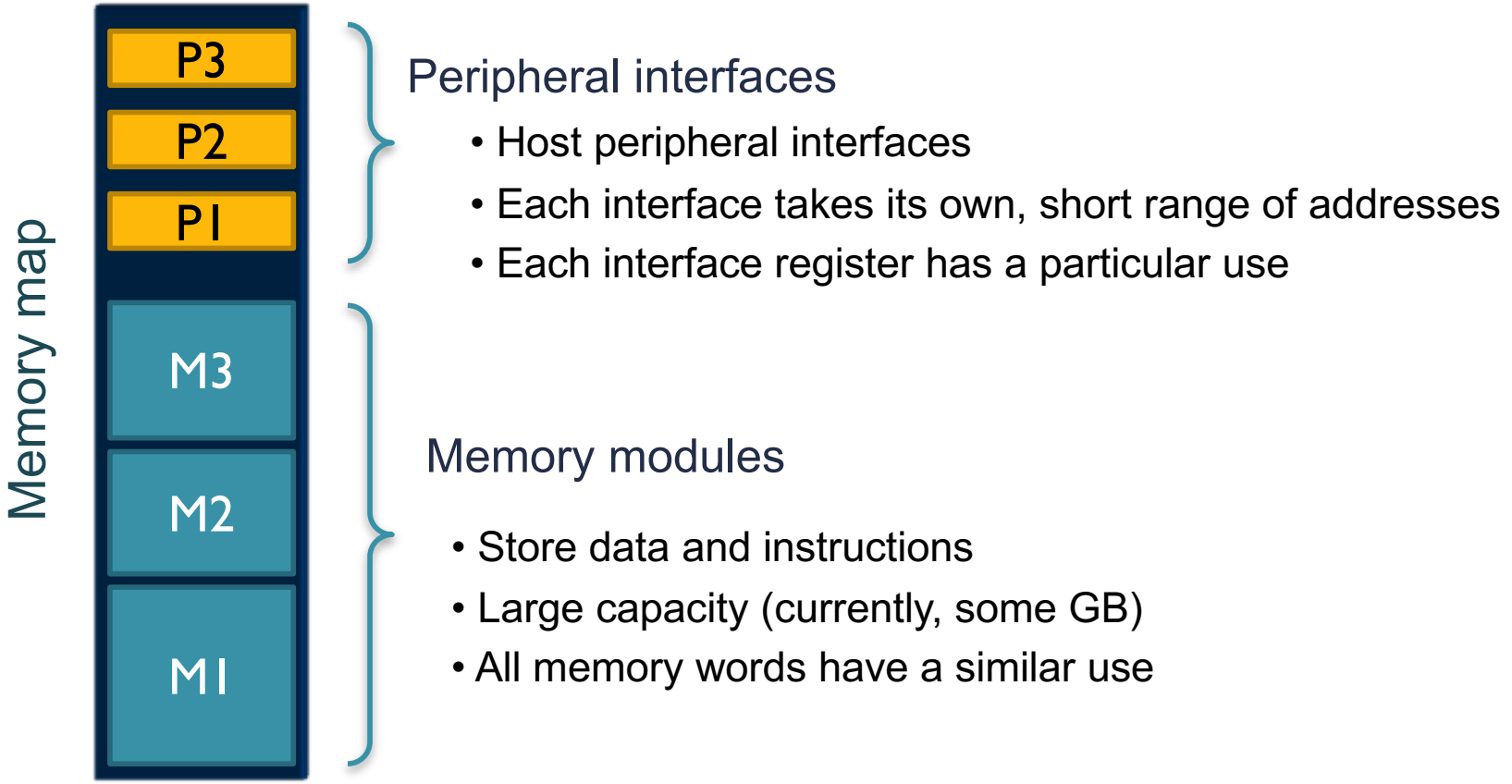
h g f e d c b a

'0' = 0 0 1 1 1 1 1 1

'3' = 0 1 0 0 1 1 1 1

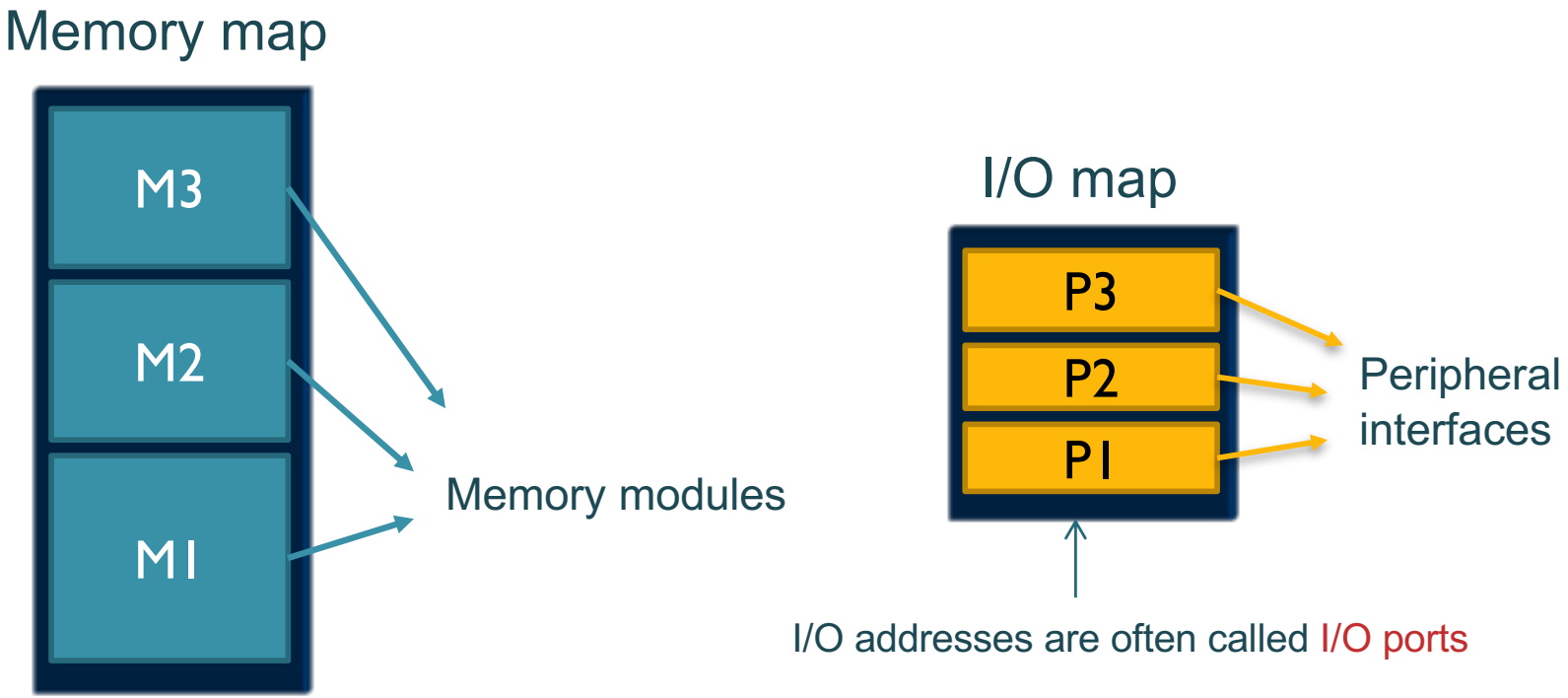
Interface addressing schemes

- Memory-Mapped I/O
  - ✓ There's a single, shared address space for both memory and I/O devices (MIPS model)
  - ✓ Register access via Load and Store instructions



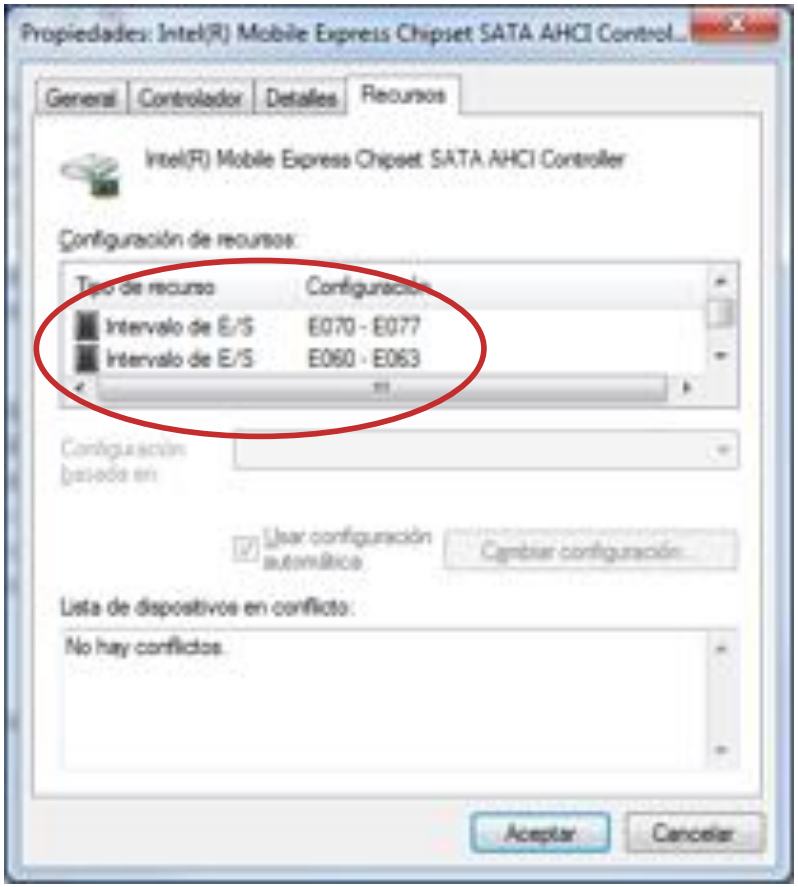
Interface addressing schemes

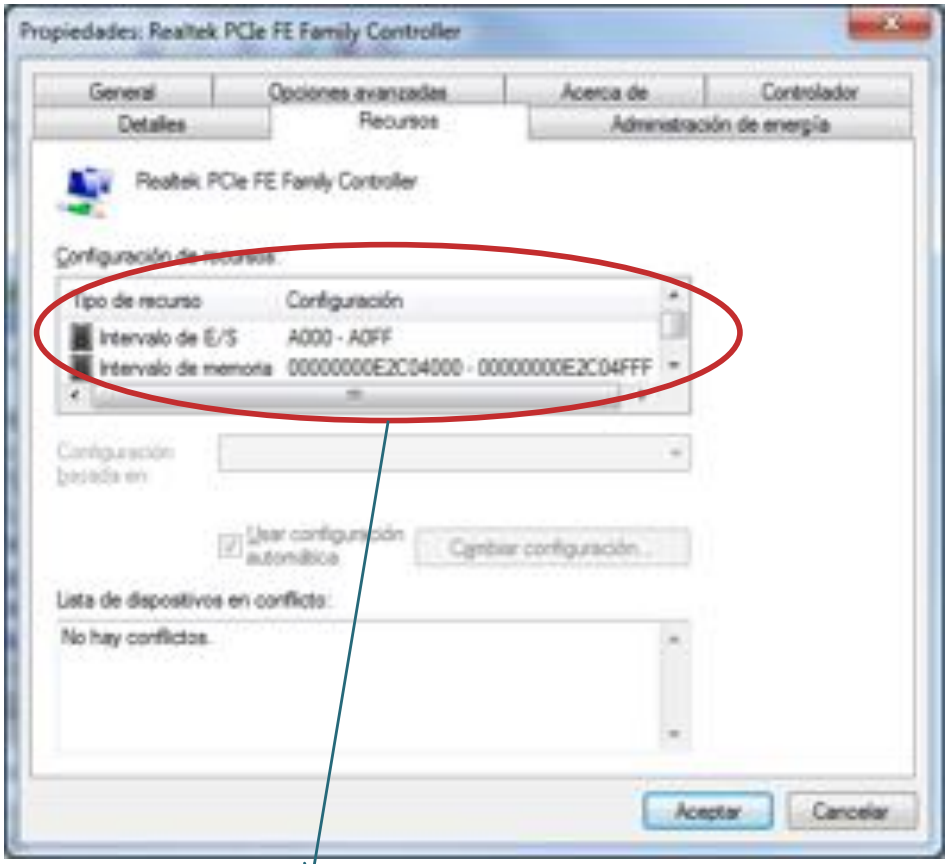
- Input/Output-Mapped I/O
  - ✓ Separate addressing spaces for memory and I/O (Intel model)
    - Load/Store used only for memory access
    - Specialized Input/Output instructions for accessing peripheral interfaces (e.g., Intel's **`in`** and **`out`** instructions)



I/O ranges – example

IDE-SATA adapter (disks)





Network adapter (Realtek)

Uses both maps, I/O and Memory

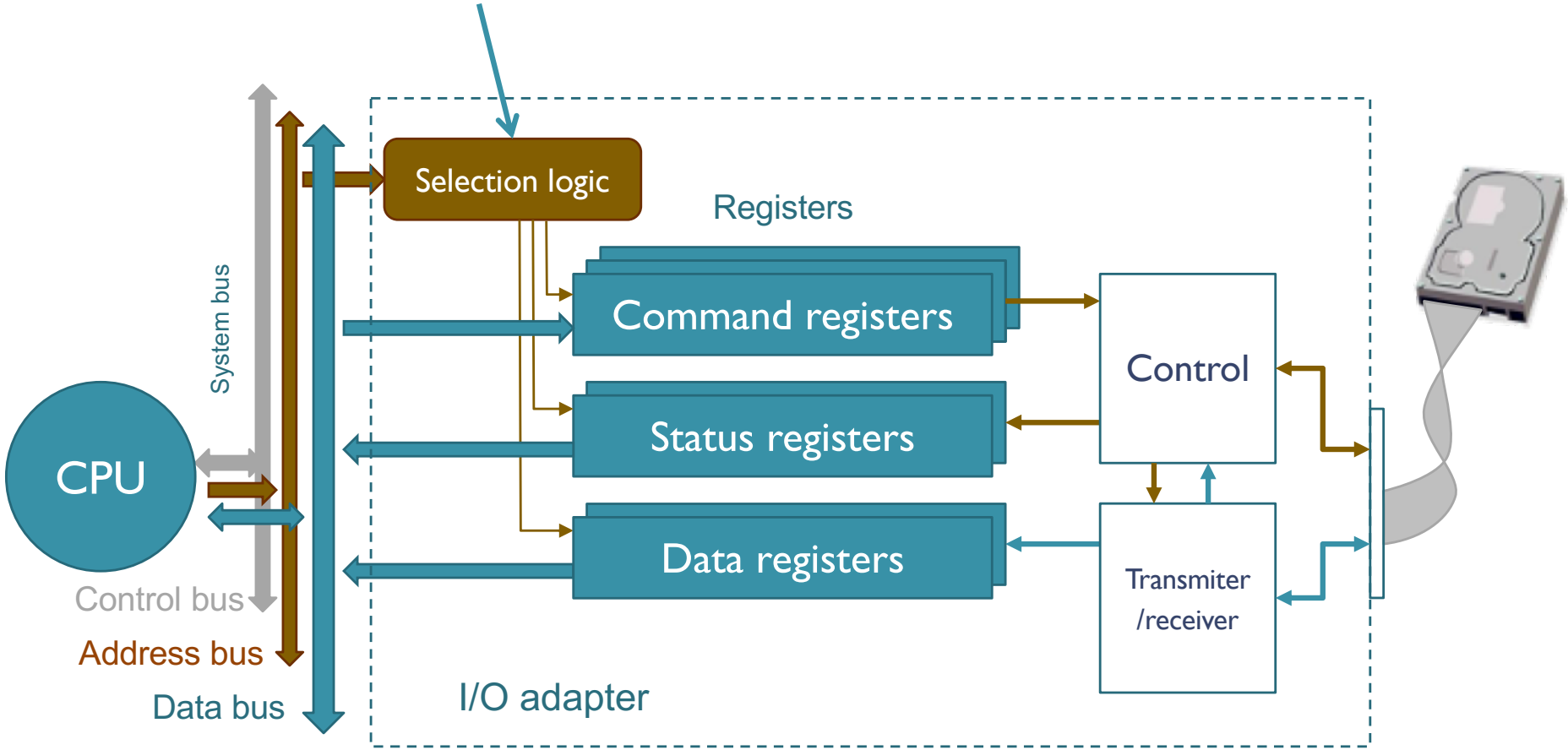
3 – Internal organization of I/O interfaces

Hardware details

Selecting the interface

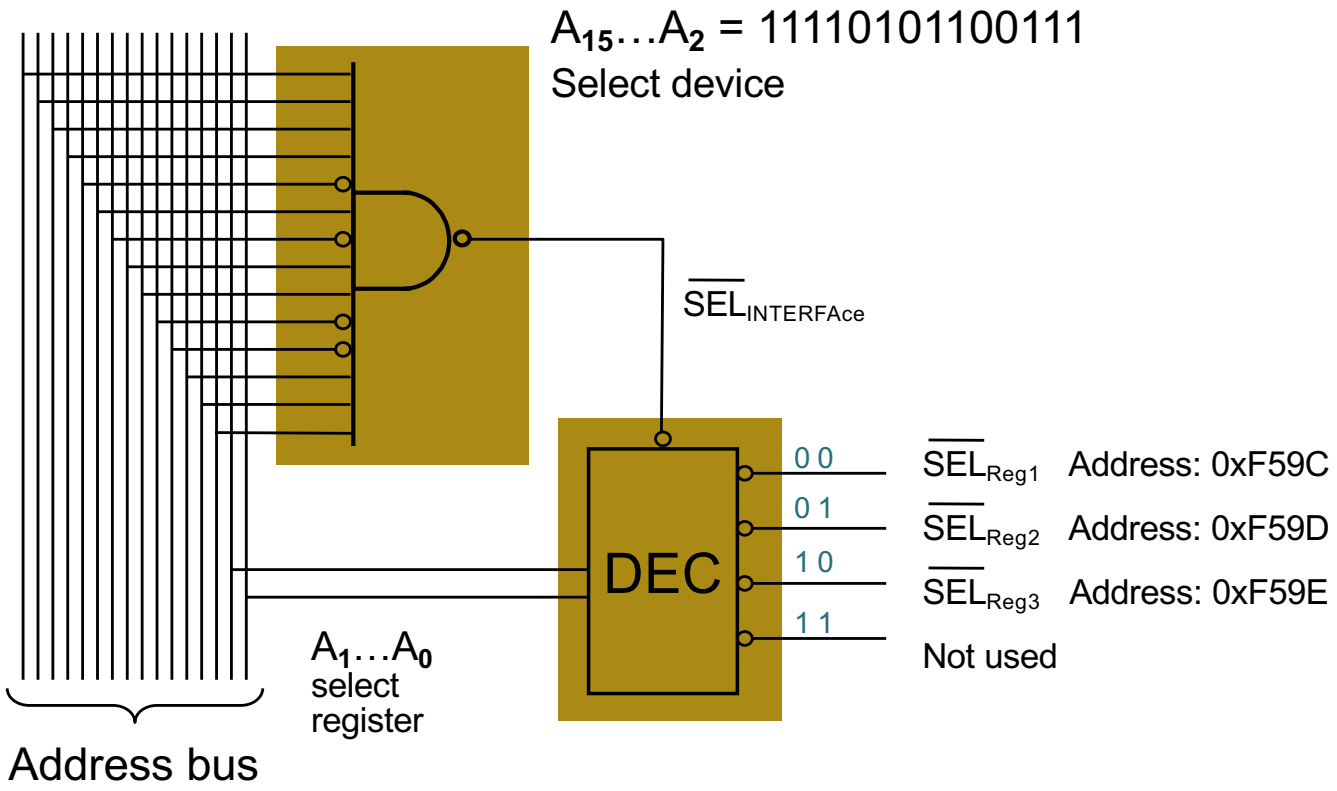
Implements the selection of the interface, enabling access to the registers when an interface address appears on the address bus.

It also implements the addressing scheme (memory- or I/O-mapped)



Selecting the interface

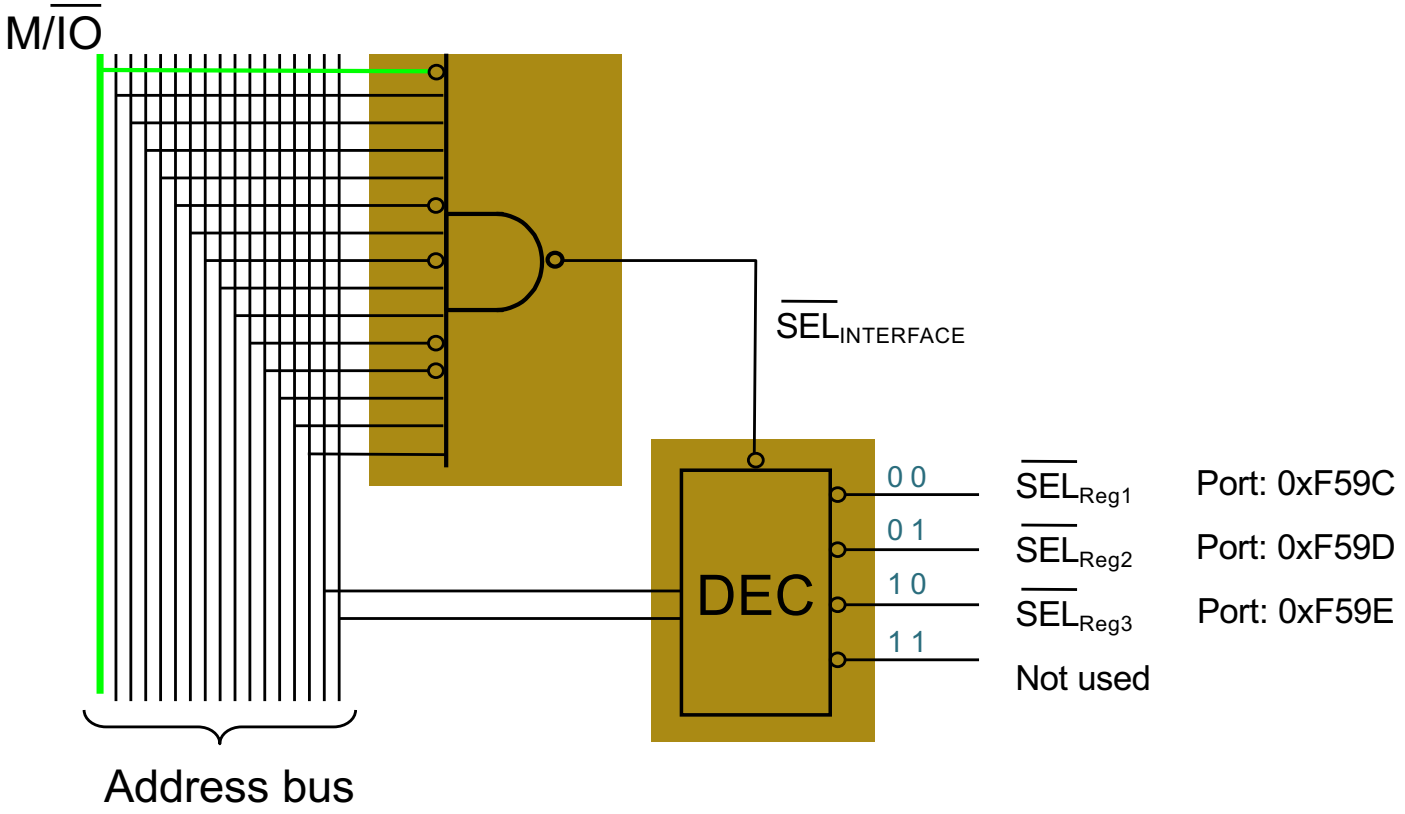
- A selection circuit using a memory-mapped I/O scheme
  - ✓ Example with 16 address lines, 8 data lines
  - ✓ Interface containing 3 registers (n=16, p=2), Base Address = 0xF59C



Selecting the interface

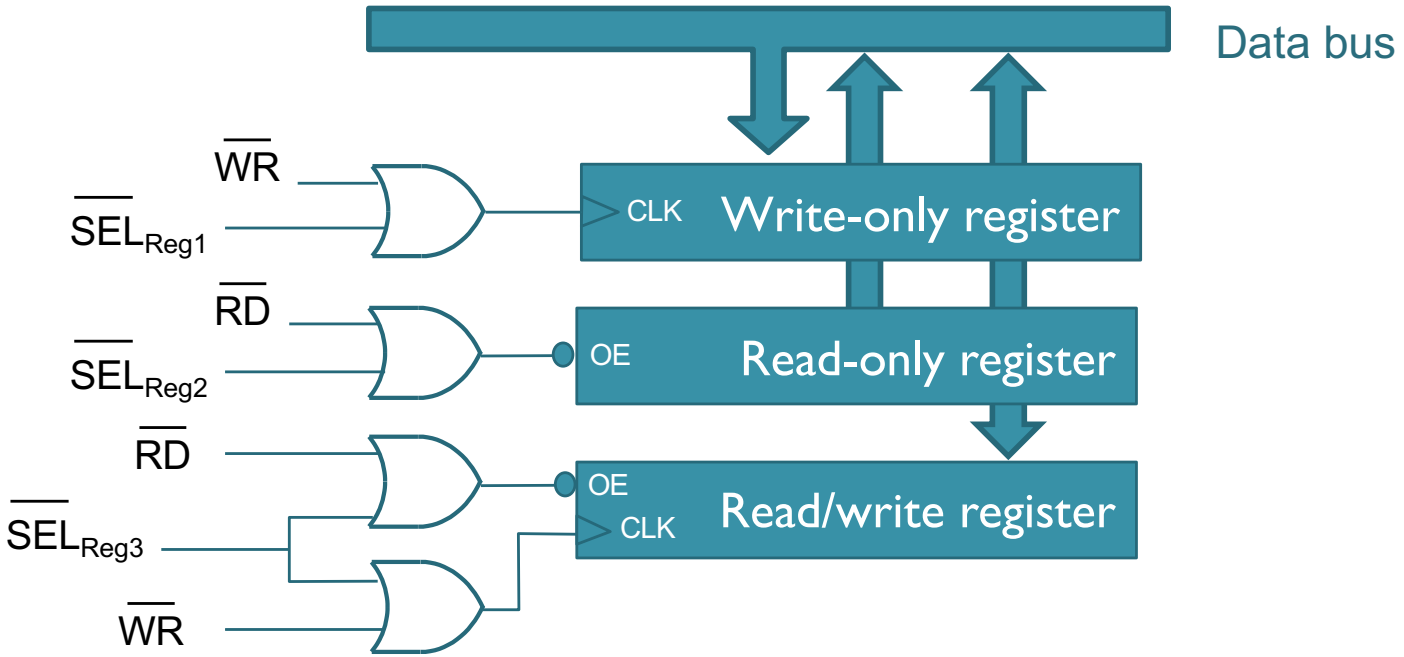
- A selection circuit using an I/O-mapped I/O scheme
  - ✓ A bus line indicates whether the address is for memory or for I/O

M/$\overline{IO}$ = 1: Memory address (`load/store`)
M/$\overline{IO}$ = 0: I/O address (`input/output`)

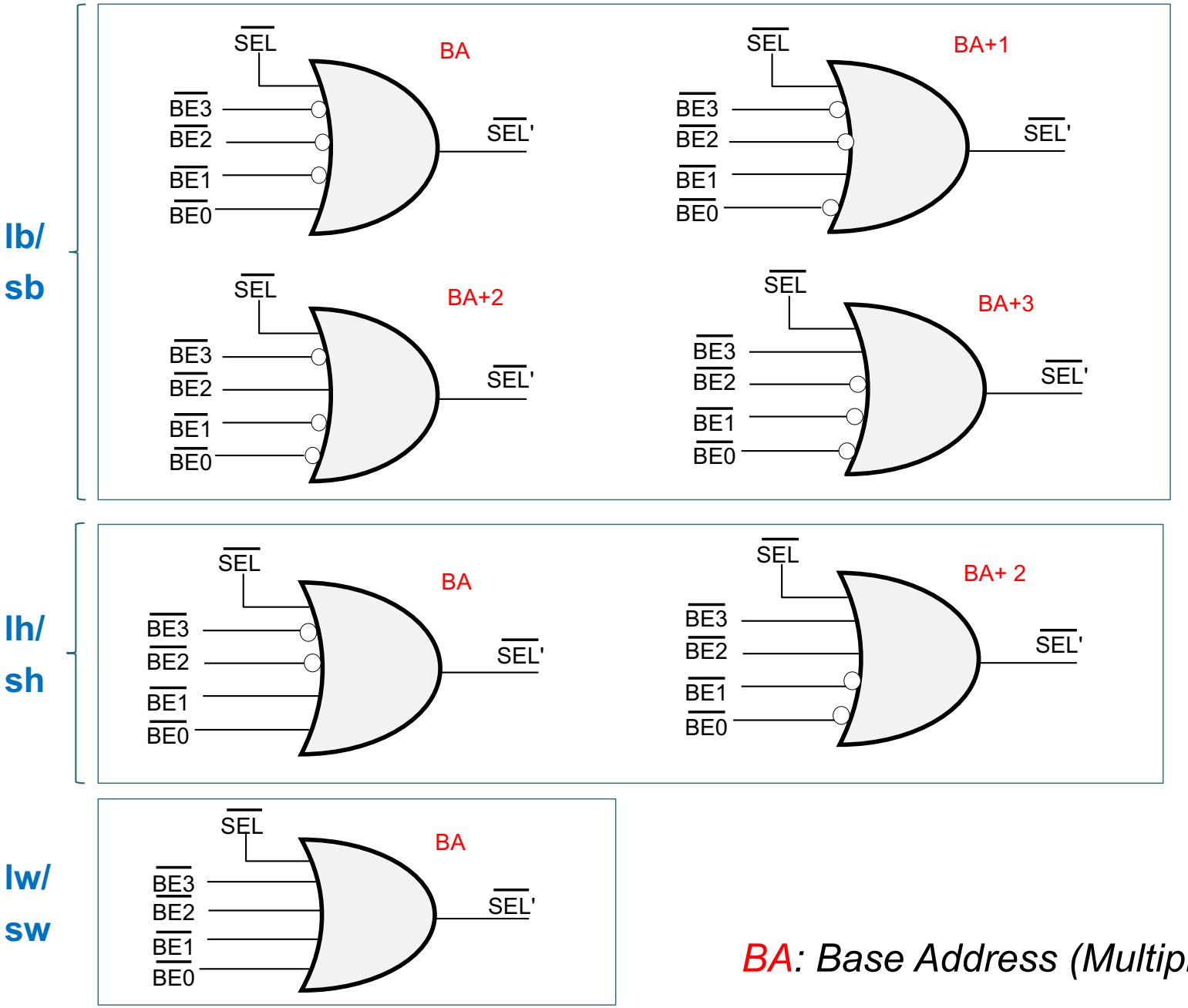


Selecting and operating with registers

- ✓ Write-only registers have a write signal (CLK, clock edge); Read-only registers have a read signal (OE*, Output Enable); Read-write registers have both
- ✓ Register bits are connected to the system data bus
- ✓ Two control-bus lines define the operation: RD* (read) and WR* (write)

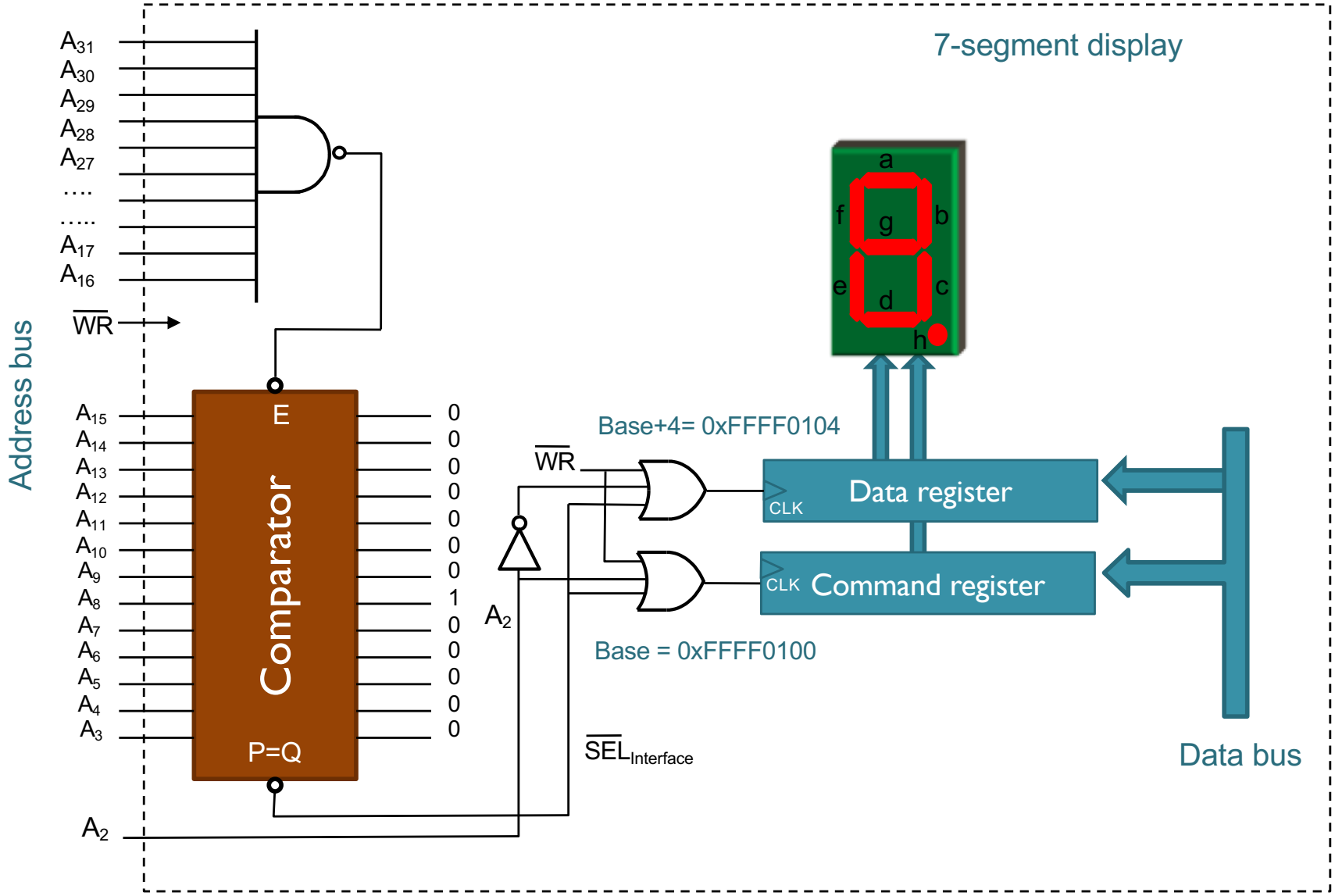


Use of BE* lines for interface selection



BA: Base Address (Multiple of 4)

Example 1: Selecting the 7-segment display

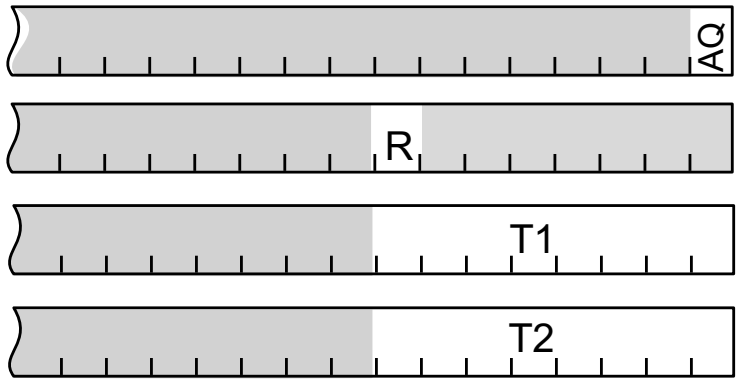


Example 2: Temperature Multi-sensor

- ✓ Two temperature sensors (thermocouples T1 and T2)
- ✓ Base address = 0xFFFF0200
- ✓ 32-bit registers

Two registers share the same address, one read-only and the other write-only

Name	Address	Access	Structure
Command	Base	Write	Bit 0: (AQ) when '1' activates temperature acquisition
Status	Base	Read	Bit 7: (R) when '1' indicates new temperature available auto-reset when temperature is read
Temp1	Base+4	Read	Bits 7…0: (T1) sensor 1 temperature (0…255 ºC)
Temp2	Base+8	Read	Bits 7…0: (T2) sensor 2 temperature (0…255 ºC)



Base - Command

Base - Status

Base+4 - Temp1

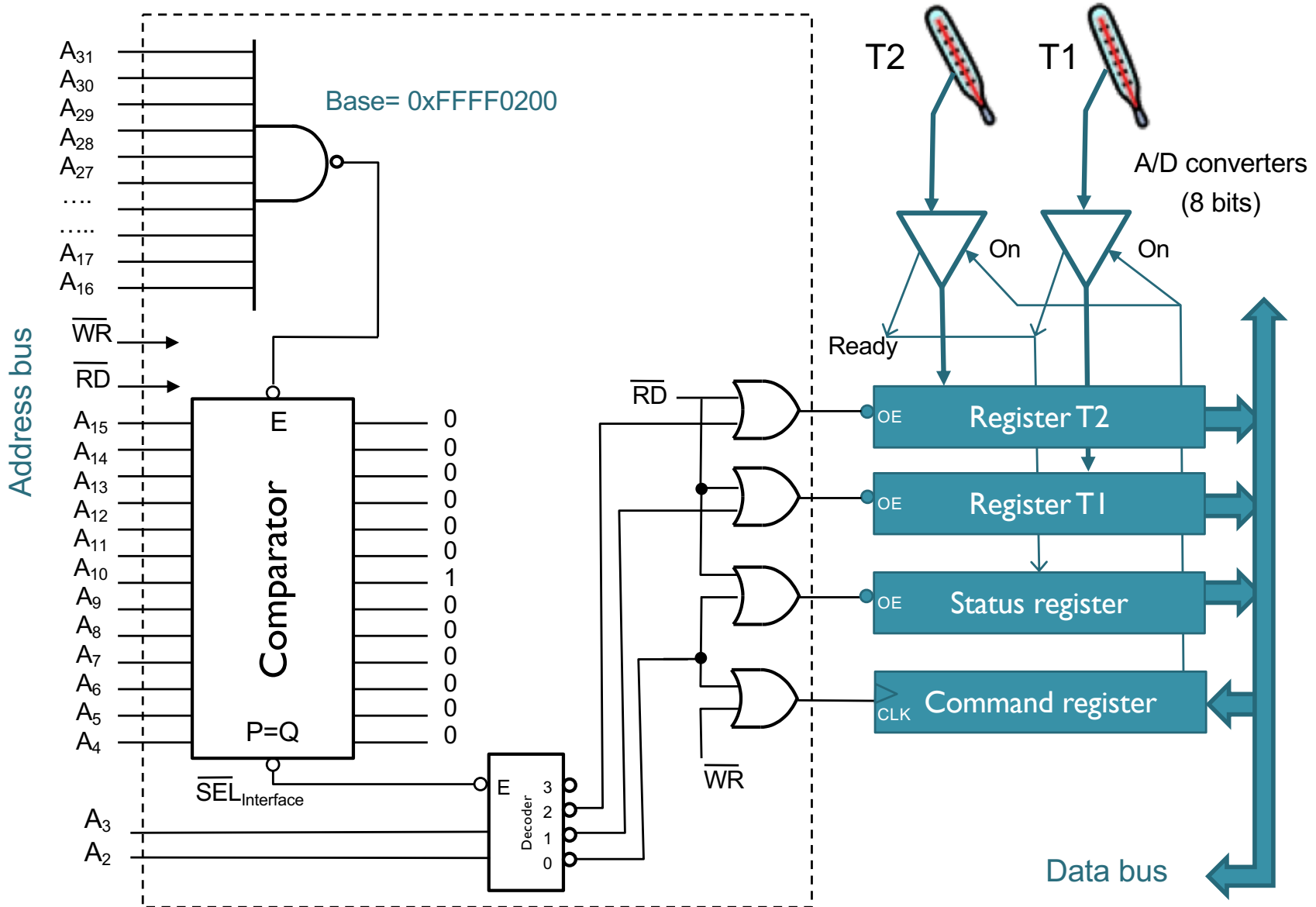
Base+8 - Temp2

Example 2: Temperature Multi-sensor - Exercise

- ✓ Draw the schematic for the selection logic of the temperature multi-sensor
- ✓ Write a MIPS assembly program to read both sensors and write their values to variables Temp1 and Temp2 in memory
  - Note that you must issue an ACQUIRE command and then wait for the interface to signal the end of the acquisition, before you can read the temperature values from the registers

Solution 

Selection circuit for temperature multi-sensor



Programming the temperature multi-sensor

```
        .data 0x00000000
        Temp1       .word 0
        Temp2       .word 0


        .text 0x0040000
            .........
        la $t0,0xFFFF0200 # Base address
       # activate acquisition
        li $t1,0x01
        sw $t1,0($t0)              # Set AQ

       # wait for end of acquisition
wait:
        lw $t1,0($t0)                 <-------- Need for SYNCHRONIZATION
        beq $t1,$zero,wait         # Check for R = 1

       # read temperatures
        lw  $t1,4($t0)             # Read temps and update
        sw  $t1,Temp1              # Temp1 and Temp2
        lw  $t1,8($t0)
        sw  $t1,Temp2
        .........
```