EMBEDDED SYSTEMS DEVELOPMENT

1. Thông tin chung về môn học (General Information)

- **Tên học phần (Course name)**: EMBEDDED SYSTEMS DEVELOPMENT (*XÂY DỰNG CÁC HỆ THỐNG NHÚNG*)
- **Mã học phần (Course code)**: INT1461_CLC
- **Số tín chỉ (Number of credits)**: 3
- **Loại học phần (Course type)**: Compulsory
- **Học phần tiên quyết (Prerequisites)**:
- **Học phần trước (Previous courses)**:
  - Programming techniques
  - Computer architecture
- **Học phần song hành (Parallel courses)**:
  - Lecture room: Projector, microphone and speaker, black board or white board.
  - Laboratory: Computer
- **Giờ tín chỉ đối với các hoạt động (Teaching and Learning hours)**:
  - Lectures: 36h
  - Discussion and Group Activities: 8h
  - Individual reading: 1h

Address of the Faculty/Department in charge of the course:

- Address: Faculty of Information Technology 1 - Posts and Telecommunications Institute of Technology, Km10, Nguyen Trai Street, Ha Dong District, Hanoi.
- Phone number: (024) 33510432

2. Objectives

Knowledge:

- Provide students with knowledge about embedded systems and the design and installation of embedded systems.

Skills:

- After completing the course, students master the knowledge of embedded systems and be able to analyze and design simple embedded systems.

Attitude:

- Ensure the number of hours in class and self-study.

Detailed objectives for each subject content

Content	Level 1	Level 2	Level 3
Chapter 1. Introduction to embedded systems	Understand general concepts of embedded systems	Obtain the basic features and requirements of embedded systems	Evaluate and analyze the requirements and characteristics of embedded systems
Chapter 2: Hardware components of embedded systems and 8051 microprocessors	Understand the definitions of basic components of embedded systems and the 8051 microprocessor	Obtain the technical specifications and incorporate basic components and the 8051 microprocessor	Analyze requirements and technical specifications of components; Joined parts; and 8051 microprocessor
Chapter 3: Software components of embedded systems and 8051 microprocessors	Understand software concepts	Understand the characteristics and requirements of embedded system software and the 8051 microprocessor	Analyze and develop specifications for embedded system software and the 8051 microprocessor
Chapter 4: Design and implementation of embedded systems	Understand the steps of system design and implementation	Obtain the specification requirements in system construction and installation	Analyze requirements and build embedded system hardware and software

3. Nội dung chi tiết học phần (Outlines)

1. Prerequisites

Chapter 1: Introduction to embedded systems

1.1. What is an embedded system?
1.2. Characteristics of embedded systems
1.3. Embedded system requirements
1.4. A general model of embedded systems
1.5. Classification of embedded systems
1.6. Embedded System design process
1.7. Software for Embedded Systems Overview

Chapter 2: Hardware components of embedded systems

2.1. Embedded system boards, circuit boards, and processors
2.2. Memory
2.3. Bus system

2.4. Input and output modules
2.5. 8051 microprocessor: Architecture, memory, and practical interface

Chapter 3: Software components of embedded systems
3.1. Device drivers
3.2. Real-time operating system
3.3. Middleware and application software
3.4. 8051 Structure and programming

Chapter 4: Design and implementation of embedded systems
4.1. System design
4.1.1. Determine requirements and Specific description
4.1.2. Hardware-software partitioning
4.1.3. System design
4.2. Install and test embedded systems
4.2.1. Hardware installation
4.2.2. Install software
4.2.3. System testing
4.3. Complete product design

4. Học liệu (Textbooks)
4.1. Học liệu bắt buộc (Required Textbooks)

[1] Huỳnh Thúc Cước, *Bài giảng Xây dựng hệ thống nhúng*, Học viện Công nghệ BC-VT, 2010.
[2] David, Liam. "Embedded Systems: Architecture, Programming, and Design" by Raj Kamal." *PAKISTAN JOURNAL OF LINGUISTICS* 5.1 (2023): 108-113.

4.2 **Học liệu tham khảo (Optional Textbooks)**

[2] Tammy Noergaard, *Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers*, Newnes, 2005.
[3] Steve Heath, *Embedded Systems Design*, Second Edition, Newnes, 2002.
[4] Daniel D. Gajski, Frank Vahid, Sanjiv Narayan and Jie Gong, *Specification and Design of Embedded Systems*, Prentice Hall, 1994.
[5] Frank Vahid and Tony Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, John Wiley & Sons, 2002.
[6] Prabhat Mishra, University of Florida, *Introduction to Embedded Systems*, http://www.cise.ufl.edu/~prabhat/Teaching/cis6930-f04/systems.html.
[7] CMP Media LLC, http://www.embedded.com/.

5. Phương pháp, hình thức kiểm tra – đánh giá kết quả học tập học phần (Grading Policy)

Grading method	Percentage	Group/Individual
- Attendance	10 %	Individual

- Exercises	20%	Individual
- Mid-term projects/exams	10%	Individual
- Final examination	60%	Individual

Content and criteria for evaluating assignment types:

Examinations	Evaluation criteria
- Assignment: Essay Content: Analyze requirements and propose to build applications using embedded systems	- Master the basic knowledge to present the essay - Answer the question correctly
- Examinations	-Master subject knowledge -Answer questions and exercises correctly

Duyệt *(Ký và ghi rõ họ tên)*	**Trưởng Bộ môn** **(Head of Department)** *(Ký và ghi rõ họ tên)*	**Giảng viên biên soạn** **(Lecturer)** *(Ký và ghi rõ họ tên)*
Ngô Xuân Bách	**Ngô Xuân Bách**	**Đỗ Tiến Dũng**

POST AND TELECOMMUICATIONS INSTITUTE OF TECHNOLOGY

Embedded systems

Faculty: **Computer science – IT1**

COMPONENT POINTS

- Attendance: 10%
- Exercices: 10%
- Project: 10%
- Final exam: 60%

CONTENT

- Chapter 1: Introduction
- Chapter 2: Hardware components of embedded systems Chapter
- Chapter 3: Software components of embedded systems Chapter
- Chapter 4: Embedded system design and installation

Chapter 1: INTRODUCTION

DEFINITIONS OF EMBEDDED SYSTEMS

1. Function: organize or carry out one or more tasks based on a set of rules, plans, programs.

2. .Combine and arrange functional units to perform tasks together.

3. For example: clock, washing machine...

DEFINITIONS OF EMBEDDED SYSTEMS

1. A system where hardware is integrated with software to execute a certain application, or part of an application in a larger system.

2. A system that has software with certain functions embedded in the computer.

3. Is a hardware system for a certain product or application. It can be a stand-alone system or part of a larger system. Software is embedded in ROM or flash.

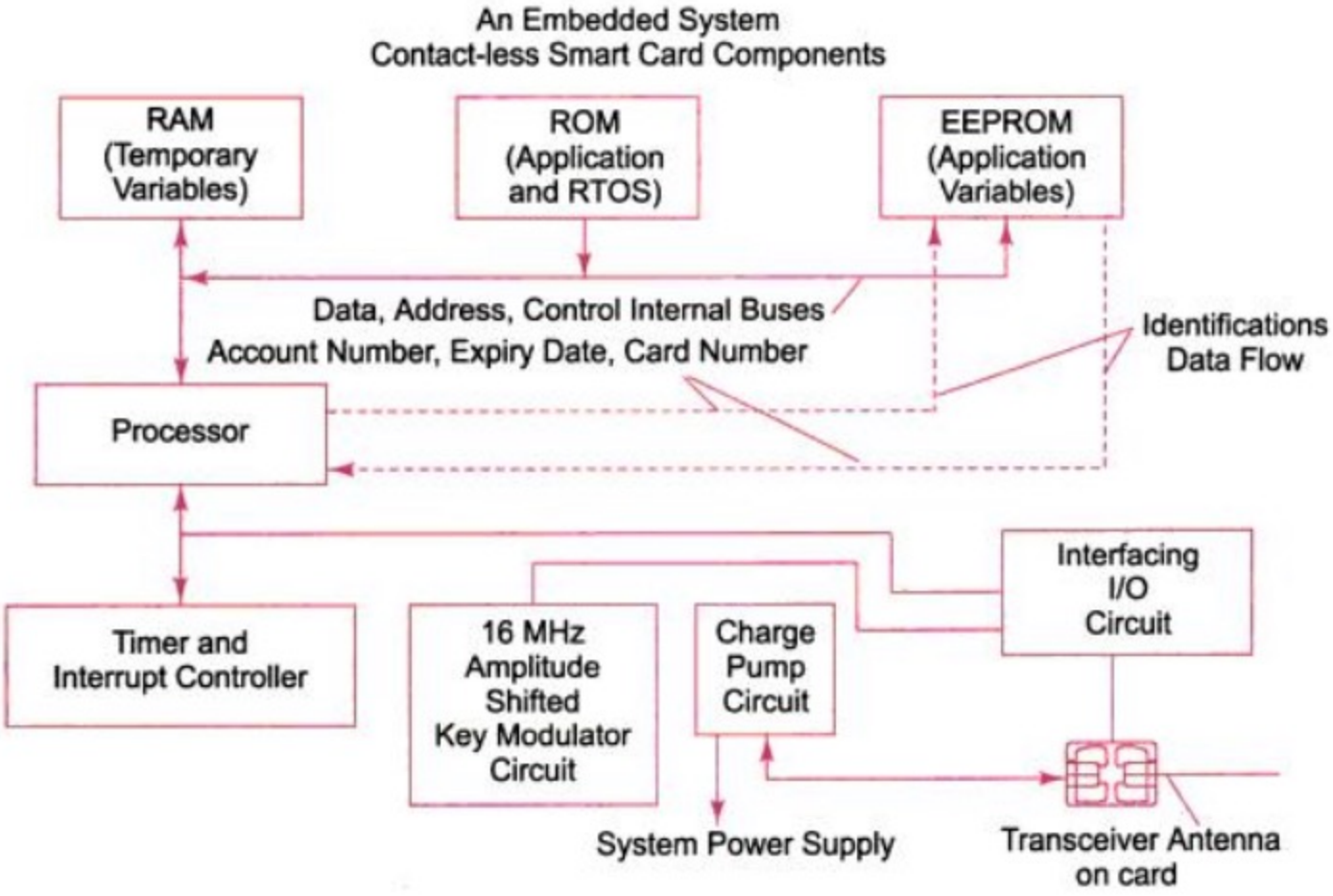
MAIN PARTS

1. Hardware (ROM, RAM, EEPROM, I/0, processor, Power supply …).

2. Software

3. Real time Operating System (RTOS).

EXAMPLES


An Embedded System
Contact-less Smart Card Components
RAM (Temporary Variables)
ROM (Application and RTOS)
EEPROM (Application Variables)
Data, Address, Control Internal Buses
Account Number, Expiry Date, Card Number
Identifications Data Flow
Processor
Timer and Interrupt Controller
16 MHz Amplitude Shifted Key Modulator Circuit
Charge Pump Circuit
Interfacing I/O Circuit
System Power Supply
Transceiver Antenna on card

EXAMPLES

Telecom

- Mobile Computing
- Mobile Access

(a)

Smart Cards

- Banking
- Security

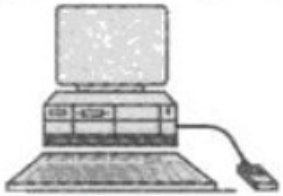
(b)

Missiles and Satellites

- Defence
- Aerospace
- Communication

(c)

Computer Networking Systems and Peripherals



- Networking Systems
- Image processing
- Printers
- Networks Cards
- Monitors and Displays

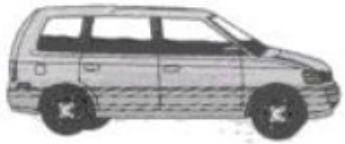
(d)

Digital Consumer Electronics

- DVDs
- Set top boxes
- High definition TVs
- Digital cameras

(e)

Automotive



- Motor Control System
- Cruise Control
- Engine/Body Safety
- Robotics in Assembly Line
- Car Entertainment
- Car Multimedia

(f)

Features of complicated Embedded Systems

1. Dedicated functions
2. Dedicated complex algorithms
3. GUI and user interface
4. Real time operations
5. Multi- rate operations: audio, video …

System Requirements

1. Memory

2. Processing speed

3. Low energy consumption

Design Requirements

1. Performance
2. Power
3. Size
4. Design cost
5. Packaging cost

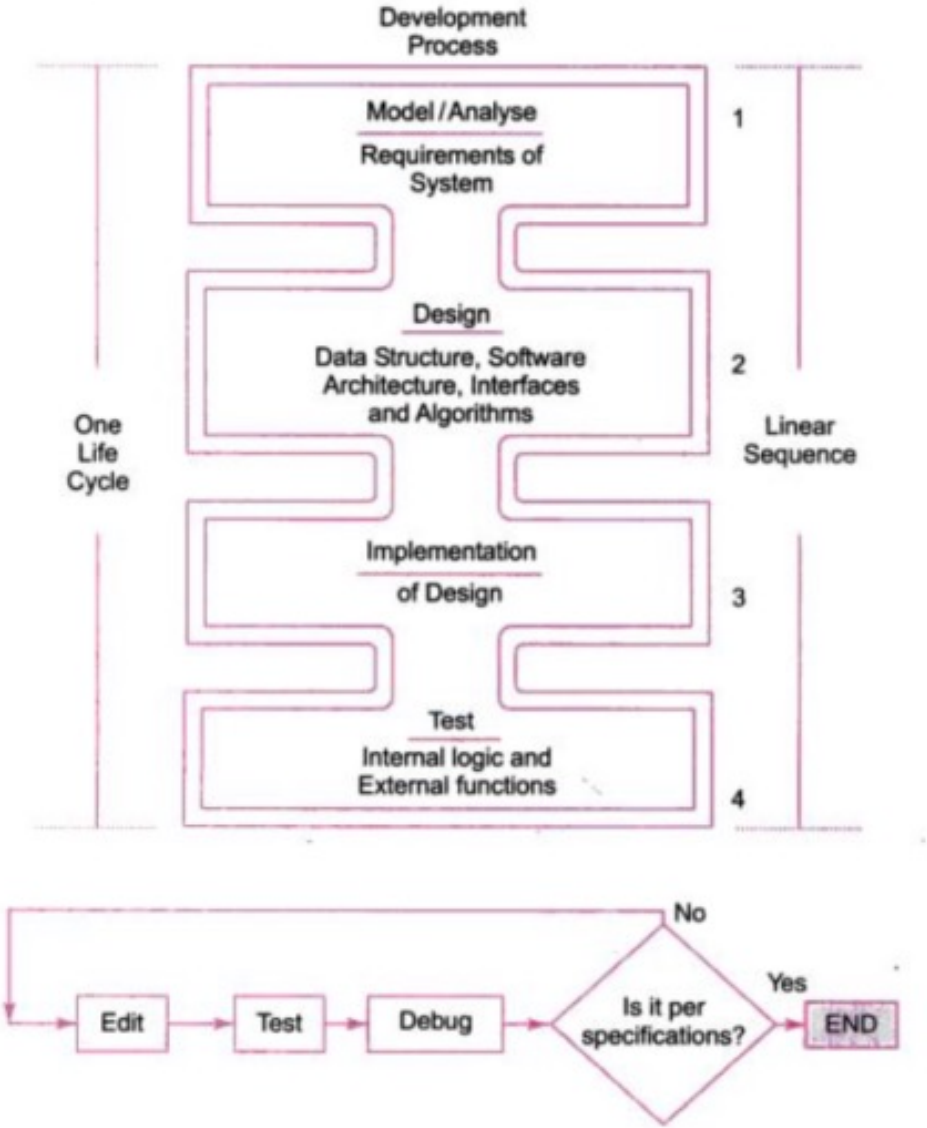
Design Requirements

Design Metrics	*Description*
Power Dissipation	For many systems, particularly battery operated systems, such as mobile phone or digital camera the power consumed by the system is an important feature. The battery needs to be recharged less frequently if power dissipation is small.
Performance	Instructions execution time in the system measures the performance. Smaller execution time means higher performance. For example, a mobile phone, voice signals processed between antenna and speaker in 0.1s shows phone performance. Consider another. For example, a digital camera, shooting a 4M pixel still image in 0.5s shows the camera performance.
Process deadlines	There are number of processes in the system, for example, keypad input processing, graphic display refresh, audio signals processing and video signals processing. These have deadlines within which each of them may be required to finish computations and give results.
User interfaces	These include keypad GUIs and VUIs.
Size	Size of the system is measured in terms of (i) physical space required, (ii) RAM in kB and internal flash memory requirements in MB or GB for running the software and for data storage and (iii) number of million logic gates in the hardware.
Engineering cost	Initial cost of developing, debugging and testing the hardware and software is called engineering cost and is a one-time non-recurring cost.
Manufacturing cost	Cost of manufacturing each unit.
Flexibility	Flexibility in design enables, without any significant engineering cost, development of different versions of a product and advanced versions later on. For example, software enhancement by adding extra functions necessitated by changing environment and software re-engineering.
Prototype development time	Time taken in days or months for developing the prototype and in-house testing for system functionalities. It includes engineering time and making the prototype time.
Time-to-market	Time taken in days or months after prototype development to put a product for users and consumers.
System and user safety	System safety in terms of accidental fall from hand or table, theft (e.g., a phone locking ability and tracing ability) and in terms of user safety when using a product (for example, automobile brake or engine).
Maintenance	Maintenance means changeability and additions to the system; for example, adding or updating software, data and hardware. Example of software maintenance is additional service or functionality software. Example of data maintenance is additional ring-tones, wallpapers, video-clips in mobile phone or extending card expiry date in case of smart card. Example of hardware maintenance is additional memory or changing the memory stick in mobile computer and digital camera.

Design Process

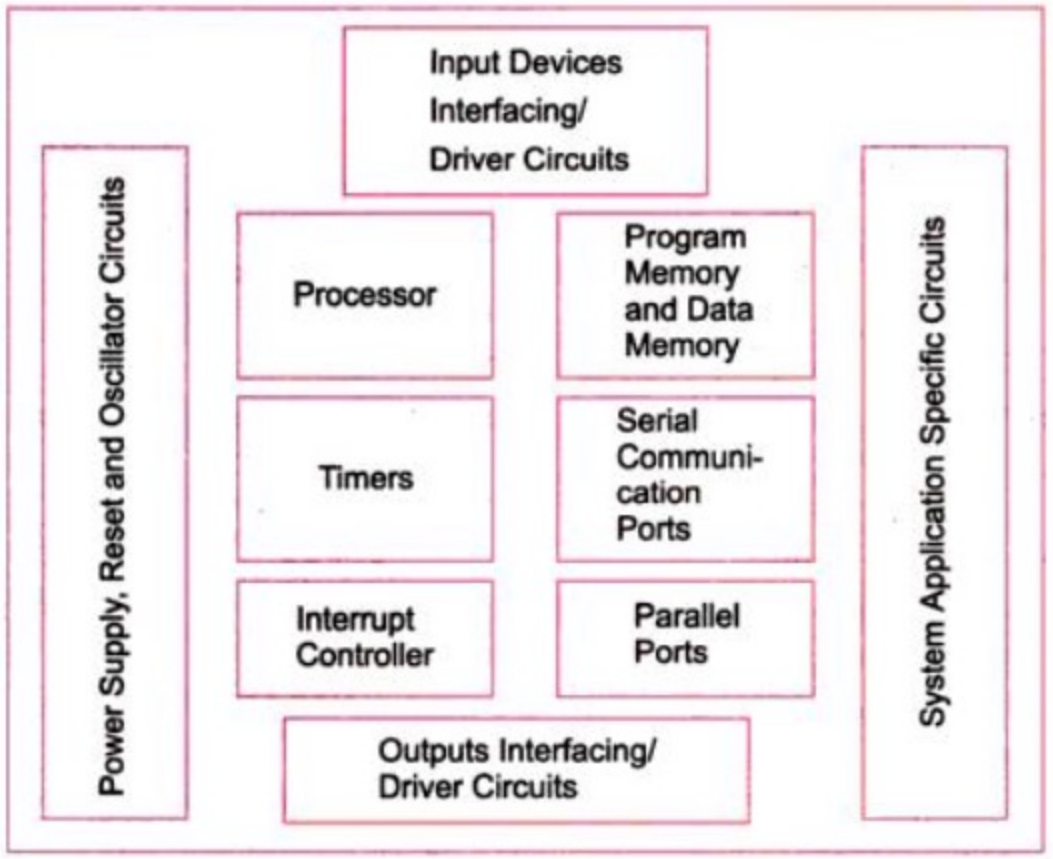
1. Identify objectives/requirements of system
2. Identify the structure of hardware, software
3. The additional functions
4. Researching the similar design
5. Divide into modules (hardware, software)
6. Mapping
7. Design system interface
8. Update system

Design Process



Main parts of Embedded Systems

1. Processors
2. Basic circuit blocks
   - Power Source
   - Clock
   - Reset Circuit
   - Memory
   - IO ports
   - DAC/ADC
   - LED, LCD



Processor of Embedded Systems

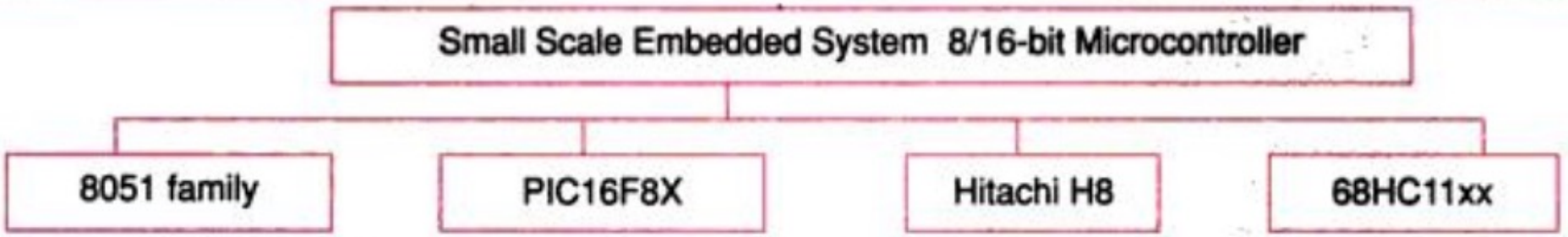
1. General purpose processor: Intel 80x86, Sparc, or Motorola 68HCxxx

2. ASIP (Application Specific Instruction-Set Processor): DSP, media, IO, network

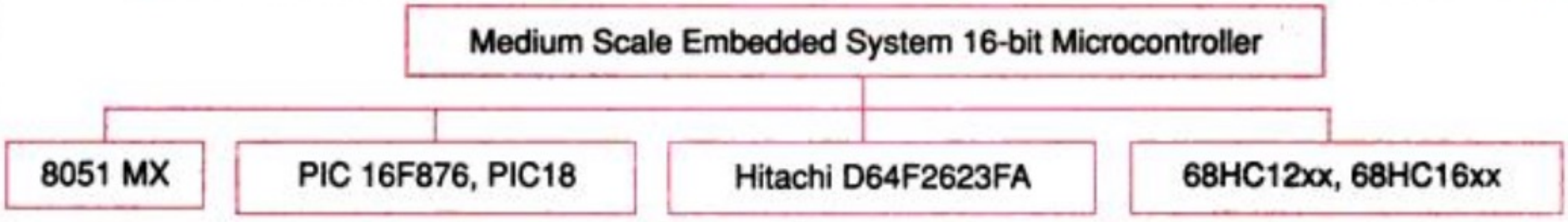
3. Single Purpose processor

4. Multiprocessor

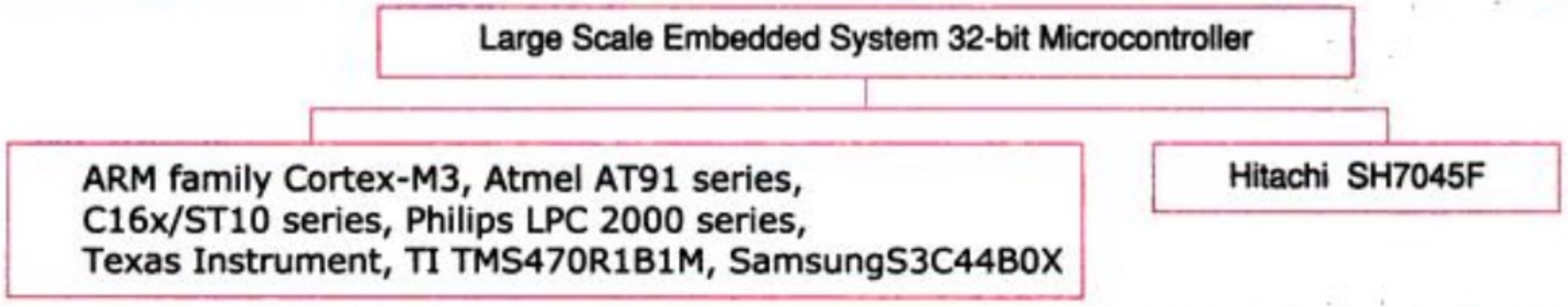
1. Small Scale Embedded System



2. Medium Scale Embedded Systems



3. Large Scale Embedded System



Single Purpose processor

- Floating point Coprocessor
- Pixel coprocessor
- Image codec in digital camera
- Graphic processor
- Speech processor
- Adaptive filtering processor
- Encryption engine
- Decryption engine
- Communication protocol stack processor

Multicore processors

- Multiprocessor system for Real time performance in a video-conference system.
- Embedded firewall cum router
- High-end cell phone

Software Requirements for Embedded Systems

- Application Programing: C++, Java.
- Device driver Programing: C, C++
- Android, web (basic) Programing.
- Script: Perl, Python, Shell script in Linux
- Good data structure and algorithms

POST AND TELECOMMUICATIONS INSTITUTE OF TECHNOLOGY

CHAPTER 2

Hardware of Embedded Systems and 8051

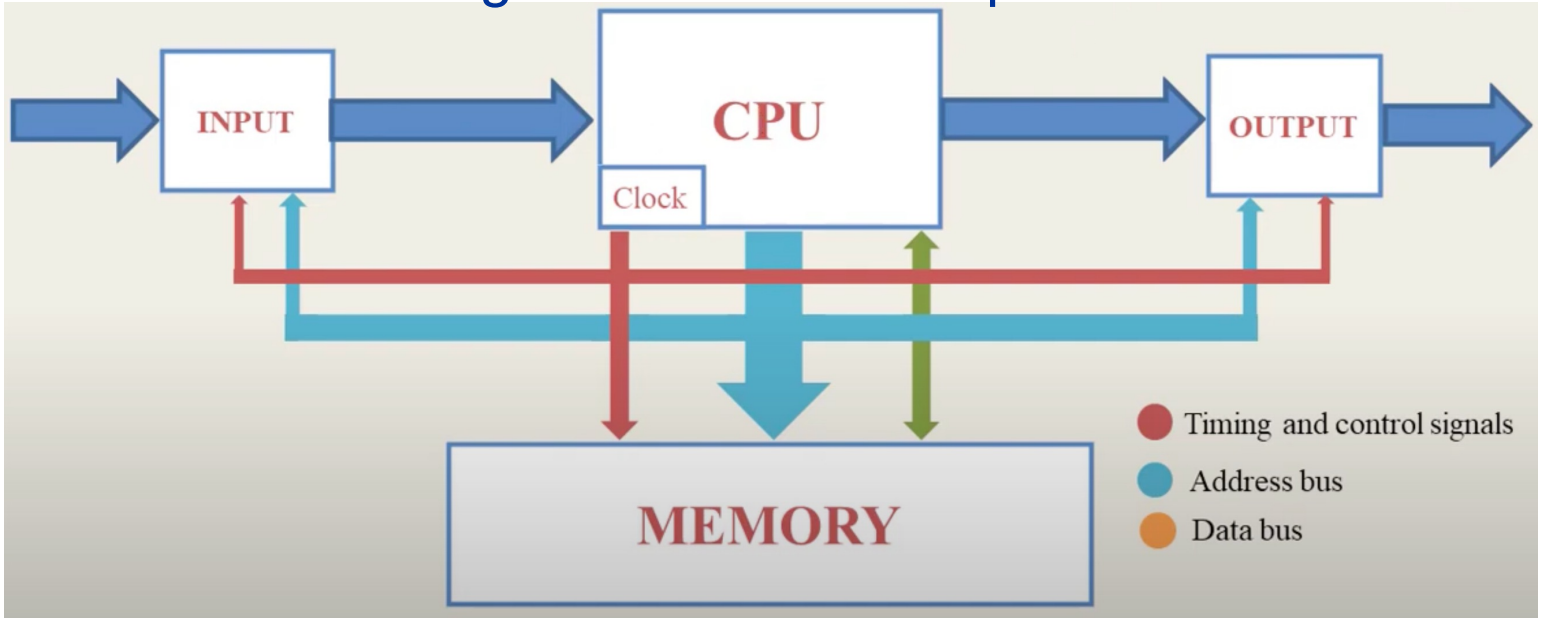
Faculty: **Computer science – IT1**

CONTENT

1. Central processing unit - CPU
2. Memory and memory design
3. Pairing with peripheral devices
4. 8051 microprocessor: Architecture, memory and practical interface

❖ Hardware Overview of Embedded Systems

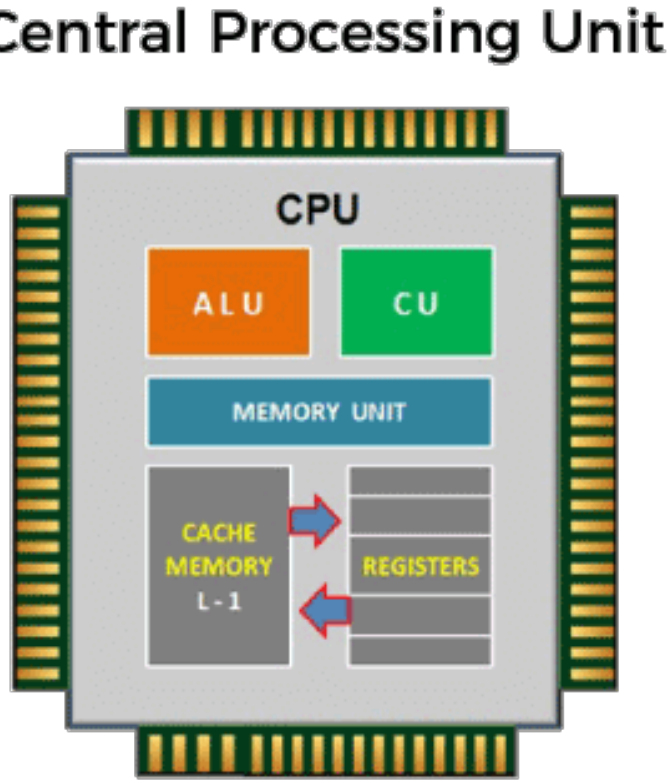
- CPU (Central Processing Unit): Executes instructions and processes data
- Memory: Program and data memory
- Peripheral: Input / output devices
- Bus: transmits signals between components



2.1: CPU

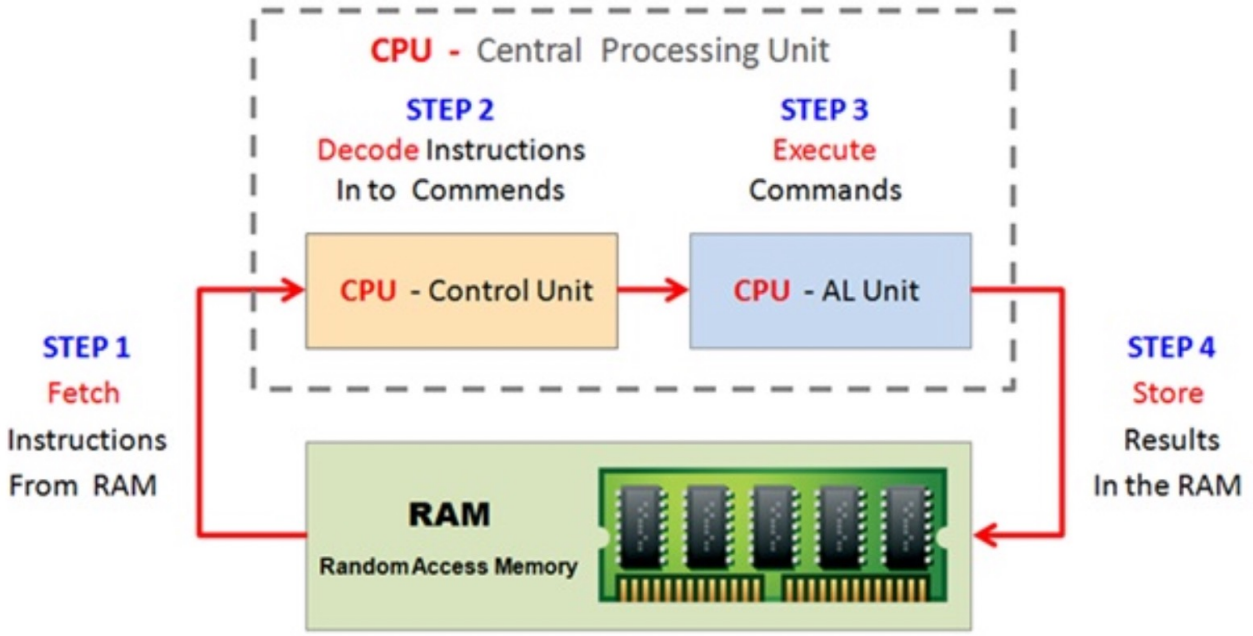
CPU

- Basic CPU block diagram includes:
  - ALU: Computational and logic unit
  - CU: Control unit
  - Memory Unit: Some memory is built into the CPU to support storage and fast calculations. Includes registers and cache.

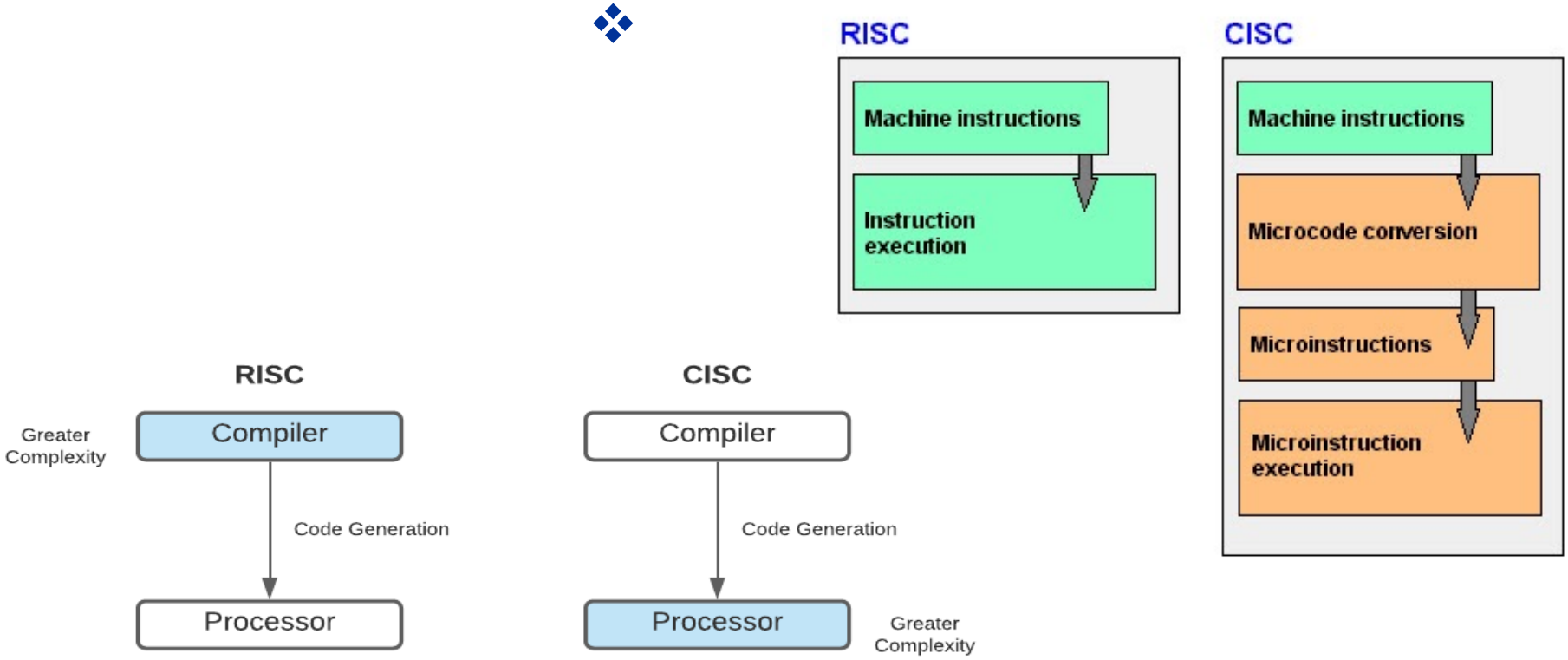


CPU

- Process of executing a command:
  1. Load instructions from memory into CU
  2. CU Decodes the command
  3. The ALU executes the command
  4. The results are saved in memory

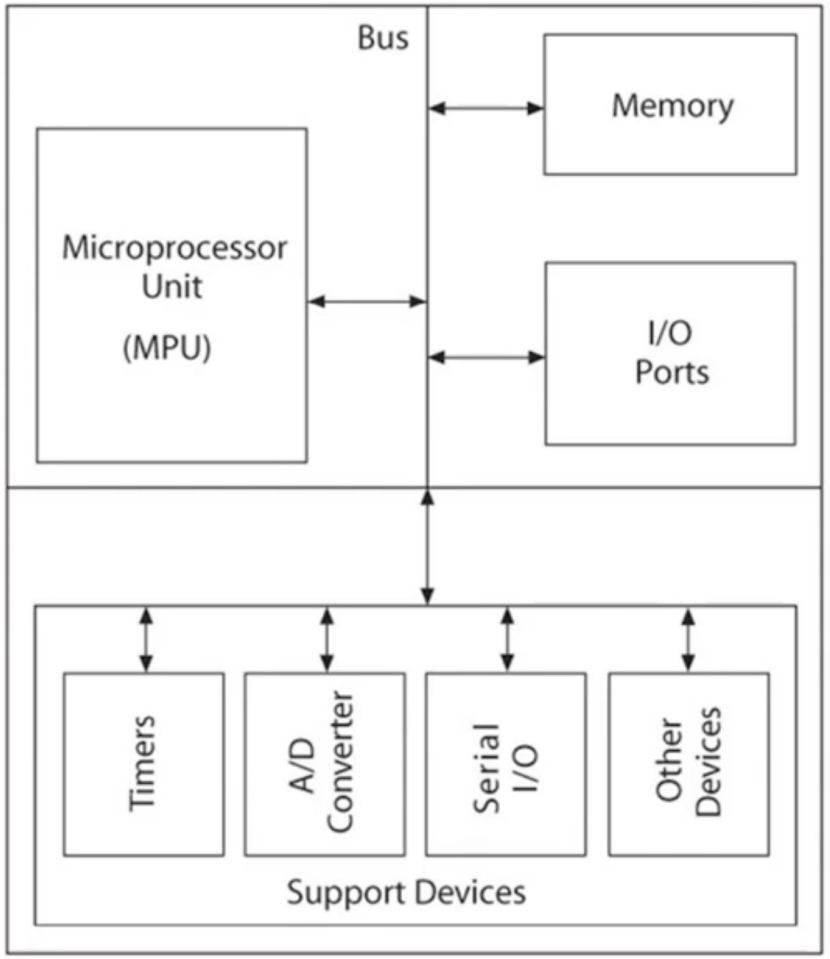


- ISA (Instruction Set Architecture):
  1. CISC (Complex Instrucstion Set Computing)
  2. RISC (Reduce Instrucstion Set Computing) - ARM

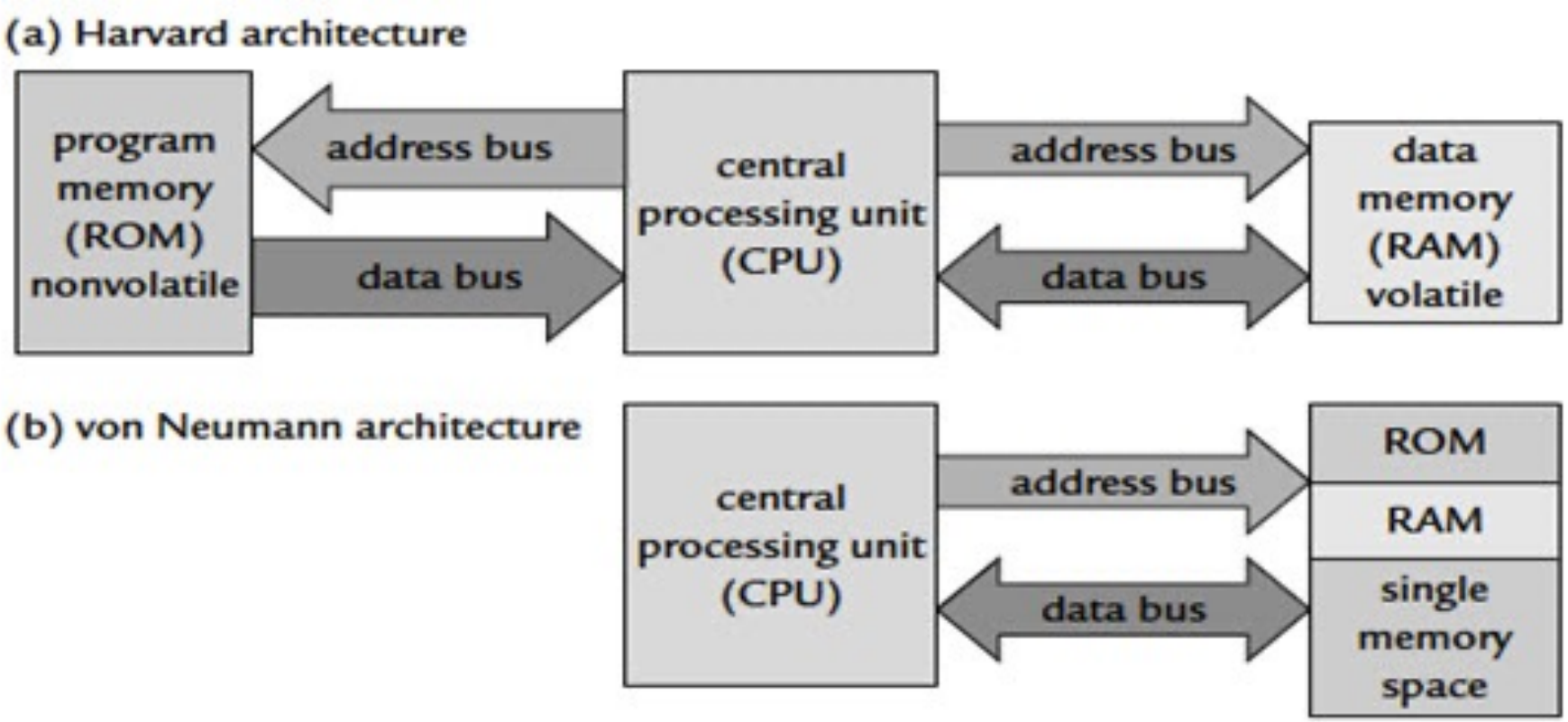


CPU

- Microcontroller is considered as a fully integrated circuit with components:
  - Processor
  - Memory
  - Peripherals (GPIO, Timer, SPI..)
- For example:
  - Processor: Core i3, i5, i7…
  - Microcontroller: PIC, AVR…



Von-Neumann & Harvard



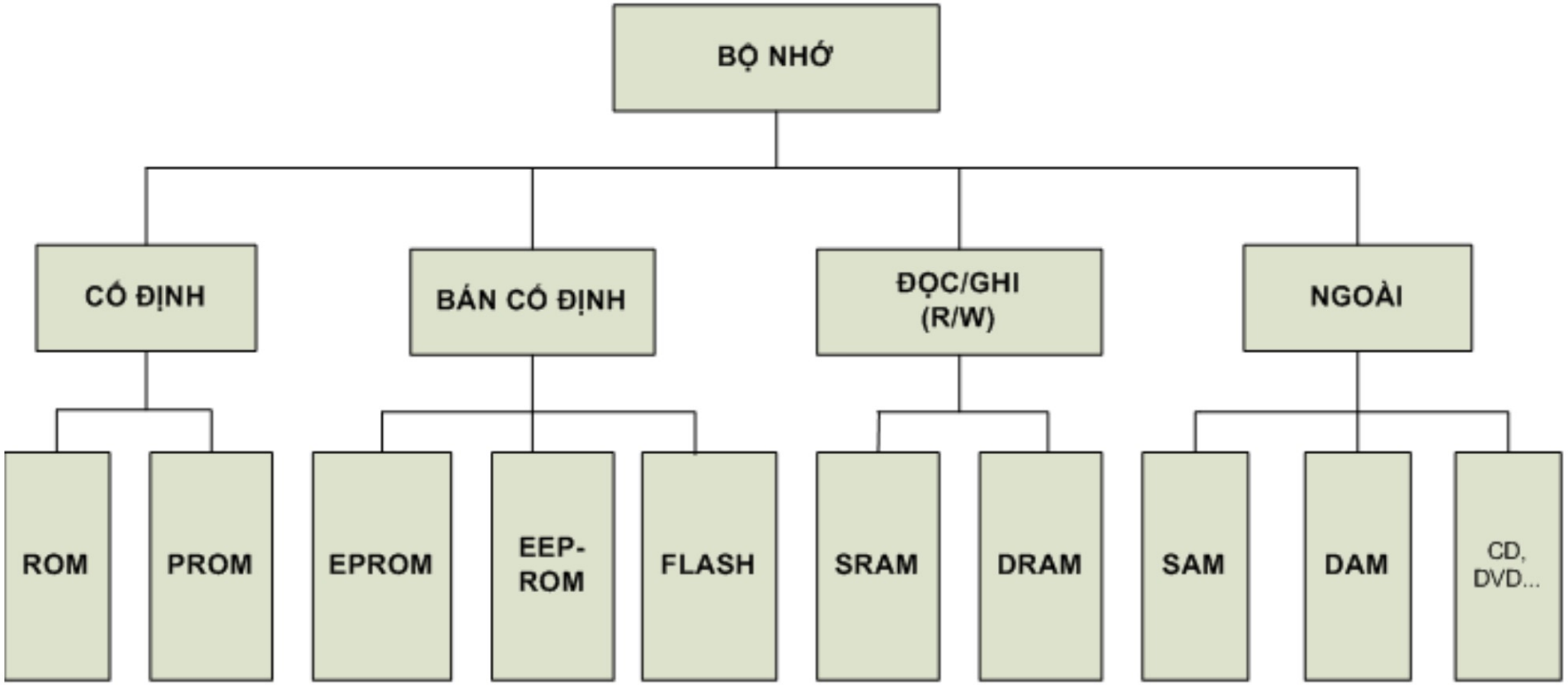
2.2: Memory and memory design

Memory and memory design

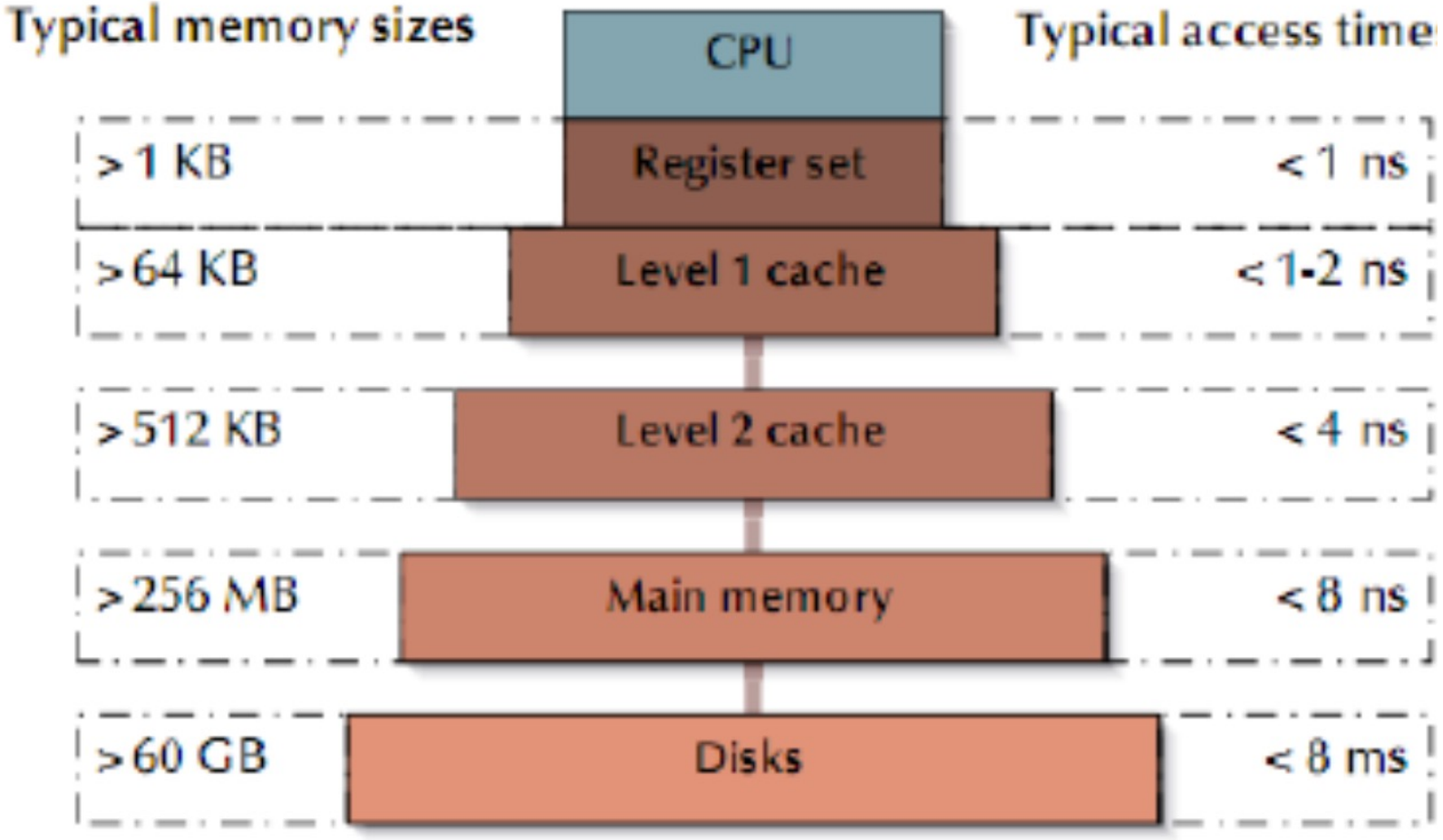
- Memory is a means of storing information including programs and data.
- Some basic information about computer memory.

Memory Type	Access Time	Cost /MB	Typical Amount Used	Typical Cost
Registers	1ns		1KB	
Cache	5-20 ns		1MB	
Main memory	60-80ns		MB	
Disk memory	10 ms		GB	

❖ Classification:



- Memory hierarchy model:

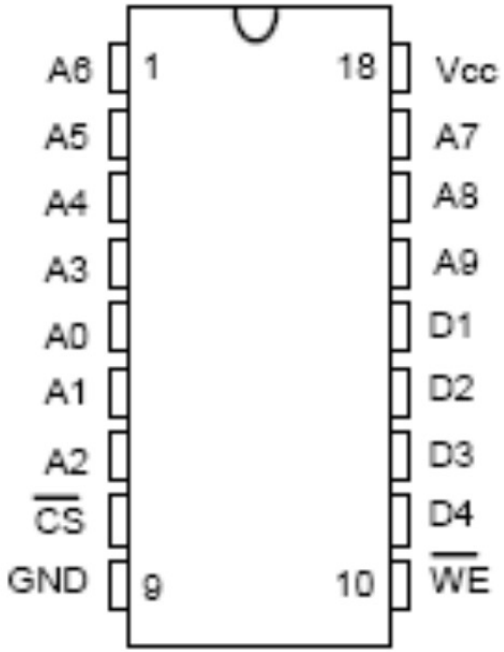


Physical memory organization and memory design

- A memory chip is a specific microchip, arranged with basic pins.
- The pins of a typical memory chip include address bus inputs, data inputs, chip select control pins, write/read and power pins.
- For example: a 1Kx4 static RAM (1024 “memory words”, each word is 4 bits long

A0 ÷ A9	Các chân địa chỉ
D1 ÷ D4	Các chân dữ liệu
CS	Chân chọn chip
WE	Điều khiển Ghi/Đọc
Vcc	Chân nguồn nuôi +5V
GND	Chân nối đất

Hình III.8 Sơ đồ nối chân một vi mạch nhớ RAM 1Kx4



Physical memory organization and memory design

❖ Designing memory from available memory chips has the following steps:

1. Determine the number of memory chips to create the required memory capacity according to the formula:

M= Q/D, whereas

Q is the capacity of memory.

D is the capacity of each chip

M is the number of memory chips needed.

2. Determine the number of base address wires (ie the number of low address wires connected directly to the memory chip or interconnect chip): according to the following expression:

$2^m = D$, whereas

m is the wire number of the base address.

D is the capacity of each chip

3. Determine the number of address lines to create chip select according to the following expression:

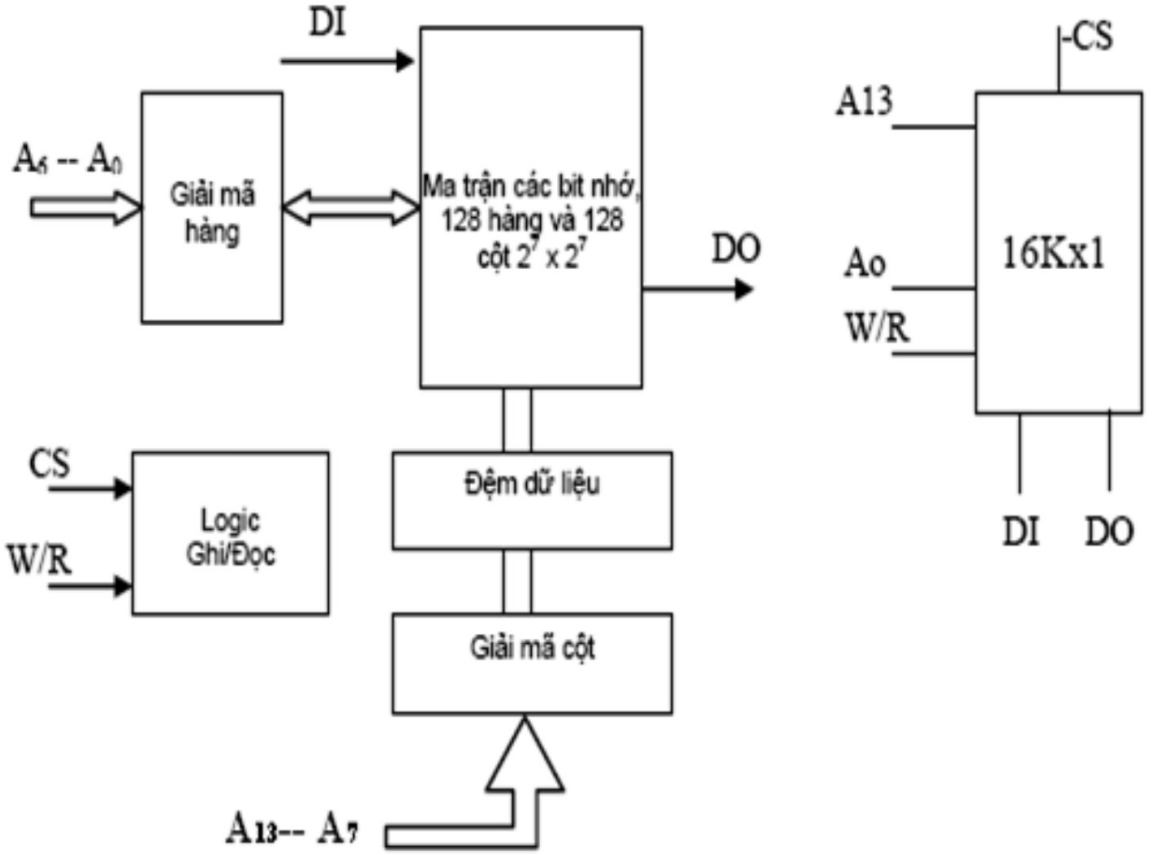
$2^i = M$, whereas

D is the capacity of each chip

i is the number of wires needed to decode the signals chip select signal (Csi)

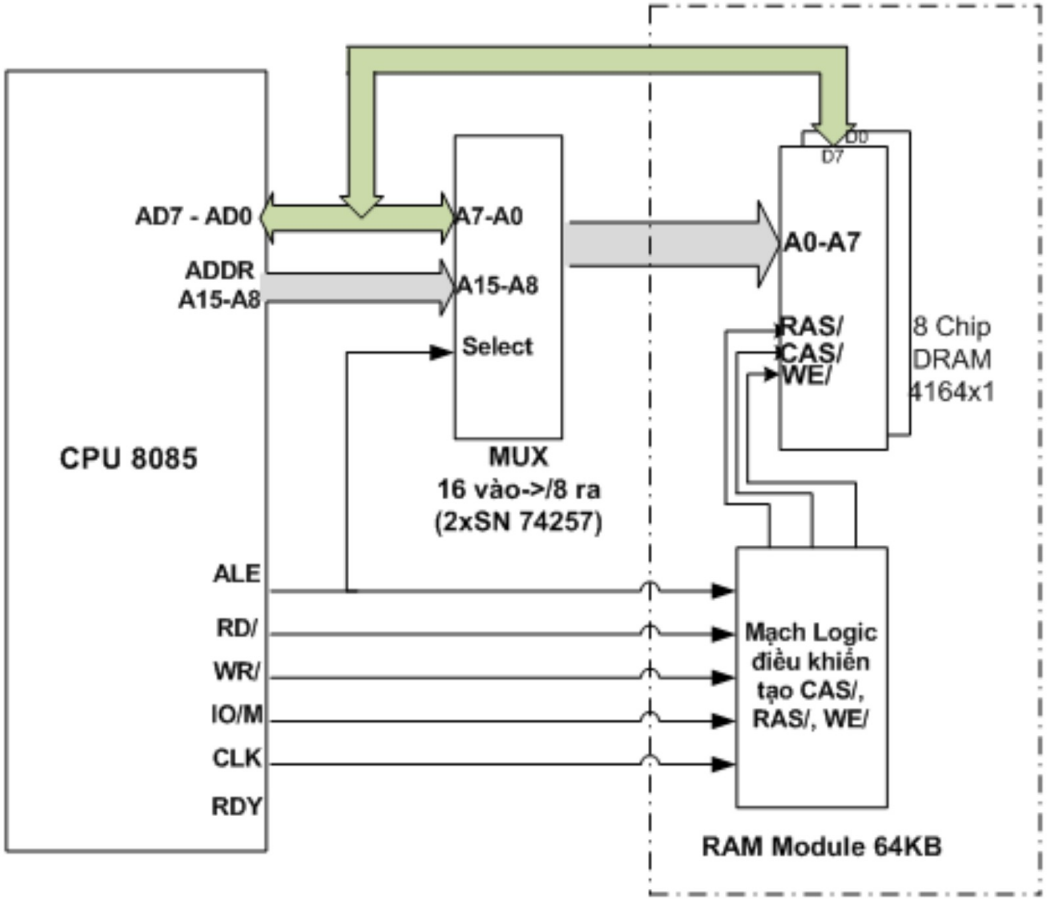
❖ *Design Static RAM*

Build a 16Kbyte memory based on 16Kx1bit SRAM chips. The 16Kbyte SRAM memory tape is built on the basis of 8 16K x 1bit SRAM chips, to obtain a memory cell with a length of 8 bits.



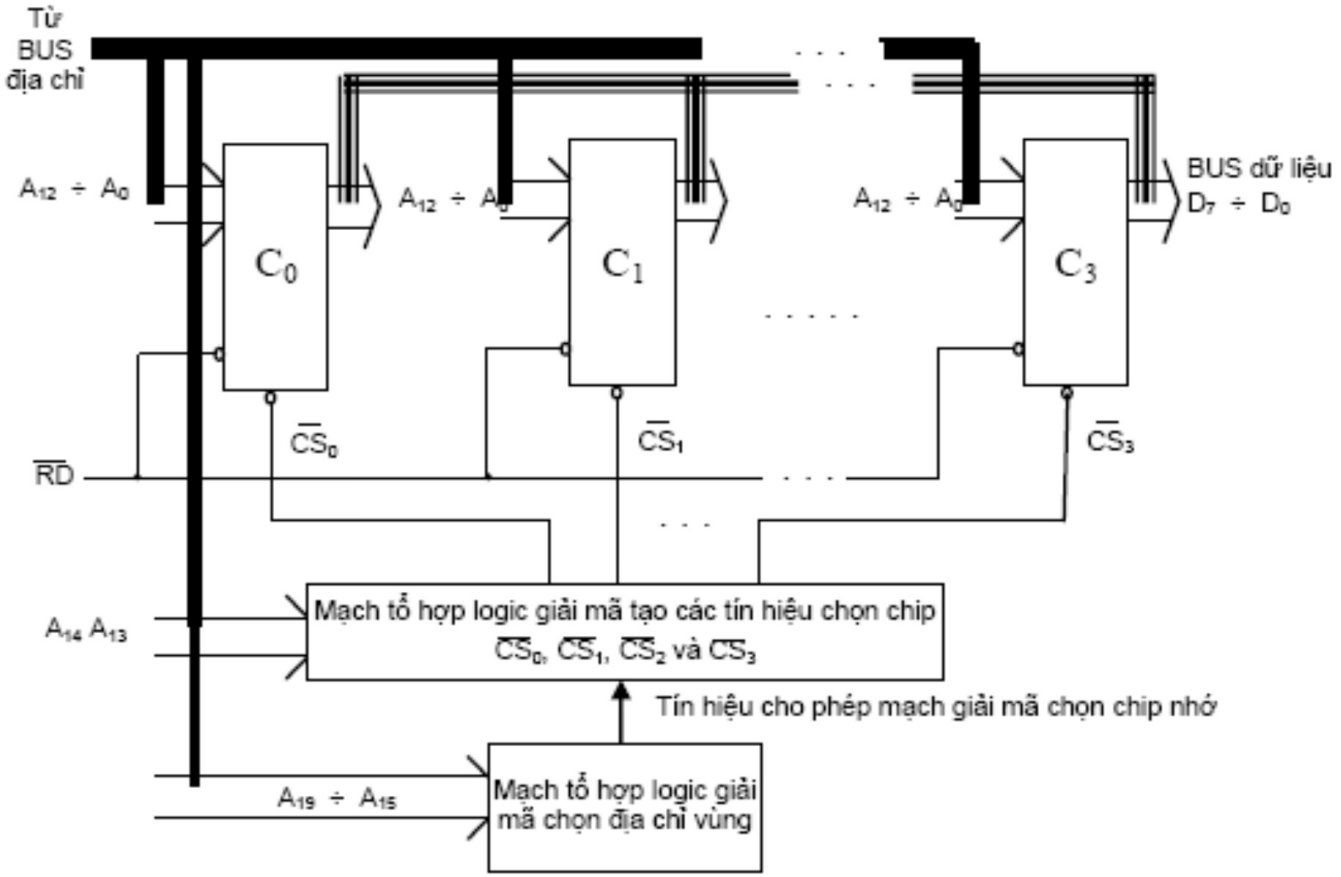
❖ *Design DRAM*

DRAM MK 4164 is 64K bit DRAM in 1 Chip. Assuming we will design RAM for the 8085 CPU with a maximum RAM of 64 KB, 8 Chips will be needed.



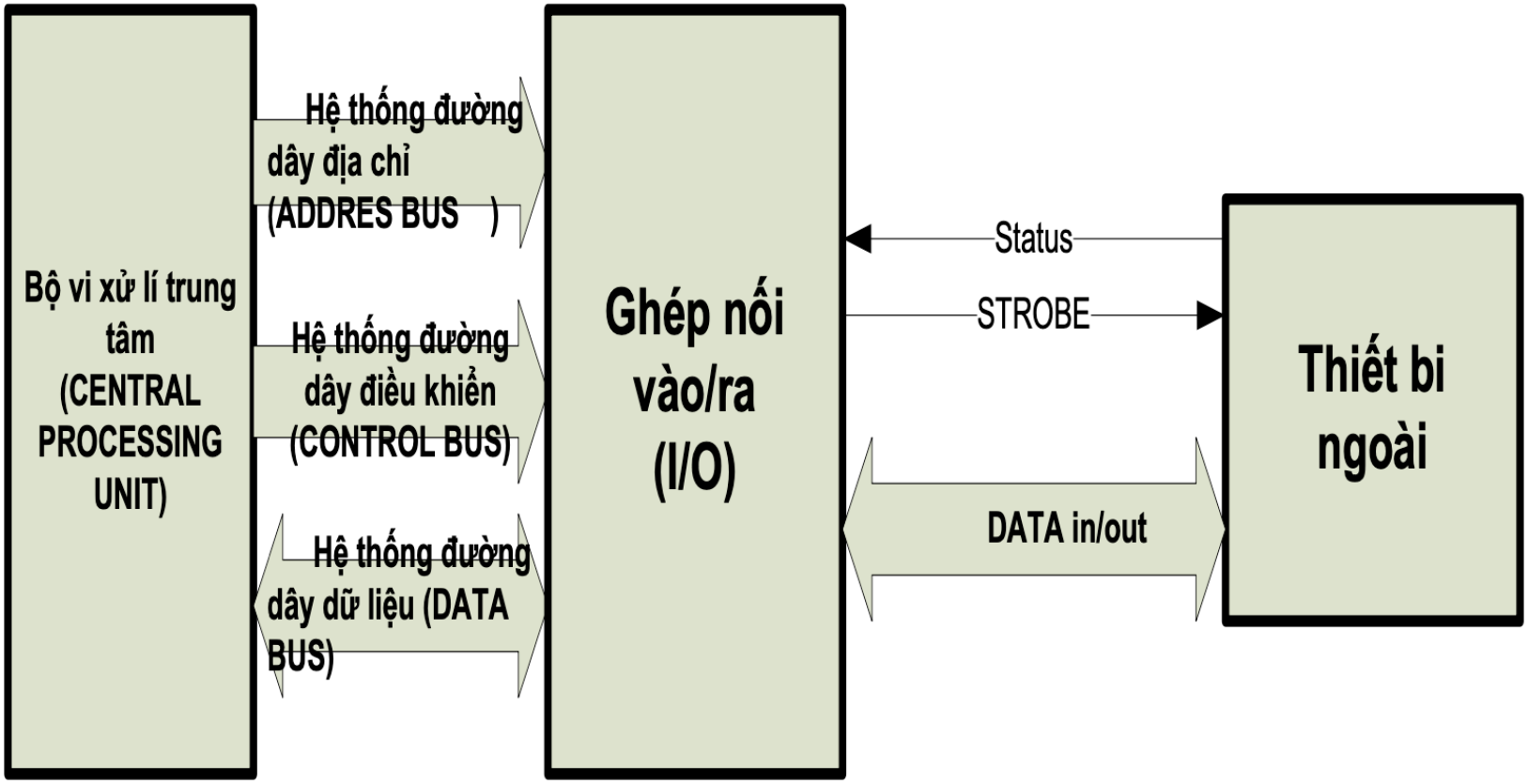
❖ *ROM/EPROM*

Build a ROM module with a capacity of 32KB, using Chip 2764 8K x 8 bits, the first address is 20000hex. The application program loads into this module.



2.3: Pair with peripheral devices

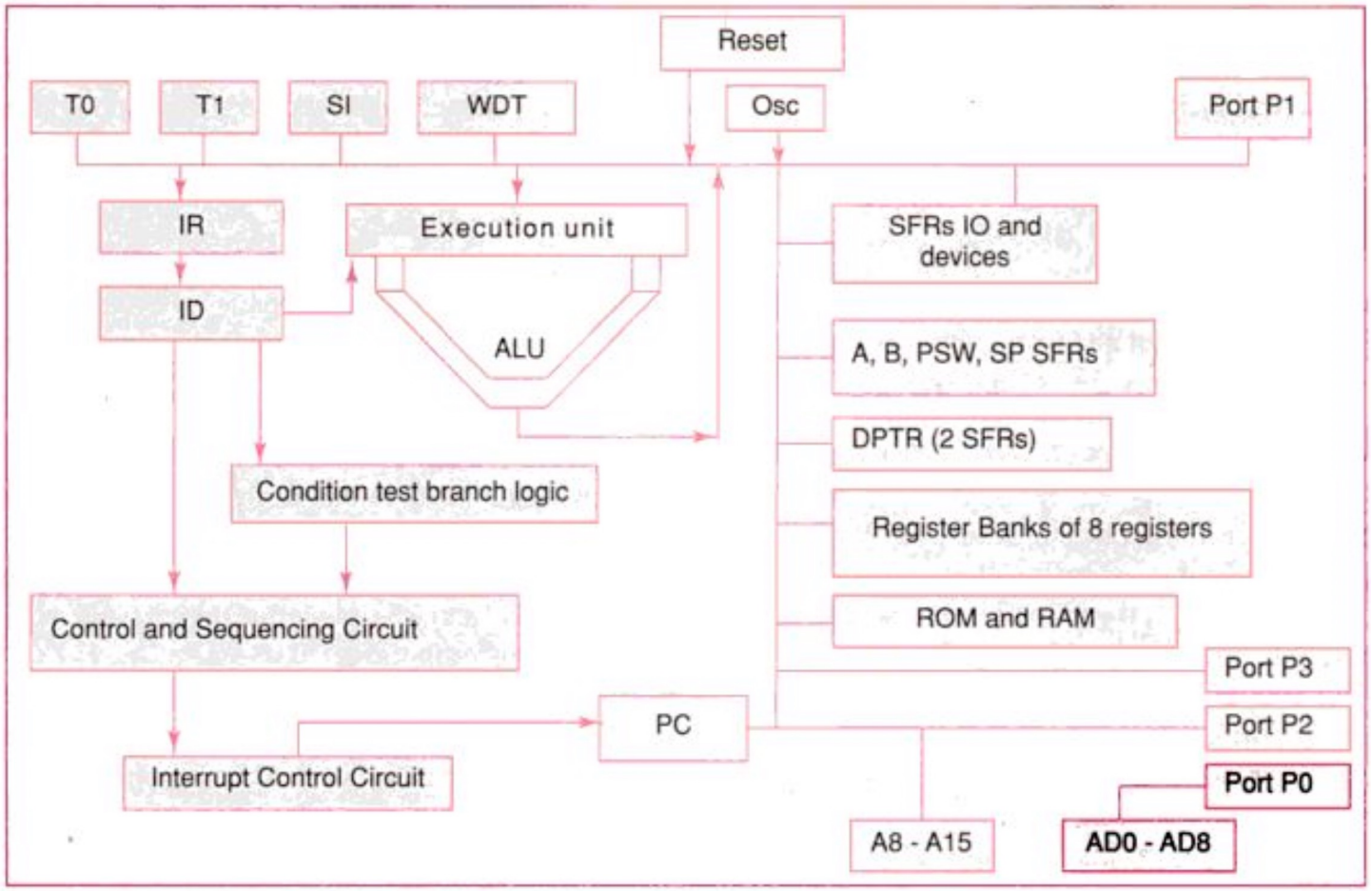
Model of coupling technique



2.4: 8051 microprocessor: Architecture, memory and practical interface

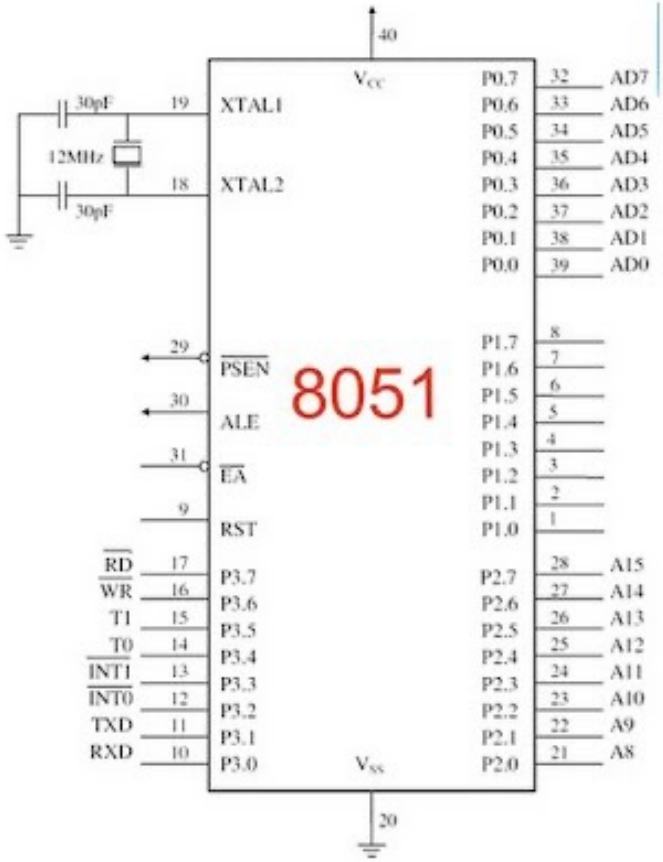
8051 microcontroller architecture

- Basic 8051 architecture: microprocessor, registers, memory (in Harvard architecture) and ports, counters/timers, I/O, and interrupt handlers.



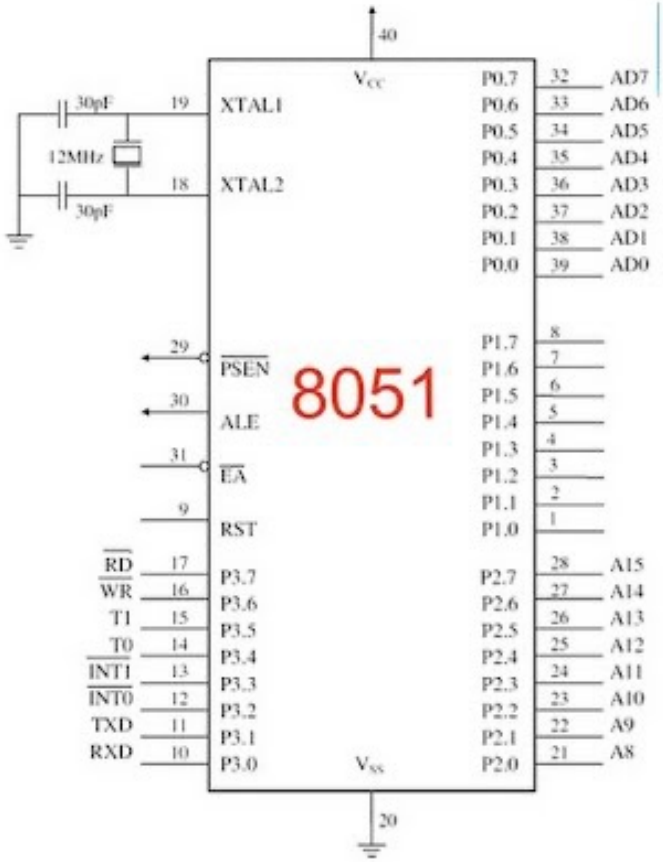
8051 microcontroller architecture

- Characteristics of 8051
  - 12 MHz clock. Processor instruction cycle time 1 μs.
  - ALU-8bit
  - Harvard Memory Architecture
  - Internal 8-bit data bus and Internal 16-bit address bus
  - CISC instruction set architecture



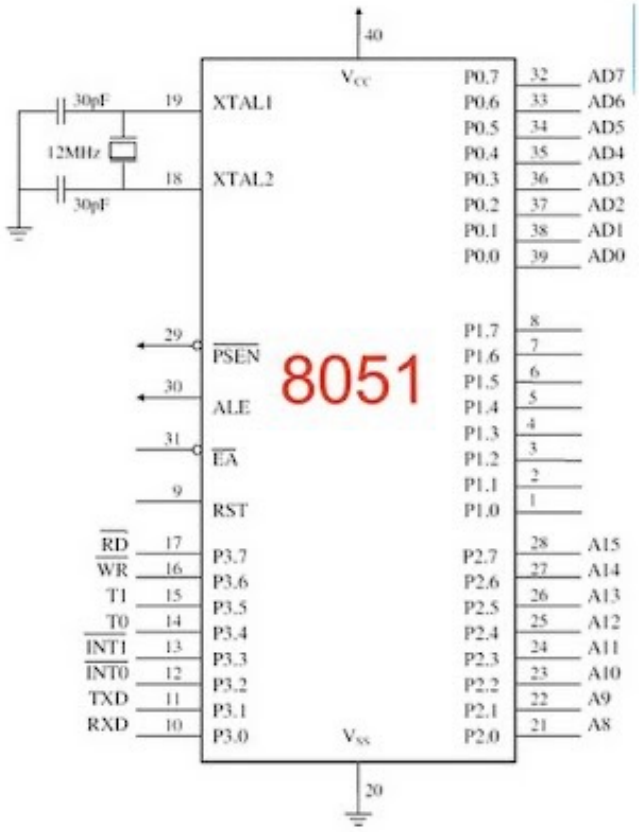
8051 microcontroller architecture

- Characteristics of 8051
  - Special Registers (SFR)
    - PSW (status word)
    - A (cumulative)
    - Register B, SP (stack pointer)
    - Registers for I/O, timers, ports, and interrupt handlers
    - 16-bit program counter (PC) with factory default value of 0x0000.
    - 8-bit stack pointer (SP) with an initial default value of 0x07



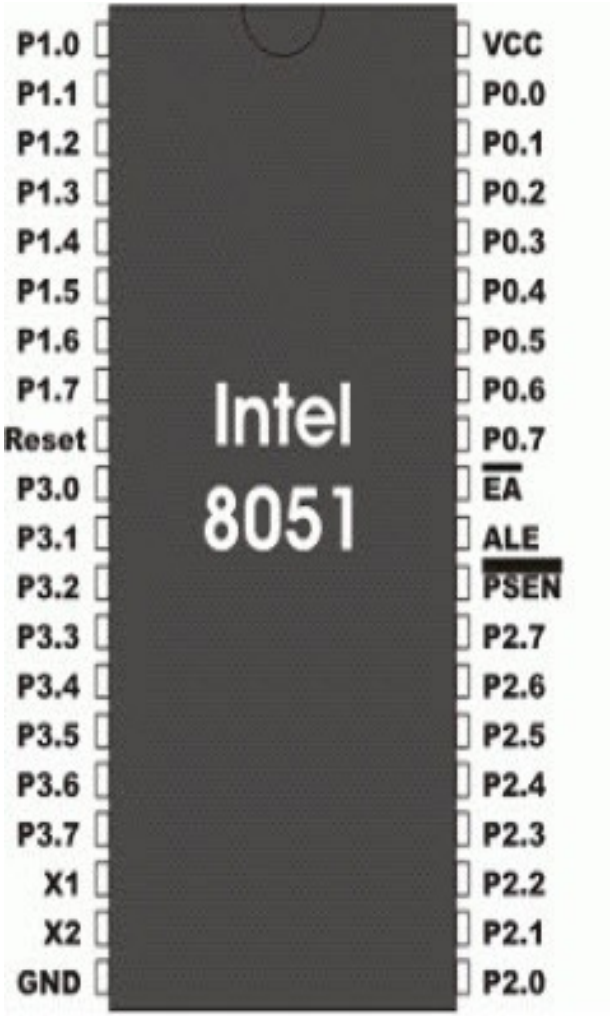
8051 microcontroller architecture

- Characteristics of the simple 8051:
  - Does not handle floating point,
  - No Cache,
  - There is no MMU memory management unit,
  - There is no atomic operations unit.
  - There is no CPU Pipeline engineering
  - Do not process commands in parallel.



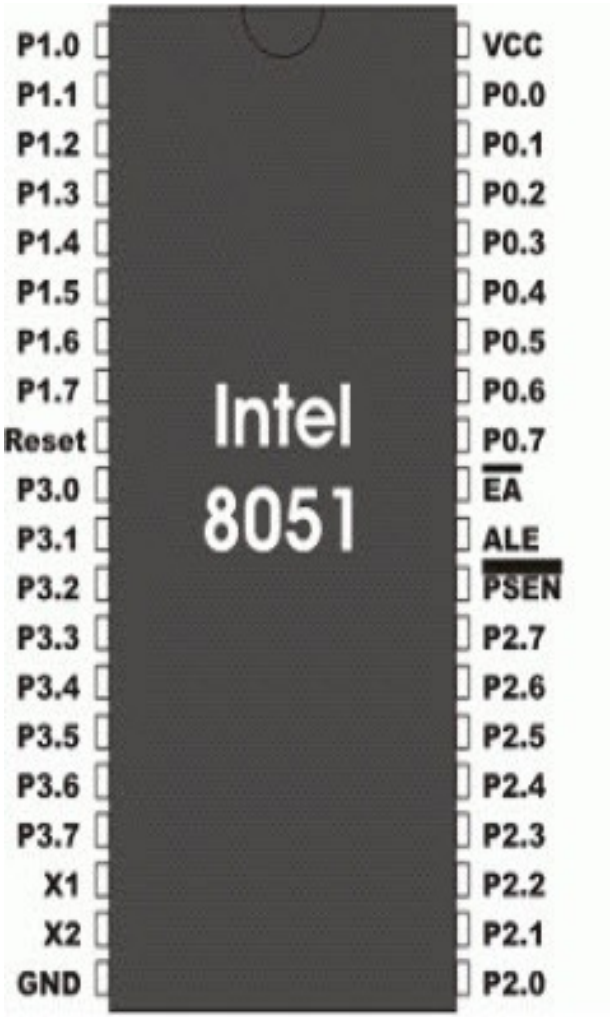
8051 microcontroller architecture

- Features of 8051:
  - RAM size 128bytes
  - The 32 bytes of RAM are used as four register sets (register-set/bank). Each bank has 8 registers.
  - Stack memory/External data memory can be added up to 64 kB.



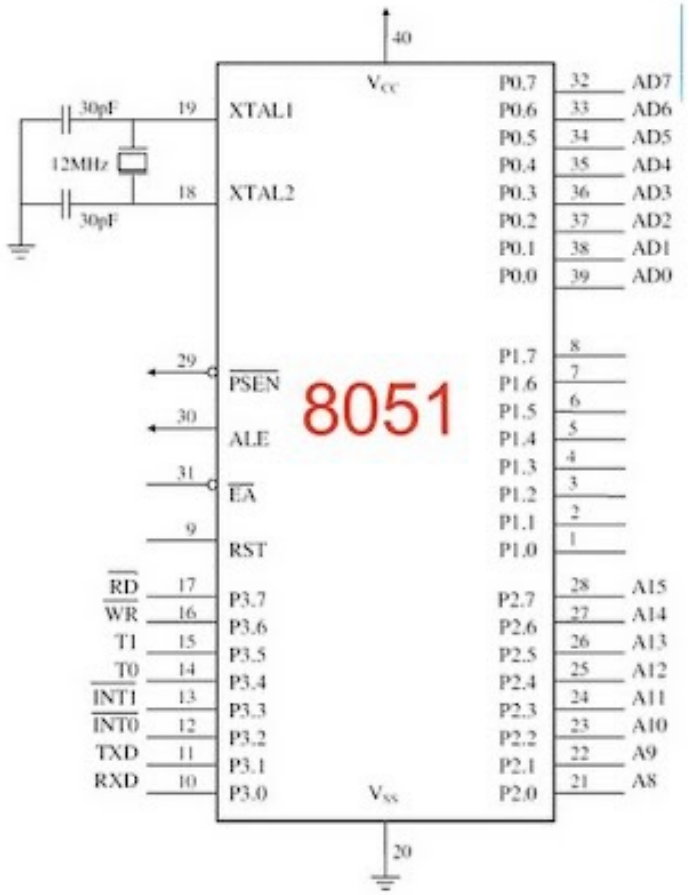
8051 microcontroller architecture

- Some improved versions of the 8051
  - Version 8351 has on-chip ROM
  - Version 8751 uses EPROM
  - Version 8951 uses on-chip EEPROM or 4 kB flash memory.
  - In the extended 8051 versions the address space is expanded up to 16 MB.



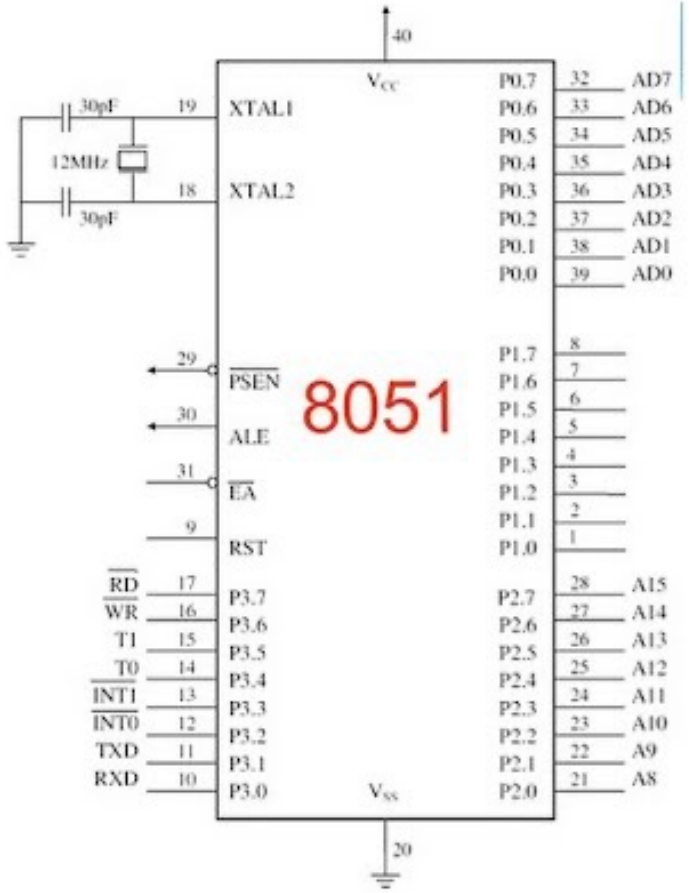
8051 microcontroller architecture

- Features of 8051:
  - Two interrupt pins 12 and 13 INT0 and INT1.
  - Four ports of 8 bits each (P0, P1, P2, P3)
  - Two timers T0, T1.
  - Serial Interface (SI) – programmable for three full-duplex UART modes for I\O



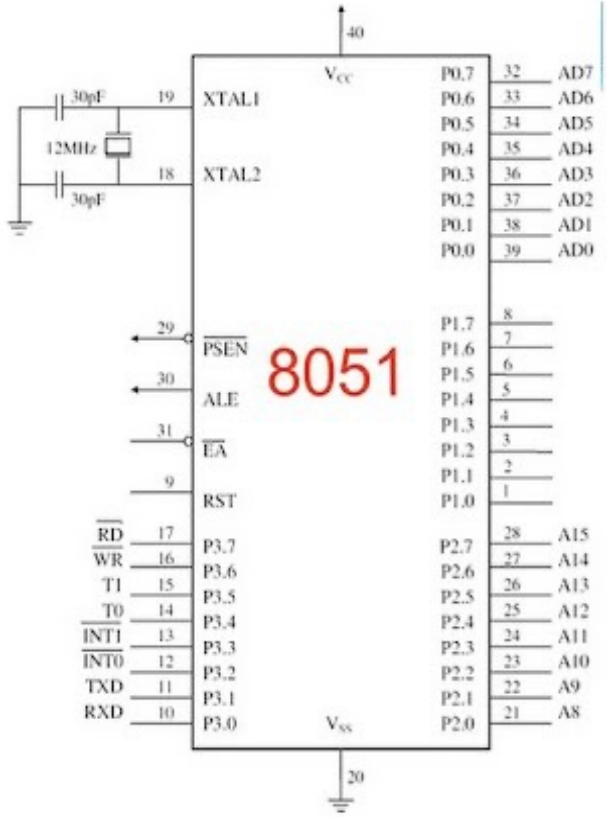
8051 microcontroller architecture

- Features of 8051:
  - In some versions – there is a DMA controller, a pulse width modulator
  - In some versions of the 8051, modems, watchdog timers, and ADCs were also integrated.
  - For example, the Siemens SAB 80535-N supports an ADC with a programmable reference voltage.



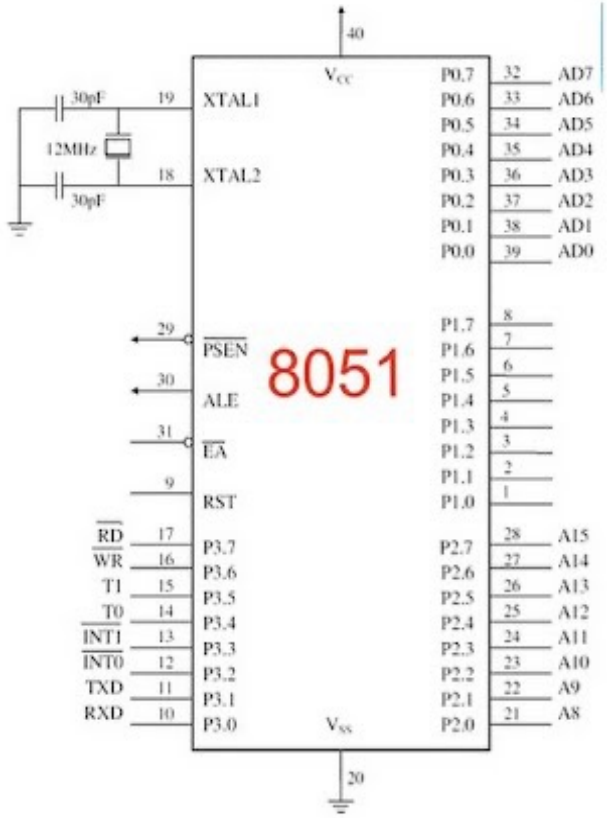
8051 INSTRUCTION SET ARCHITECTURE

- **Data transfer commands are:**
  - Move bytes between register A and another register
  - Move bytes from one register/from internal RAM to another register
  - Indirect move:
    - MOVC indirect
    - MOVX indirect
  - Instant migration:
    - MOV immediate DPTR
  - Push or Pop direct



8051 INSTRUCTION SET ARCHITECTURE

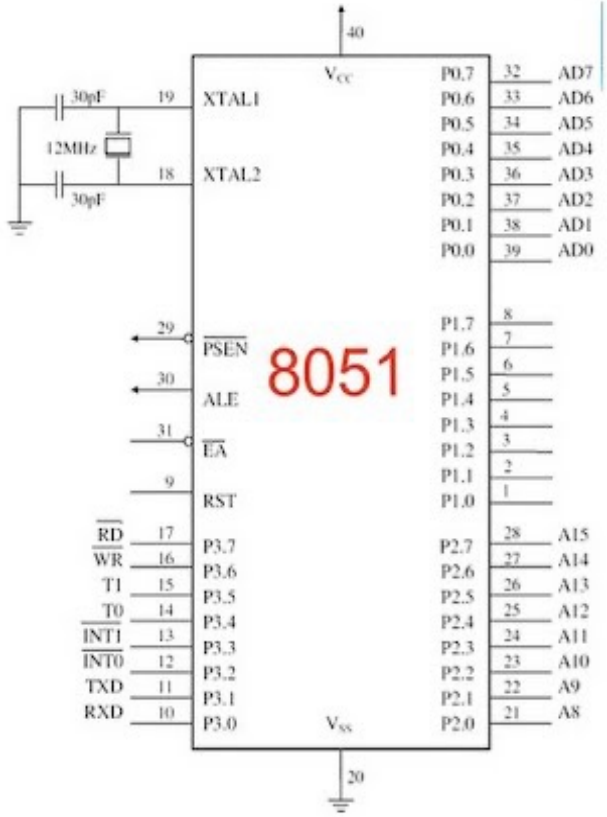
- **Manipulates bits, bytes, and logic instructions**
  - Bit manipulation:
    - Set value, add value with commands like AND, OR or MOV (Clr, Set, Mov…)
  - Logical command:
    - Logical instructions AND, OR, XOR
  - Byte operations:
    - Delete, add or swap commands and rotation commands (CLR, RL, RC…)



8051 INSTRUCTION SET ARCHITECTURE

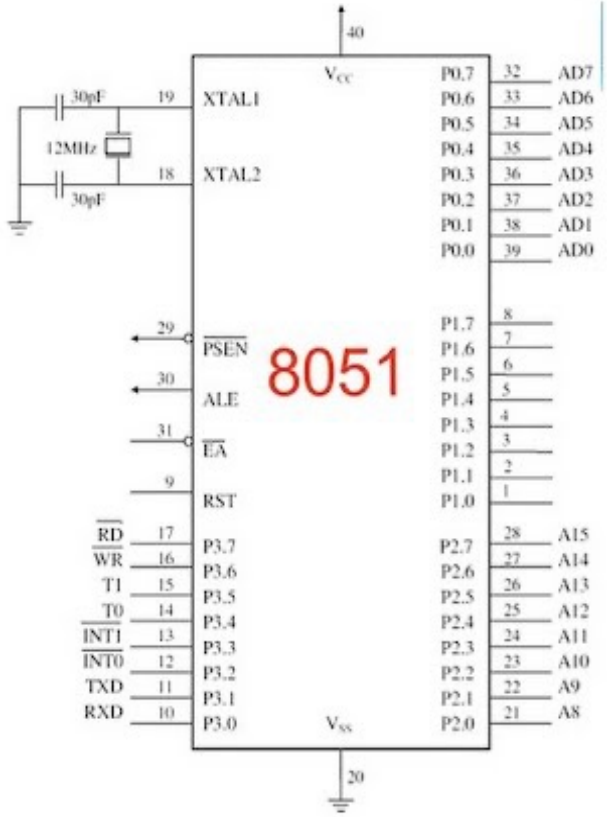
❖ Math commands:

- These are 8-bit instructions
  - Add (Add), subtract (Sub), multiply (Mul), divide (div)
- Increase or decrease orders:
  - Inc (Increment), Dec (Decrement)



8051 INSTRUCTION SET ARCHITECTURE

- **Program flow control commands:**
  - Branching
  - Conditional jump command
  - Comparison command
  - Subprogram call
  - NOP command
  - Delay command
  - Break command:
    - Flow control interrupt mask bits, priority bits
    - RETI



8051 I/O Port

- **8051 has 4 input and output ports: P0, P1, P2, P3:**
  - It is an 8 bit input and output port
  - The addresses are 0x80, 0x90, 0xA0, 0xB0 respectively
    - Each bit of port P0 is denoted P0.0 to P0.7 corresponding to addresses 0x80 to 0x87.
  - Similar to P1, there are 8 ports P1.0 to P1.7 corresponding to addresses 0x90 to 0x97. P2.0 to P2.7 are addresses 0xA0 to 0xA7, P3.0 to P3.7 are addresses 0xB0 to 0xB7.
  - For example:
    - MOV 0xA0, #0x0F ; means assigning value to port P2 bits is 0000 11112
    - INC 0xA0 ; P2 = 0000 11112 + 1 = 0001 00002

8051 I/O Port

❖ P0 và P1 ports

P0

P0.0 P0.1 P0.2
P0.3 P0.4 P0.5
P0.6 P0.7
Also as
AD0- AD7

P1

P1.0 P1.1 P1.2
P1.3 P1.4 P1.5
P1.6 P1.7
Also P1.6 as I^2C clock, P1.7 as I^2C serial data, and P1.0 and P1.1 for T2 (8052)

- The pins of P0 are both the low 8 bits of the address bus and the data bus

8051 I/O Port

❖ **P2 và P3 ports**

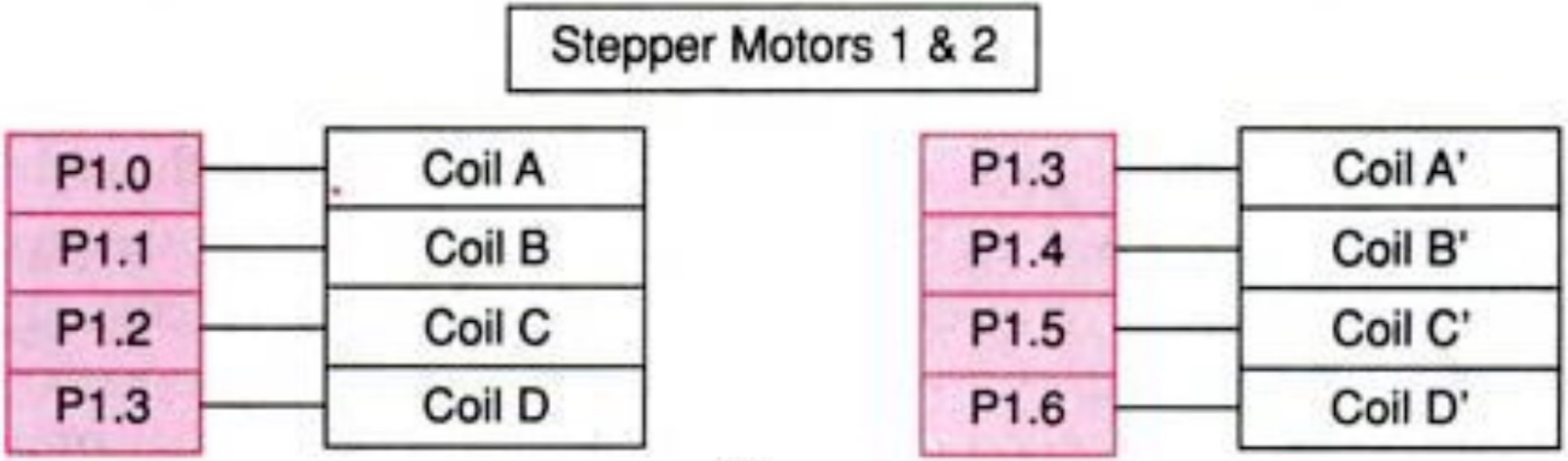
P2
P2.0 P2.1 P2.2
P2.3 P2.4 P2.5
P2.6 P2.7
Also as
A8- A15

P3
P3.0 P3.1 P3.2
P3.3 P3.4 P3.5
P3.6 P3.7
Also
RxD/SyncData,
TxD/SyncClk,
INT0/GT0,
INT1/GT1, T0,
T1, WR, RD

- The pins of P2 are the high 8 bits of the address bus

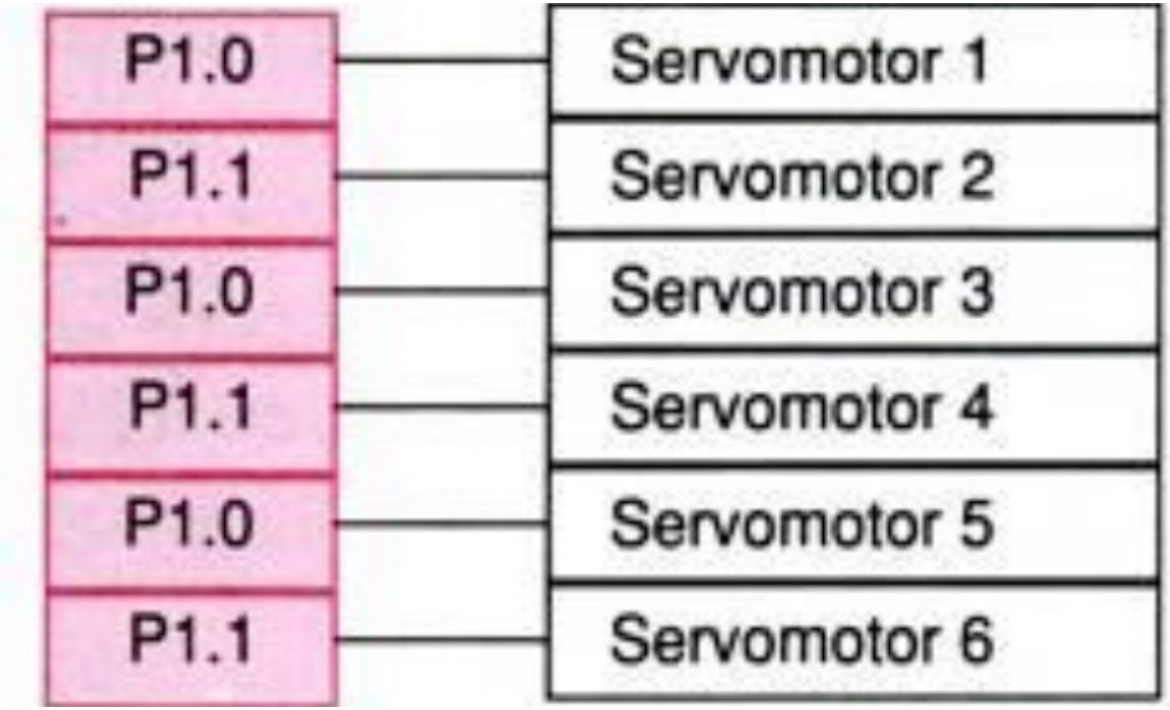
8051 I/O Circuit

- The circuit of input and output ports controls stepper motors used in printers



8051 I/O Circuit

- Circuit of input and output ports for six servo motors in the robot



Byte programming using 8051 ports

- Byte addresses of I/O ports P0, P1, P2 and P3 of the 8051 used to access and execute read or write or other commands
- The direct 8-bit address of each address is given in the instructions
  - For example: MOV 0xA0, #0x0F ; means assigning values to the bits of port P2 is 0000 11112
- The addresses of the bytes at P0, P1, P2 and P3 are 0x80, 0x90, 0xA0 and 0xB0 respectively.

Byte programming using 8051 ports

- For example:
  - MOV 0xA0, #0x0F ;
  - INC 0xA0 ;

- That means assigning values to the bits of port P2 is 0000 11112
- Then port P2 will be increased by 1 value
- P2 = 0000 11112 + 1 = 0001 00002

Byte programming using 8051 ports

- Ports P0, P1, P2 and P3 each have 8 bits
- Each bit has an address to access and perform reading or writing using bit manipulation instructions.
- Address is the bit address of the pins. Each bit address is specified in the corresponding instruction.
  - Bit addresses P0.0 to P0.7 correspond to 0x80 to 0x87.
  - Bit addresses P1.0 to P1.7 correspond to 0x80 to 0x97.
  - Bit addresses P2.0 to P2.7 correspond to 0xA0 to 0xA7.
  - Bit addresses P3.0 to P3.7 correspond to 0xB0 to 0xB7.

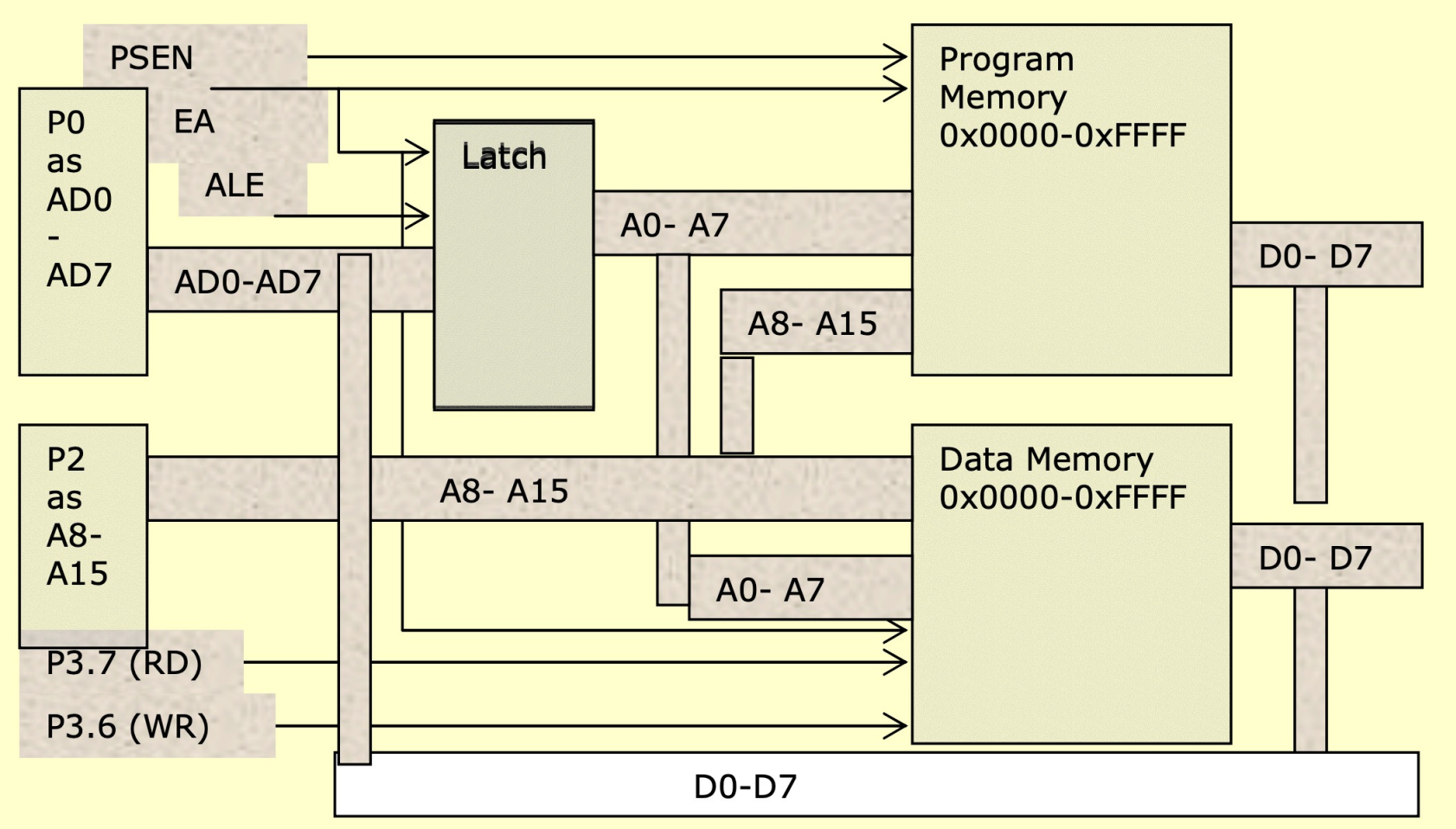
Byte programming using 8051 ports

- For example:
  - CPL 0x90 ; Port P1 is assigned the value 0
  - CLR 0x80 ; P0.0 has a value of 0.
  - SETB 0x80; P0.0 has a value of 1.
  - CLRB 0x80; Assign P0.0 to a value of 1.

Mapping in and out of memory

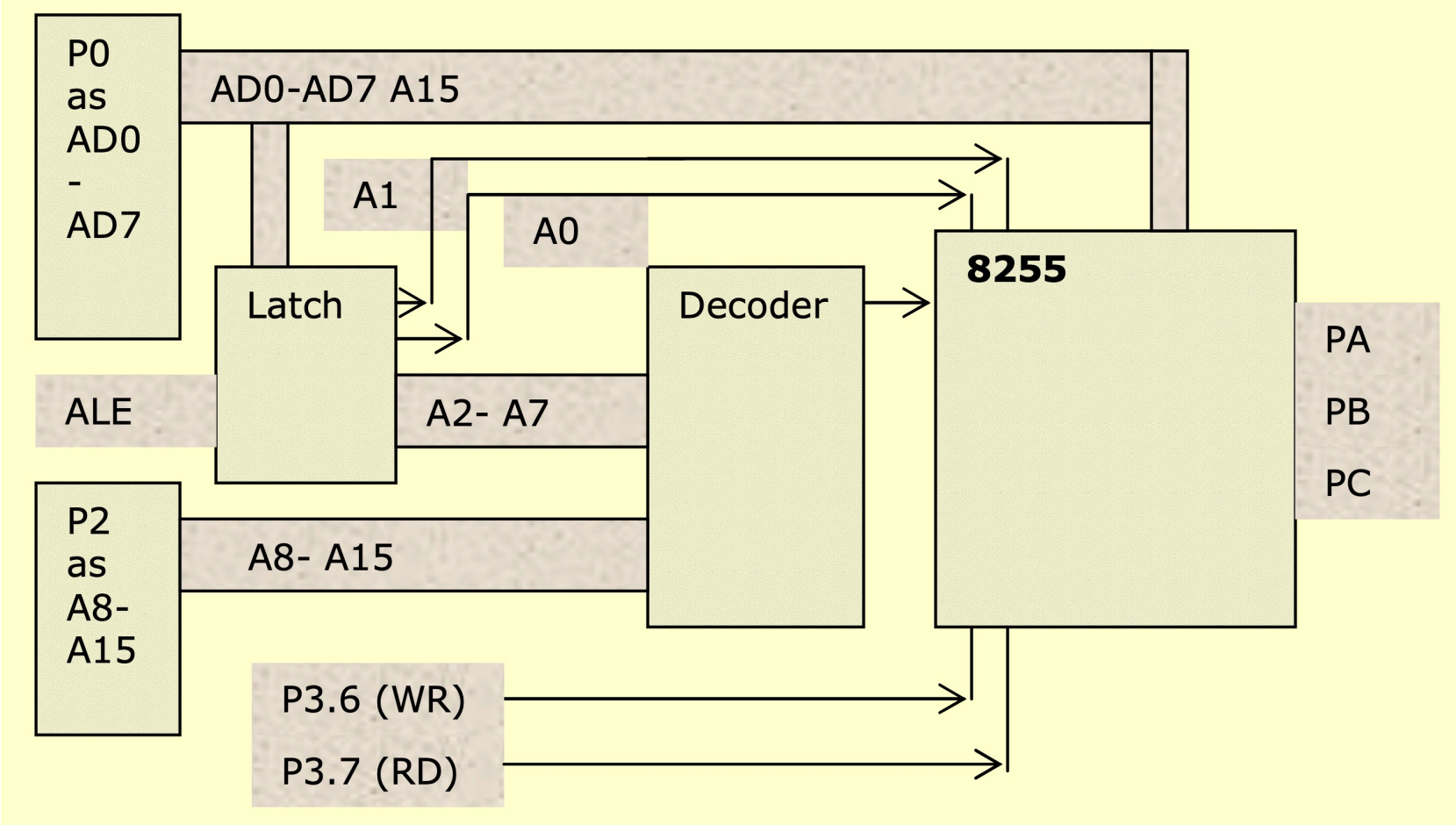
- The memory and ports in the 8051 are both assigned addresses, each port has its own address range in the data memory address space.
- The communication circuitry is identically designed for the memory to connect to the external and programmable peripheral (PPI) ports.

Connection to external program and data memory circuits



Pair with external storage

The interface uses the PPI external port of the 8051



Pair with external storage

Port P0 in extension mode and ALE signal

- The AD0-AD7 pins of port P0 are multiplexed signals of the low address bits A0-A7 of the address bus and bits D0-D7 of the data bus.

- Time division multiplexing of signals A0-A7 and D0-D7 based on the address latch enable (ALE) processor signal

Harvard memory architecture

- There are two memories – program memory and data memory.
- Two control signals – PSEN and RD to control reading from program memory or data memory.
- ALE control signal to control the use of AD0-AD7 as address or data at a given instance

EA control signal

- Is a signal that when enabled – the processor always accesses external memory addresses instead of internal memory or register addresses

Pair with external storage

External address RAM, Register, program memory when EA control signal is not active

- VXL always accesses external memory whether EA is active or not.
- RAM addresses and registers are between 0x00 and 0xFF just like external data memory addresses are between 0x0000 and 0xFFFF.
- Internal program memory addresses 0x0000 to 0xFFF (in the case of 4 kB internal ROM) are the same as external program memory addresses 0x0000 and 0xFFFF.

Counter/timer devices

- The original 8051 family had two timers T0 and T1
- The 8052 family (extension of 8051) has three timers T0, T1 and T2
- **Counting/timing device used as a timer**
- A timing device when counting inputs are provided by a clock.
- Clock pulses are given at specified intervals: acting as a timer.
- **Counting/timing device used as counter**
- A counting device when the inputs are for counting. The counter is provided with input to count from the external input pin.

Counter và Timer

- Timer T0 and T1
- TMOD special register
- TCON special register on four bits
- Four external P3 pins for external control and external counting input
- Modes 0, 1, 2, 3 of T0
- Modes 0, 1 and 2 of T1

Timing and counting devices External controls to activate or deactivate running

- When timing or counting devices are externally controlled by gate input, when GT0 or GT1 is externally triggered, the device can operate otherwise it will shut down in gate input mode.

- The GT0 or GT1 signal is given at P3.2 and P3.3.

Counter/timing device External counting input in counter mode

- T0 count T0 is given input for counting from the external input T0 pin at P3.4.
- T1 counts when T1 is given input to count from external input pin T1 at P3.5.

Two special registers TH1-TL1

- To access the number or time of the higher 8 bits and lower 8 bits of device T1
- Two 16-bit storage registers of the T1 device.

Two special registers TH0-TL0

- To access the number or time of the higher 8 bits and lower 8 bits of device T0
- Two registers store 16 bits of device T0.

TMOD special register

- Controls T1 and T0 modes using 4 bits per mode, programming count/timing of T1 and T0.
- One bit in each function controls whether the external port input controls or not.
- A bit that controls the function for which counter or timer mode is used.
- Two bits control the function mode of the timer/counter such as mode 0 or 1 or 2 or some other action

Special register TCON
Control and status bits T1 and T0

- The four bits 4 above program the counting/timing device modes T1 and T0.
- TCON.7 and TCON.5 show the timer/counter overflow status for T1 and T0, respectively.
- TCON.6 and TCON.4 control the start and stop of the timer/counter
- The lower bits of TCON are for interrupt control for INT0 and INT1

Timer/Counter T0

- 8 register bits TMOD (lower 4 bits), TCON (bits 5 and 4), TL0 (count/time bit), TH0 (count/time bit)
- Counter with input at P3.4 when bit 2 TMOD = 1, timer with internal clock timing input when bit 2 TMOD = 0
- When mode is set = 0, 8-bit Timer/Counter mode and TH0 are used and TL0 is used to pre-scale (divide) the input by 32
- When mode is set = 1, 16-bit timer/counter mode with TH0-TL0 is used for timing or counting

Timer/Counter T1

- 8 bits used Register TMOD (upper 4 bits), TCON (bits 7 and 6), TL1 (count/time bit), TH1 (count/time bit)
- Counter with input at P3.5 when bit 6 TMOD = 1, timer with internal clock timing input when bit 6 TMOD = 0
- When mode is set = 0, 8-bit Timer/Counter mode and TH1 are used and TL1 is used to pre-scale (divide) the input by 32
- When mode is set = 1, timer/set mode đếm 16 bit với TH1-TL1 được sử dụng để định thời hoặc đếm

Timer/Counter T1

- When mode is set = 2, the 8-bit Timer/Counter TH1 is used and TL1 is used to automatically reload TH1 after timeout using the preset value at TL1
- When mode is set = 3, T1 stops because TH0 is now active instead of T1.

Serial data communication input/output

- Serial Interface function (Serial Interface)
- Half-duplex synchronous serial mode 0
- Full duplex asynchronous UART mode 1, 2 or 3
- SBUF
- SCON

Serial Interface SI

Programming for:

- SI serial interface
- full-duplex asynchronous UART mode

Serial data communication input/output

Two special 8-bit registers

- SBUF (8 bit serial receive or transmit bit register depending on whether the command is using SBUF as source or destination)
- SCON (8-serial mode cum control bit register) and SFR PCON.7 bit

SBUF

- Unique register address for the transmitted and received byte buffer when serial output or input is sent.
  - 0x99 address of SI buffer.
  - The register holds the 8-bit SI line as it is written.
- For example:
  - MOV 0x99, instruction A writes A to the transfer buffer from register A
  - MOV R1, read command 0x99 Register R1 from receive buffer

SCON

- Register to control the SI interface.
- The three bits above program modes such as 0 or 1 or 2 or 3.
  - Mode 0 is full duplex synchronization.
  - Mode 1 or 2 or 3 is full duplex asynchronous mode.
  - Bit SCON.4 enables or disables the SI receiver function.
  - The two bits SCON.3 and SCON.2 specify the 8th bit transmitted and the 8th bit received when the mode is 2 or 3. One bit SCON.1 enables or disables the SI transmitter interrupts (TI) when complete the transmission.
  - Bit SCON.0 enables or disables SI receiver interrupts (RI) on completion of transmission.

Mode function 0 – Input or output

- Depends on the instruction using SBUF as source or destination
  - Synchronous serial mode data and clock input
  - Synchronous serial mode data and clock output
- Mode 0 when SCON bits 7 and 6 (mode bits) are 00

POST AND TELECOMMUICATIONS INSTITUTE OF TECHNOLOGY

CHAPTER 3

Software components of embedded systems and 8051 microprocessors

Faculty: **Computer science - IT1**

MAIN CONTENT

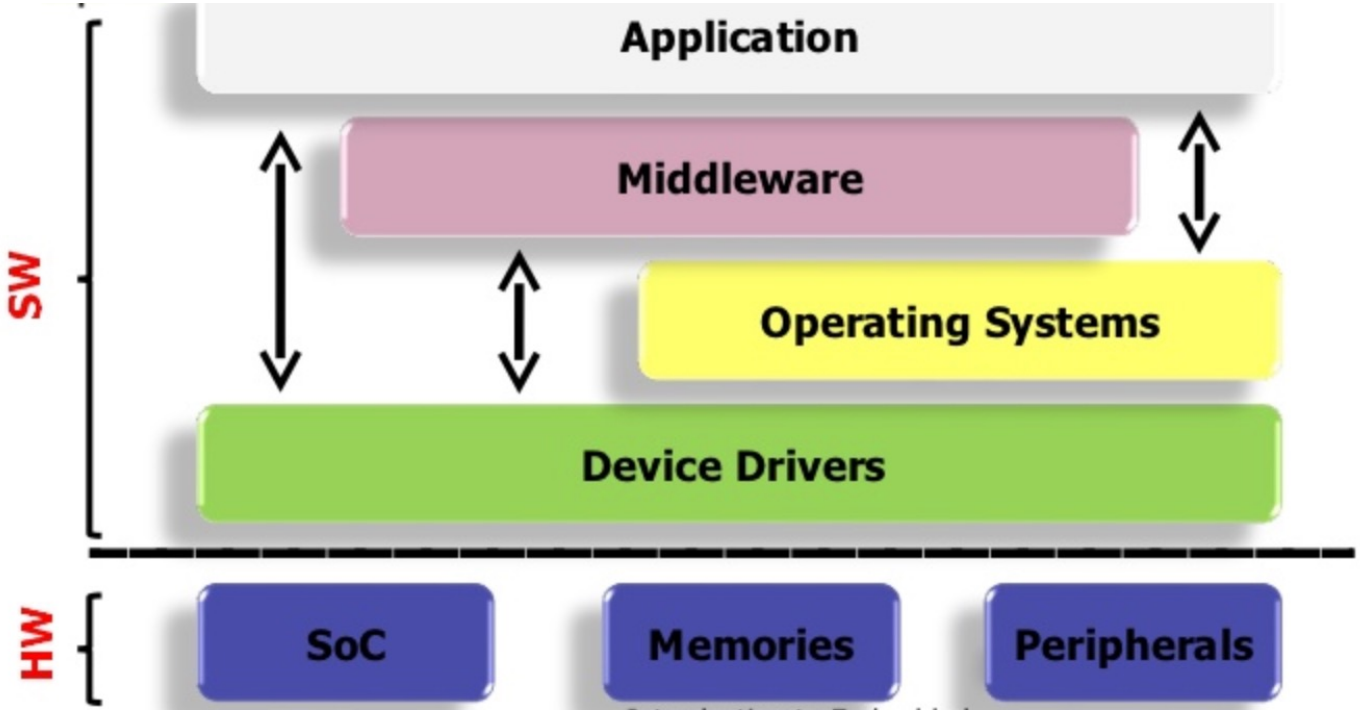
3.1. Device drivers

3.2. Real-time operating system

3.3. Middleware and application software

OVERVIEW OF EMBEDDED SYSTEM SOFTWARE

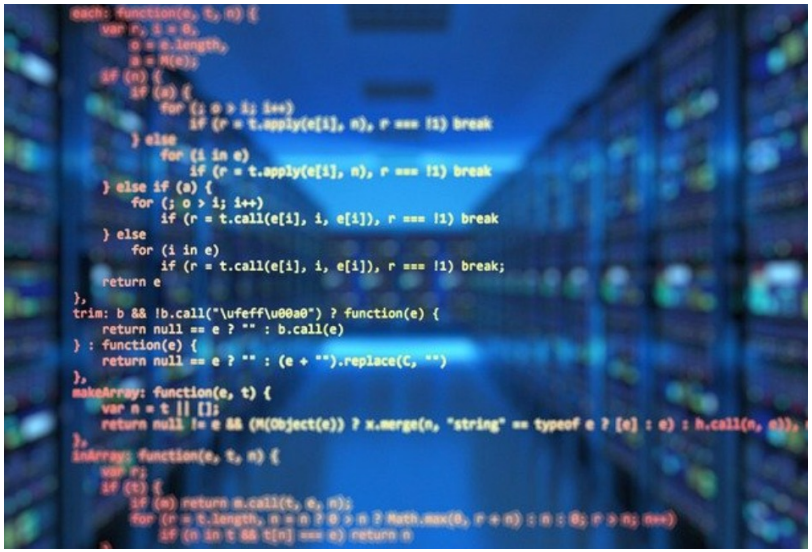
- Software Overview of Embedded Systems
- **Operating System:** System software
- **Application**: Application software



Introduction to embedded software

Introduction to embedded software

- Two approaches to building embedded software: Software design is based on conventional procedures.
  - Design based on Embedded Operating System.



❖ *Software design is based on conventional procedures.*

- Use hyperloops for applications that do not depend on system response time.
- Each procedure is executed sequentially :
  - 1. Configure shared parameters and perform initialization of hardware components such as memory and registers.
  - 2. Start with the first task and execute it.
  - 3. Execute the second task.
  - 4. Execute the next task.
  - 5. ….
  - 6. ….
  - 7. Execute the last task.
  - 8. Jump to the first task and execute in a threa
  - similar.

```
void main ()
{
Configurations ();
Initialisations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

- ***Software design is based on conventional procedures.***
  - Advantage :
    - No operating system required
    - Simple design
    - Low price
  - Disadvantage:
    - Easily affected by the entire system, if any an error task
    - Lack of processing time

```
void main ()
{
Configurations ();
Initialisations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

- ***Software design is based on conventional procedures.***

Introduction to embedded software

- ***Software design based on Embedded Operating System:***
- An operating system contains the operating system.
- Create and run applications on the operating system.
- Can be a generic operating system or a real-time operating system (RTOS)
  - General operating system:
    - unifies the embedded system with the general computing system of the operating system
    - support programming interfaces can be used
  - Real-time operating system :
    - embedded products require response time
    - contains a software responsible for preventing multitasking, planning execution schedules, multi-threading, a real-time operating system that allows flexible scheduling of system resources such as CPU and memory and output several ways to communicate between tasks

- ***Software design based on Embedded Operating System.***
  - Advantage :
    - No operating system required
    - Simple design
    - Low price
  - Disadvantage:
    - Easily affected by the entire system, if any
    an error task
    - Lack of processing time

```
void main ()
{
Configurations ();
Initialisations ();
while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

3.1 Device drivers

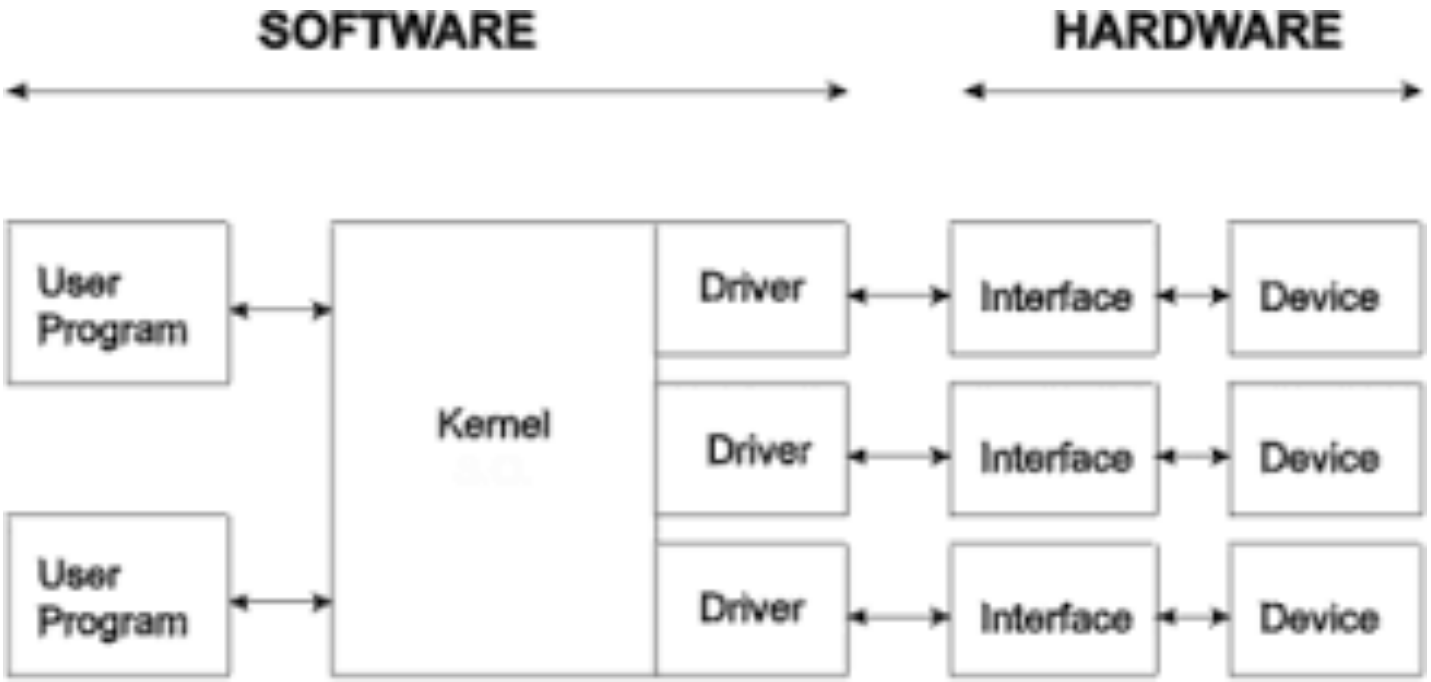
What is Driver?

- Device driver: connection between OS and I/O devices.
- Function: translate and convert requests from the OS into commands that the peripheral controller can understand.
- Is part of the OS kernel (drivers are software modules that are built separately and installed into the OS when needed).

DEVICE DRIVER DEFINITIONS (Device driver)

- Examples: printer driver, Bluetooth driver...
- A complete system has many drivers installed on the operating system.
- Device drivers operate in privileged mode: need to be well designed and secured to avoid malicious code.

DEVICE DRIVER DEFINITIONS (Device driver)



Advantages of Device drivers

1. Simplify OS operations

2. Without a device driver, the OS is responsible for communicating directly with the hardware ⇨ overloading the OS.
3. If a new device is installed, the OS must be changed.

Key design issues

1. OS / Driver Communication
   - Information exchange(command, data)
   - Support functions that the kernel provides
2. Driver / Hardware Communication
   - Information exchange(command, data)
   - Software communicates with Hardware
   - Hardware communicates with Software
3. Driver Internal Operations
   - Translate command received from OS
   - Arrange the order of requirements
   - Manage data transmission between interfaces(OS và hardware)
   - Accept and process interrupts
   - Maintain consistency of data structures between driver and kernel.

Types of Device Drivers

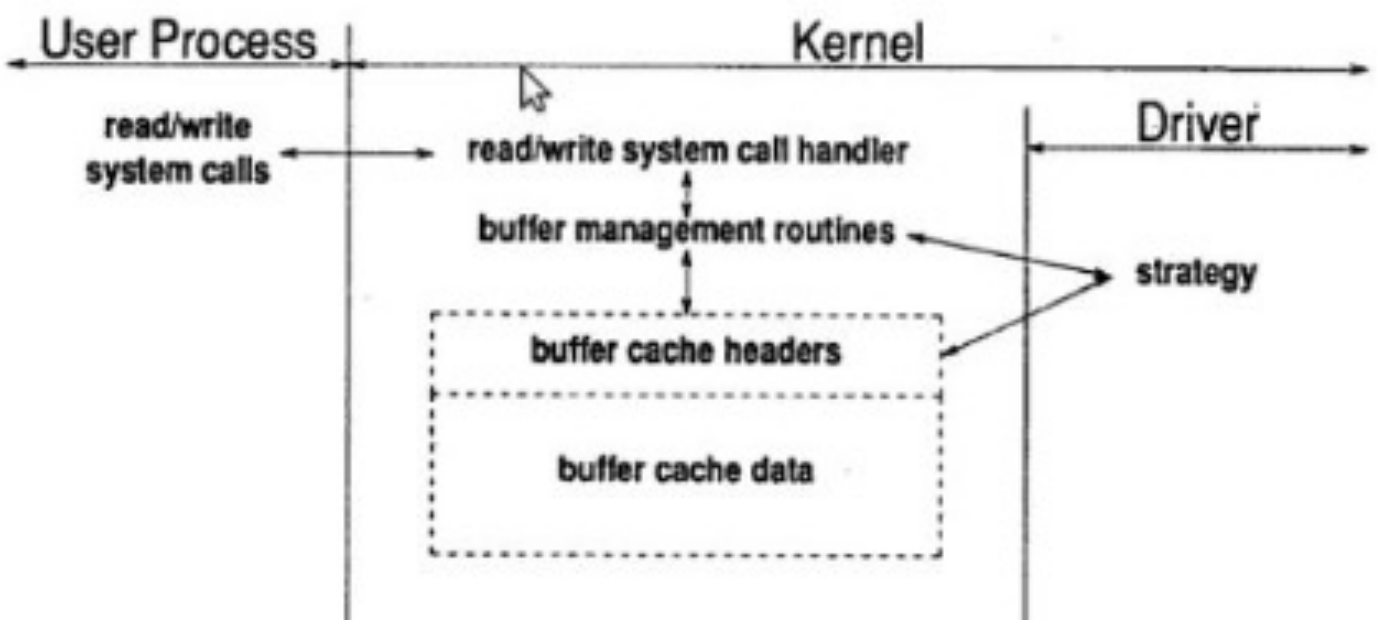
- Block Driver
- Character Driver
- Terminal Driver
- Stream Driver

Block Drivers

- Communicates with the OS through a set of fixed-sized buffers (block of data).
- The OS manages the cache of caches and responds to user requests for data by accessing the cache.
- The driver is only called when the requested data is not in the cache or when the cache changes.
- The driver works with requests from the OS to fill or clear buffers of a fixed size.

Block Drivers

❖ Ví dụ: disks are applied as block devices.

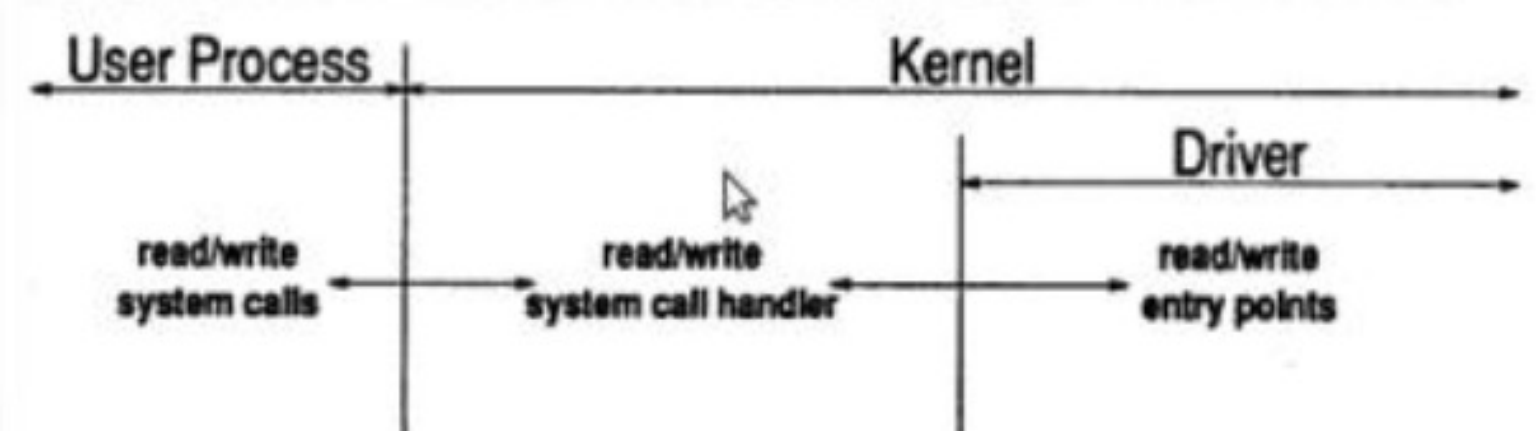


Character Drivers

- Requests with packet sizes of any length of I/O.
- Used for devices that communicate bytes at a time or with data blocks larger or smaller than the block drivers' data buffer.

  VD: printer
- I/O requests are passed directly to the driver, and the driver transfers data to VXL memory.

Character Drivers



Block driver vs Character Drivers

Block Driver	Character Driver
Transfer data to the kernel buffer cache	Pass data directly to the user process
Not suitable for devices communicating with variable size packets	Suitable for devices that communicate in packets of variable size
Can be used to support UNIX-based devices	UNIX-based devices are not supported
Suitable for supporting drives and hard drives	Suitable to support serial port, parallel port, network card, keyboard...

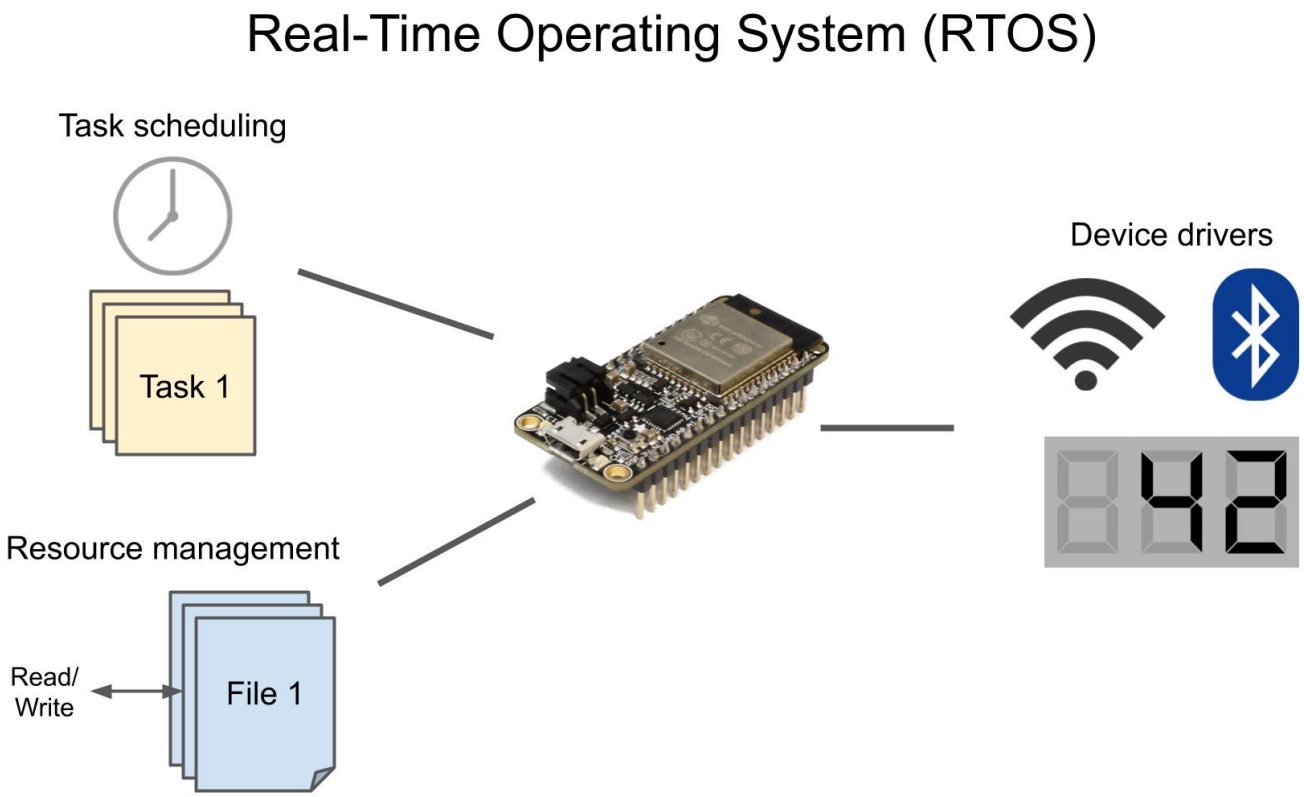
Characteristics of a device driver

- Driver is a set of entry points called by the OS.
- One driver includes:
  - Separate data structure for each device.
  - Separate protocol for each driver
- Most drivers are written in one source file.
- The first part of the driver is called the prologue declaration.

Characteristics of a device driver

- Prologue includes: :
  - #include
  - #define
  - Declare variables and their characteristics
- The rest of the driver :
  - Entry points (C functions are referenced by the OS)
  - Routines (C functions interact with the driver)

3.2 . Real Time Operating System (RTOS)



II. Real Time Operating System (RTOS)

- Definition: is a multi-tasking operating system for applications with real-time constraints.
- Real-time constraints: require a certain delay for an event and system response.

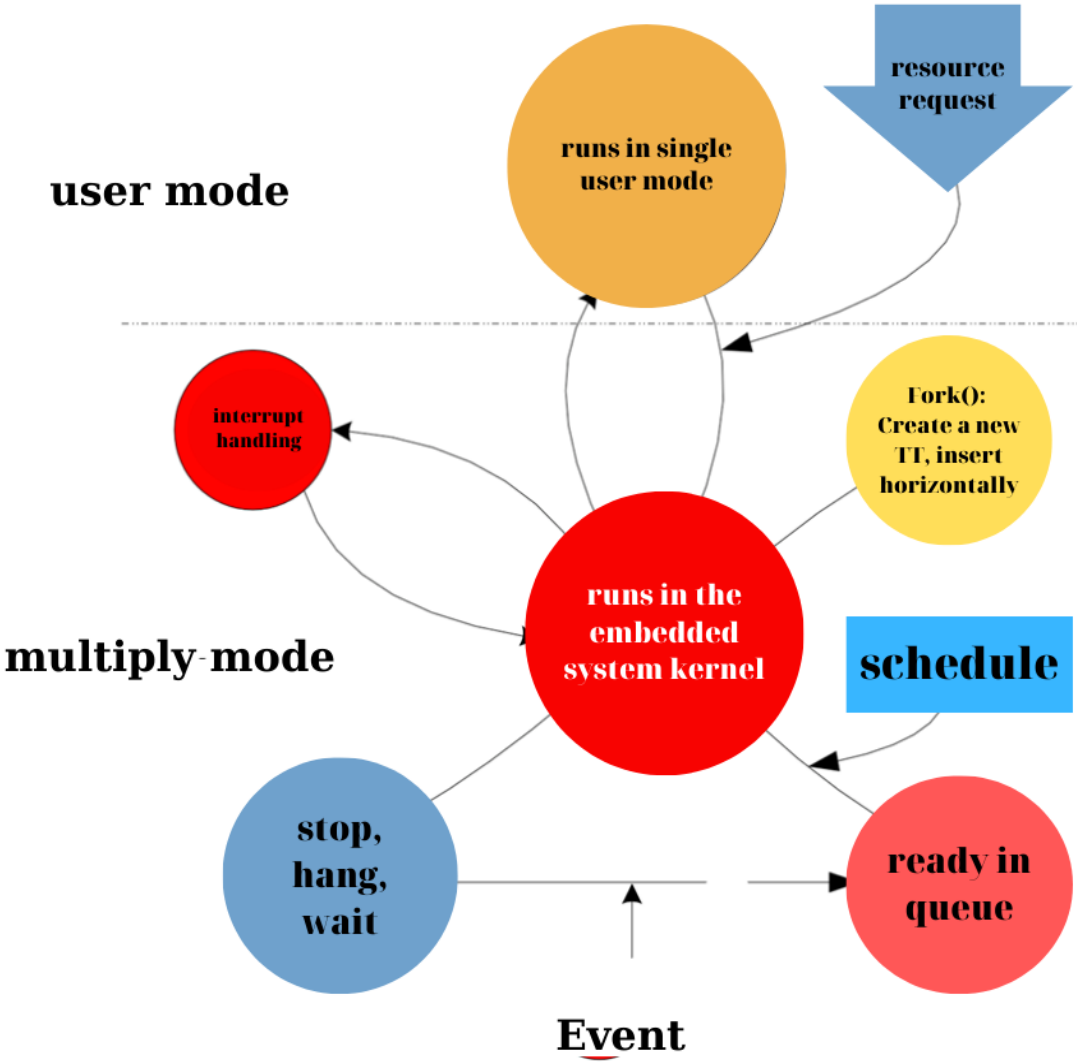
Benefits of Real Time Operating System (RTOS)

- Divide the project into smaller parts to "treat".
- Easily divide work in group work.

Basic functions

- Kernel
- Process management
- Resource management
- Equipment management
- Manage I/O peripheral device systems
- Manage network devices

Process management



Hierarchy of priority processes

- Hierarchy of user priorities, called static priority hierarchy, or real-time priority hierarchy.
- Real-time priority hierarchy is higher than static priority hierarchy and idle priority hierarchy.
- Idle priority tasks run when there are no high priority tasks running.

Managing progress through intervention

- The RTOS interrupts a low-priority process when a message or event is waiting for the high-priority process.
- The RTOS has intervention points at the end of critical code, and thus the RTOS can be intervened at these points by real-time high priority tasks.
- A small portion of RTOS functions are non-interferable.

Process management by inheriting priority levels (priority inheritance)

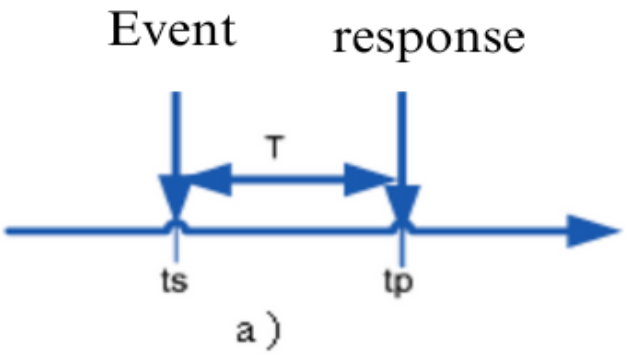
- Priority inheritance allows sharing resources with low priority tasks.
- A medium priority task will not be able to interfere with a low priority task while it is locked to run with the high priority task.
- Helps improve system performance.

Design of RTOS

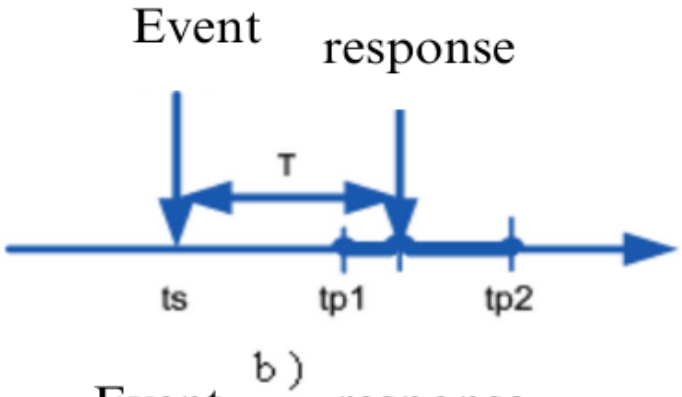
- An embedded system with a single CPU can run only one process at a time.
- A process at any given time can be run by an ISR, a kernel function, or a task.
- Run threads in the kernel to increase processing speed.

RTOS design

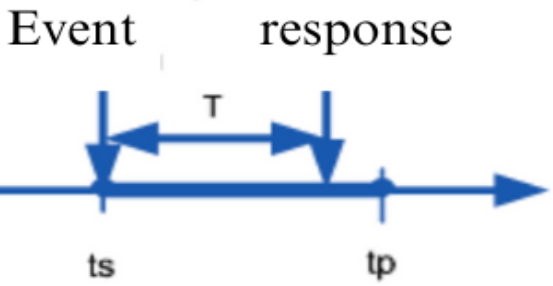
- Provides efficient handling of ISRs, drivers, ISTs, streams...
- Turn interrupts on and off
- Allocate and deallocate memory at fixed times and memory blocks.
- Provides efficient scheduling, running, and stopping of tasks in case of multiple tasks.


Event
response
T
ts
tp

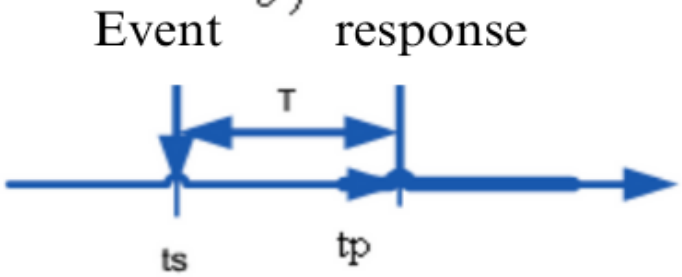
a)


Event
response
T
ts
tp1
tp2

b)


Event
response
T
ts
tp

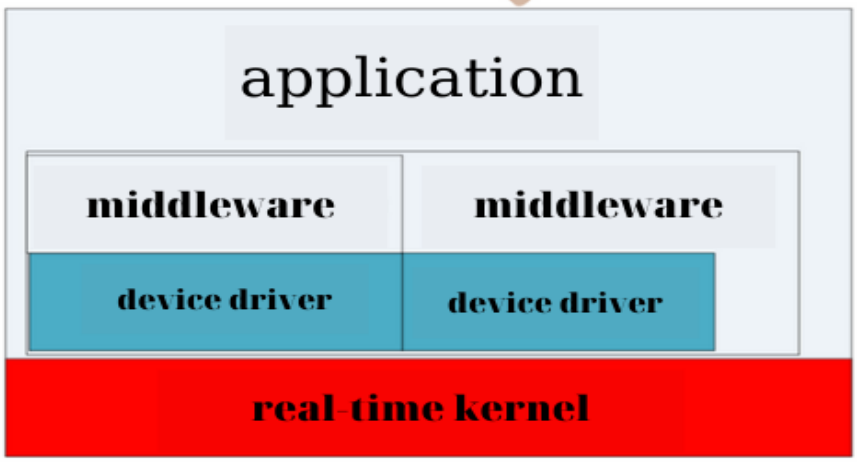
c)

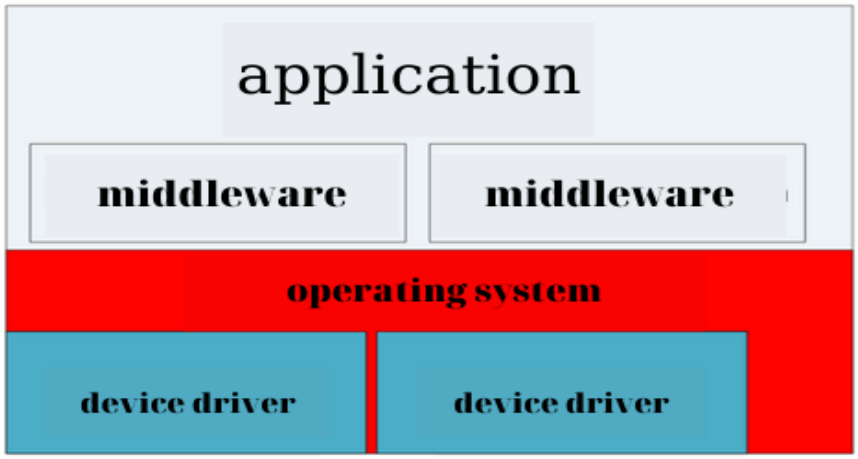

Event
response
T
ts
tp

d)

RTOS design

- Managing I/O (devices, files, mailboxes...) becomes easy with RTOS
- Effectively manage multiple states of CPUs, devices such as internal and external hard drives, virtual devices...
- An embedded system running an OS makes working with multiple inputs simpler (through drivers) and processing data easier.

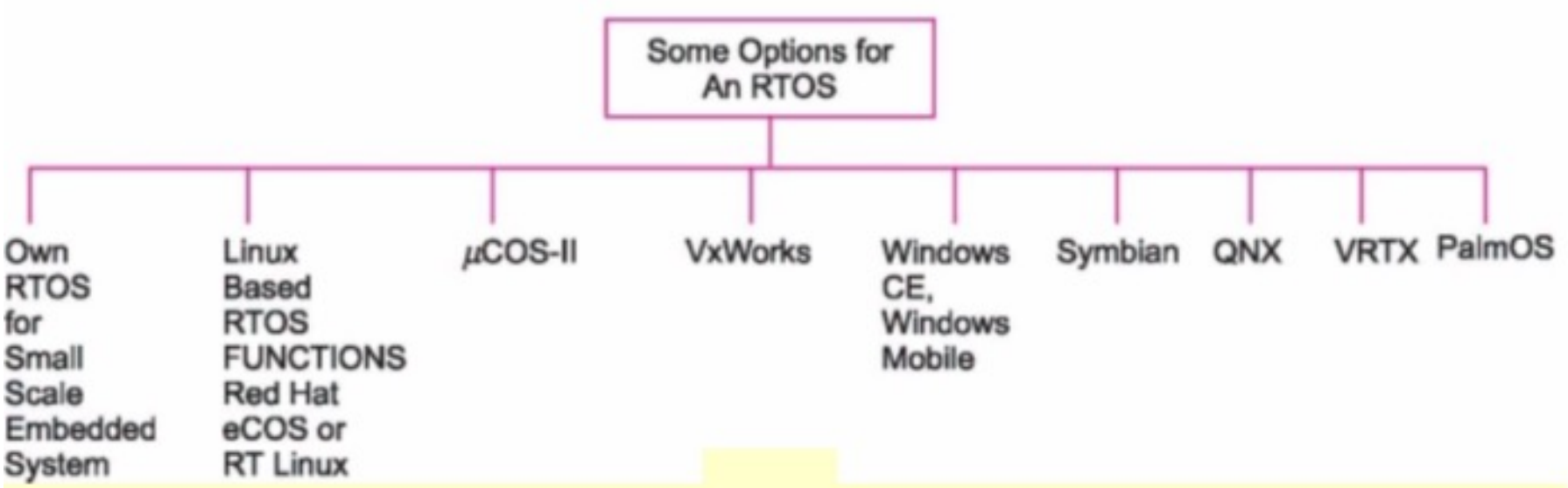

application
middleware
middleware
device driver
device driver
real-time kernel


application
middleware
middleware
operating system
device driver
device driver

RTOS design

- Managing I/O (devices, files, mailboxes...) becomes easy with RTOS
- Effectively manage multiple states of CPUs, devices such as internal and external hard drives, and virtual devices.

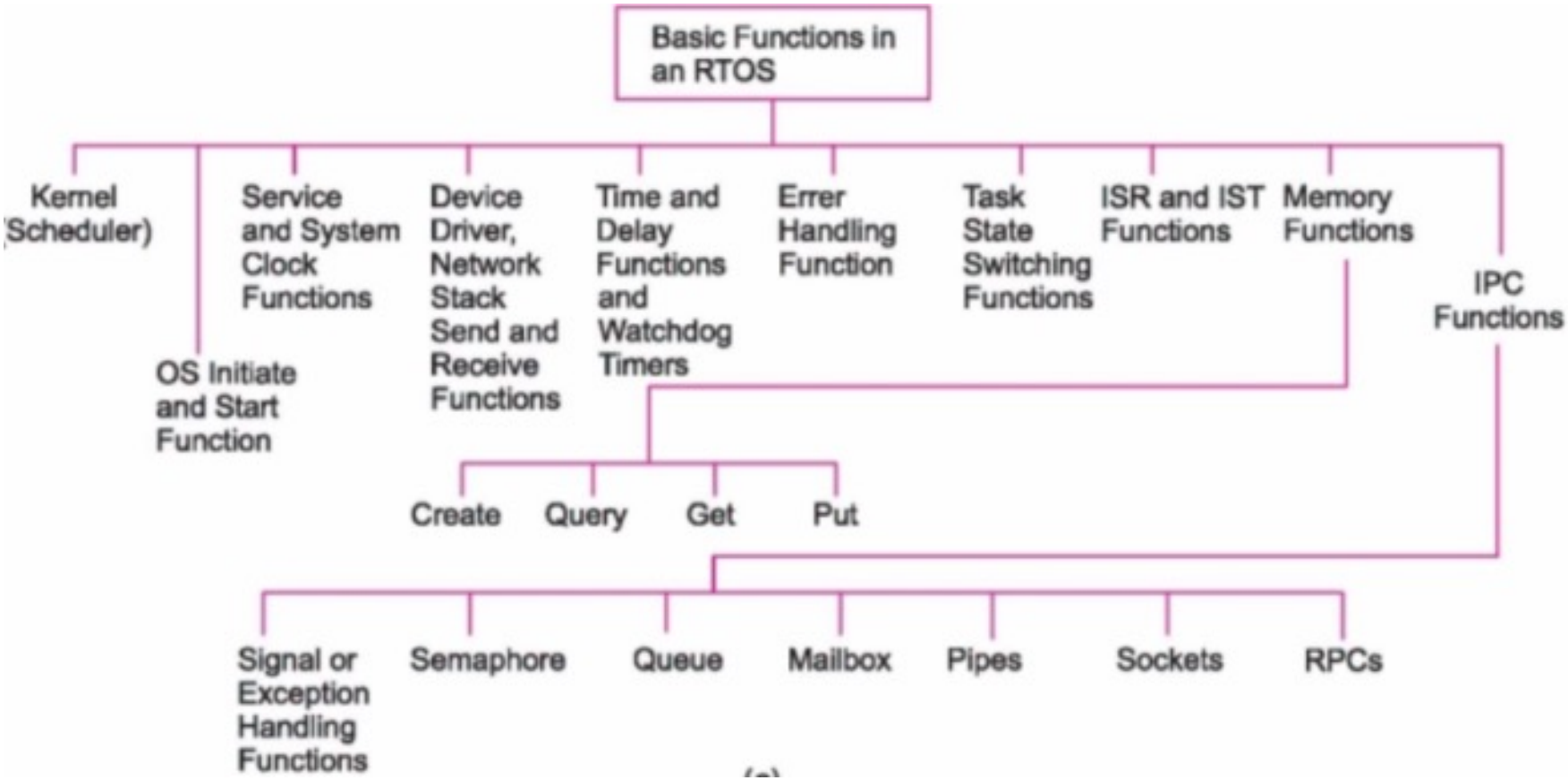
Options for RTOS



Complex multitasking embedded system design requirements

- IDE
- C/C++ multitasking functions
- Real-time clock – timers
- Schedule(scheduler)
- Device drivers, device manager
- Communication functions between processes, multi-threaded processing functions...
- Other functions : TCP/IP, USB port, network
- Testing and debugging software for RTOS testing

Basic functions in RTOS



Classification of RTOSes

- In-house Developed RTOSes

1. Codes are written for specific application or product requirements.
2. Requirements can be refined.

Classification of RTOSes

- Commercial RTOSes

1. Provide testing and debugging tools.
2. Supports many VXL architectures (ARM, x86...)
3. Support GUIs
4. Supports communication with many devices and many different connection protocols.
5. Optimized software and device support.
6. Simplify the developer's coding process.
7. Speed up product development.
8. Save time and maintenance costs.

Classification of RTOSes

- Commercial RTOSes

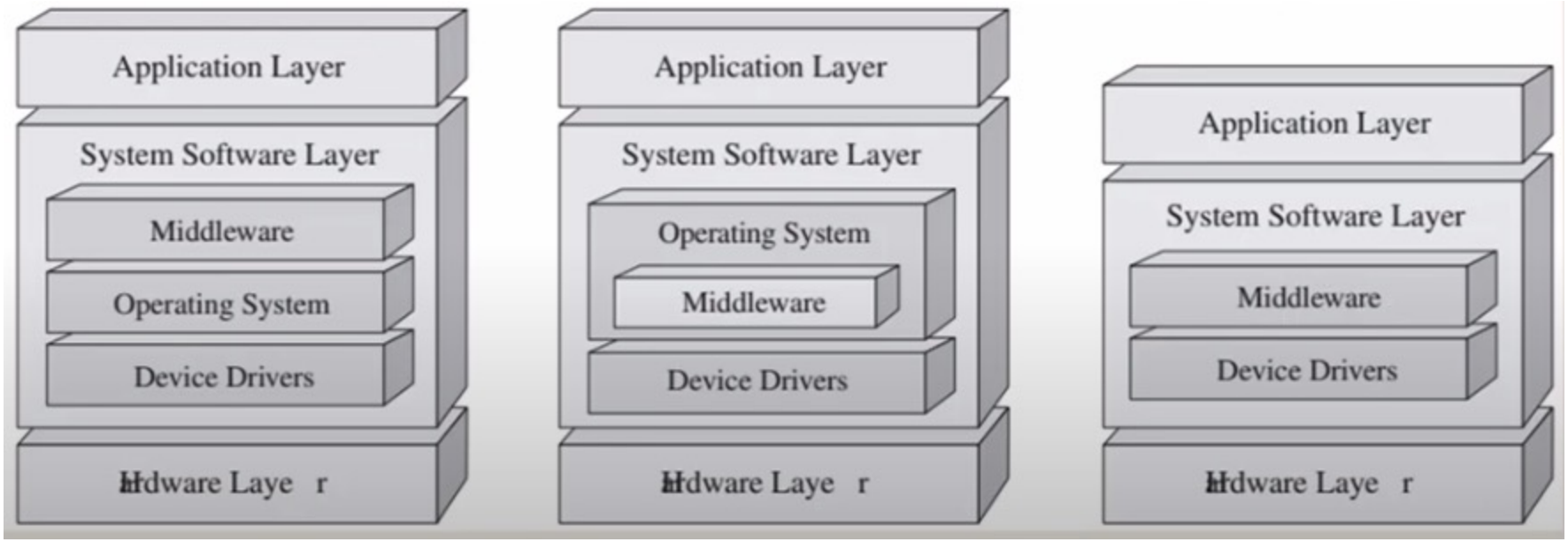
1. Provide testing and debugging tools.
2. Supports many VXL architectures (ARM, x86...)
3. Support GUIs
4. Supports communication with many devices and many different connection protocols.
5. Optimized software and device support.
6. Simplify the developer's coding process.
7. Speed up product development.
8. Save time and maintenance costs.

Classification of RTOSes

- General purposes OS with RTOS

1. Linux, Windows OS.
2. Has a strong GUI configuration, many multimedia interfaces, and low price.
3. Supports GUIs

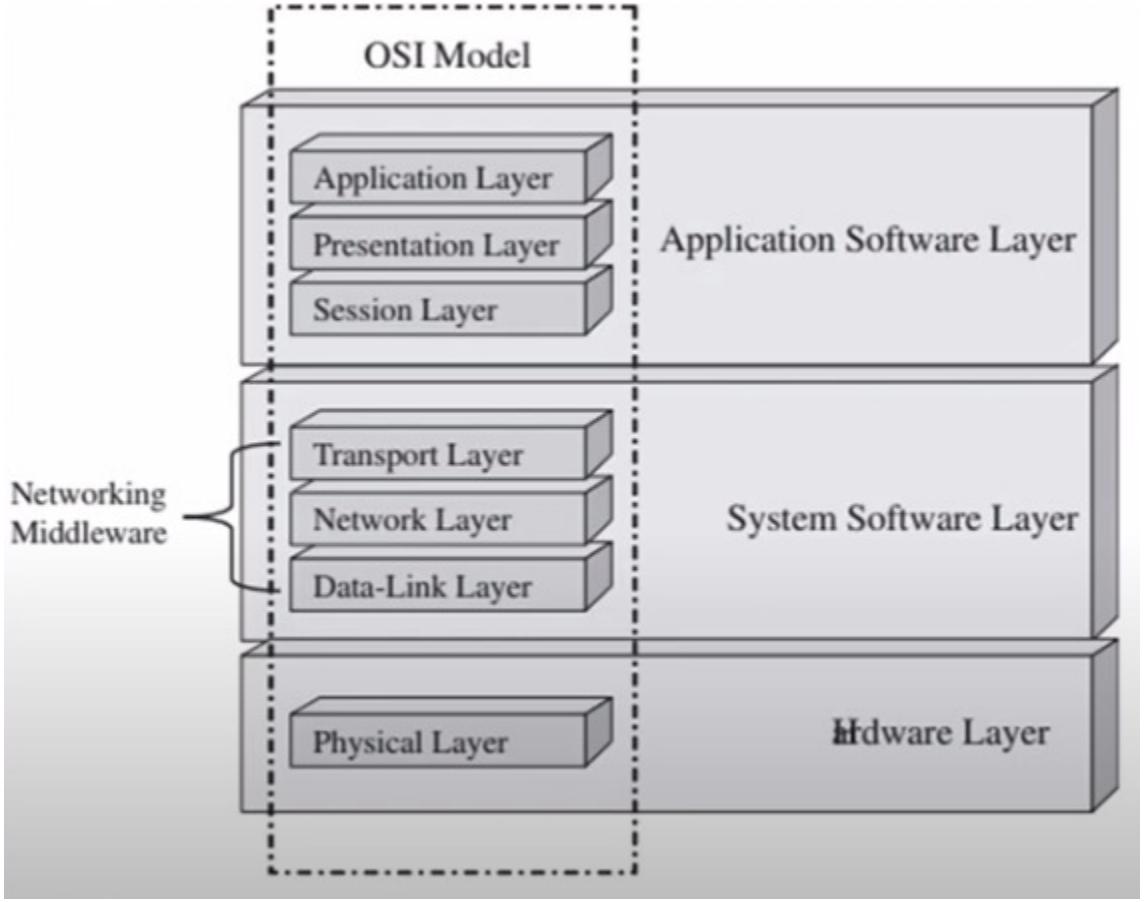
1. Middleware: located in many places, separate from the app, OS, and drivers.
2. Application: The software sits at the top layer, communicating with the user.

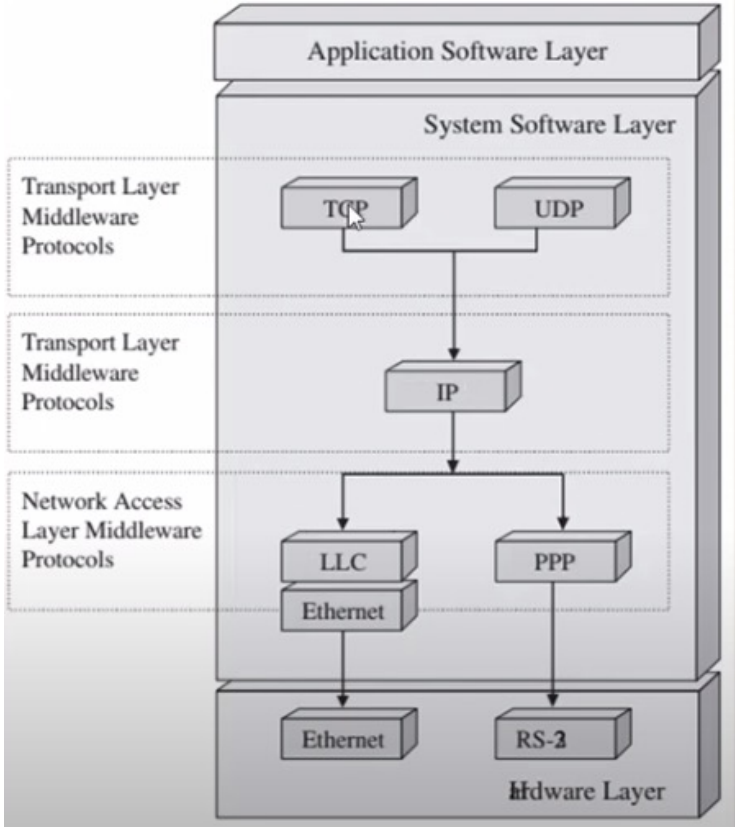


Middleware

1. Middleware: considered middleware, ensuring flexibility, security and helping to link software at adjacent layers.
2. Contains service functions, which are called repeatedly by other software.

Middleware (ví dụ)





What does embedded software development require?

1. Compilation software
2. Development board
3. Loader/Debugger
4. Libraries (for ARM core, peripheral drivers...)
5. Documents (datasheet)

Steps to compile software in IDE (translation software)

1. Create project
2. Configure the compiler software (choose chip type, select loader circuit type)
3. Write programs and compile
4. Load into the flash memory of the development board
5. Run the program and check for errors

POST AND TELECOMMUICATIONS INSTITUTE OF TECHNOLOGY

CHAPTER 4

Design and install embedded systems

Faculty: **Computer science – IT1**

MAIN CONTENT

4.1. System design

4.2. Test installation

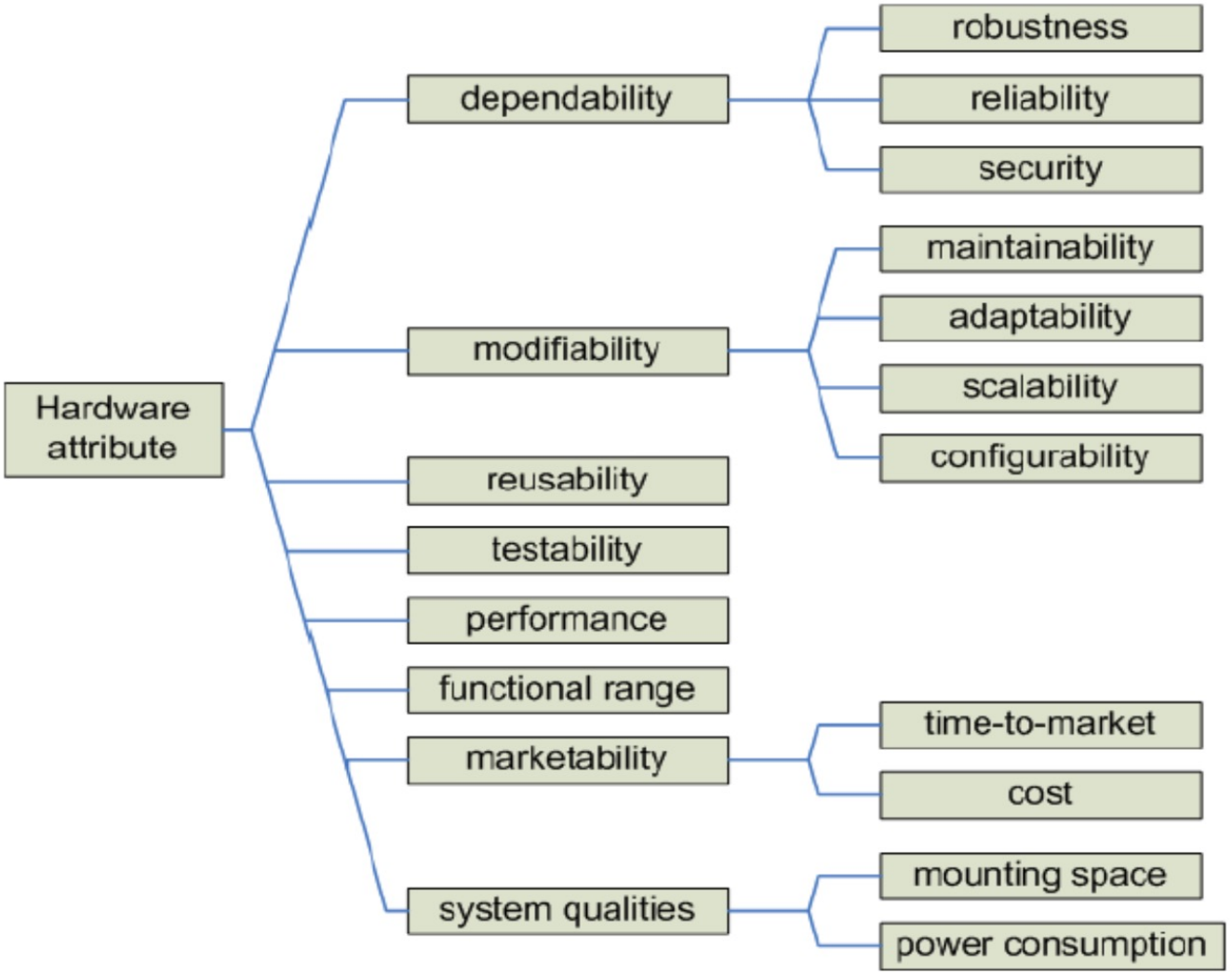
4.3. Complete product design

OVERVIEW OF EMBEDDED SYSTEM DESIGN

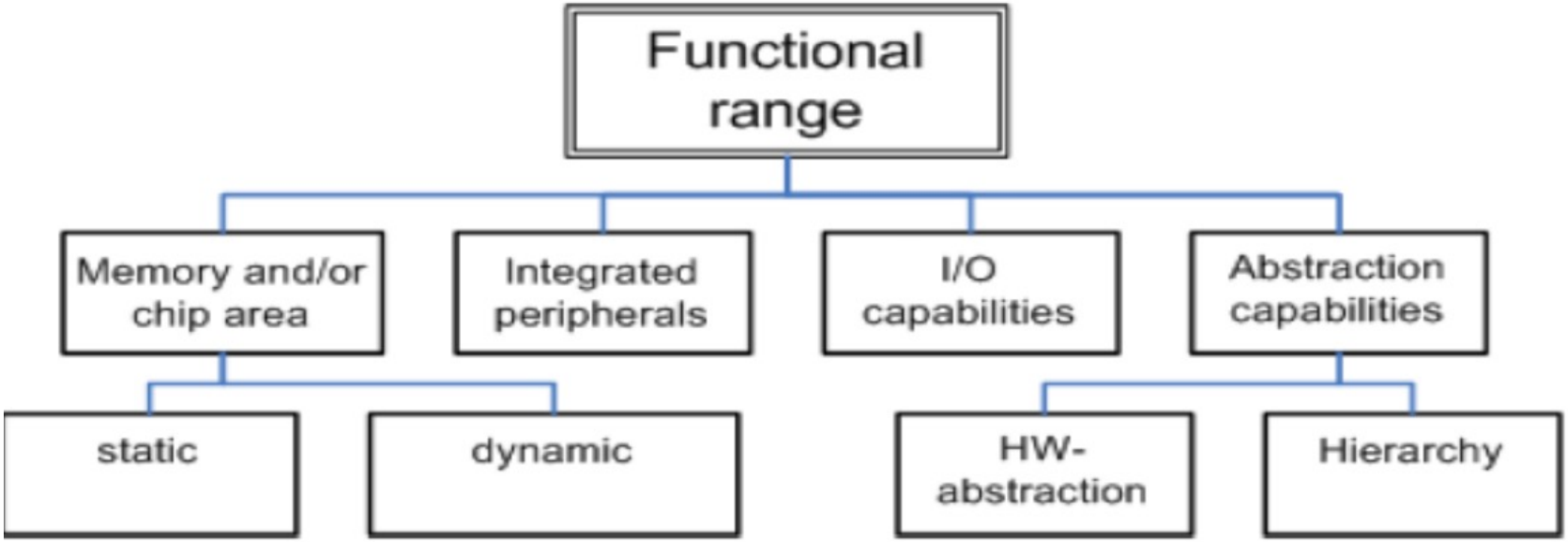
- Points to consider when designing HTN include:
- A system with components:
  - Processor
  - Memory
  - Peripherals
- Integrated system:
  - Microcontroller
  - Expanded microcontroller
  - Microprocessor based
  - Board based
- Software:
  - System software
  - Embedded application software, embedded application algorithms.

4.1 SYSTEM DESIGN

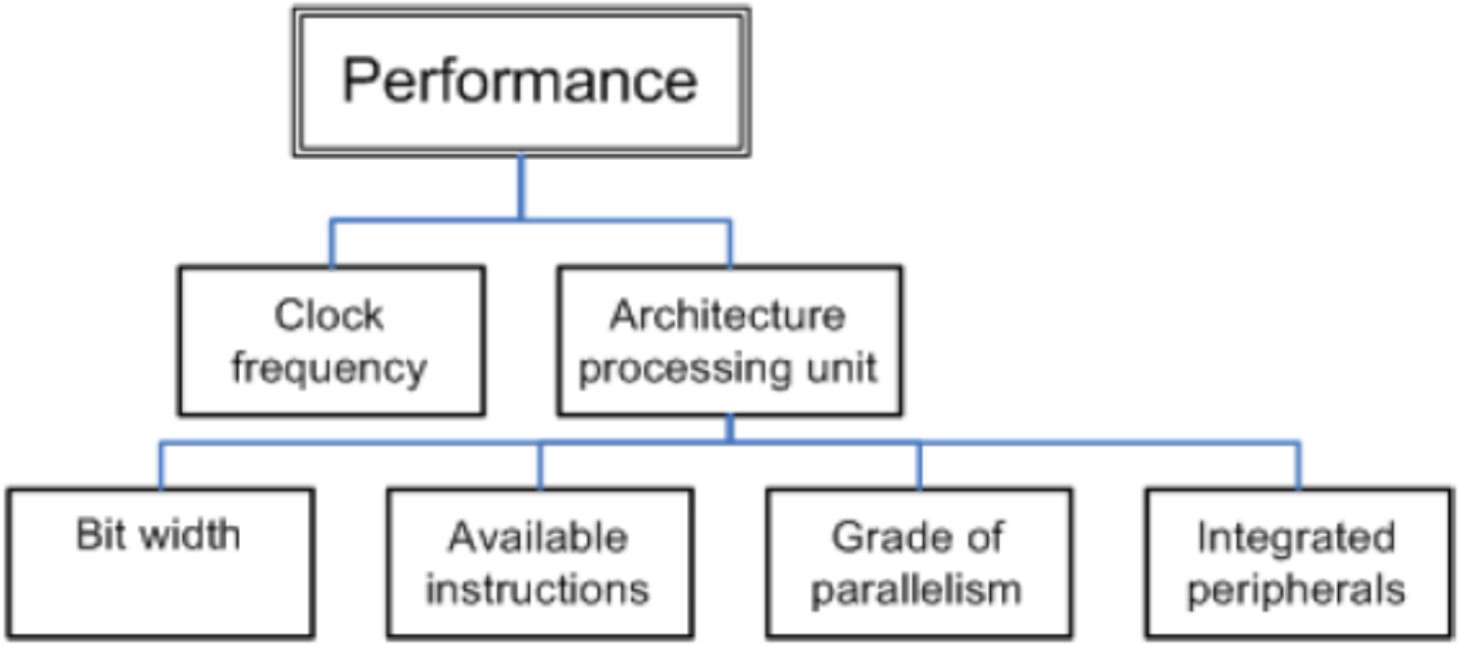
System design



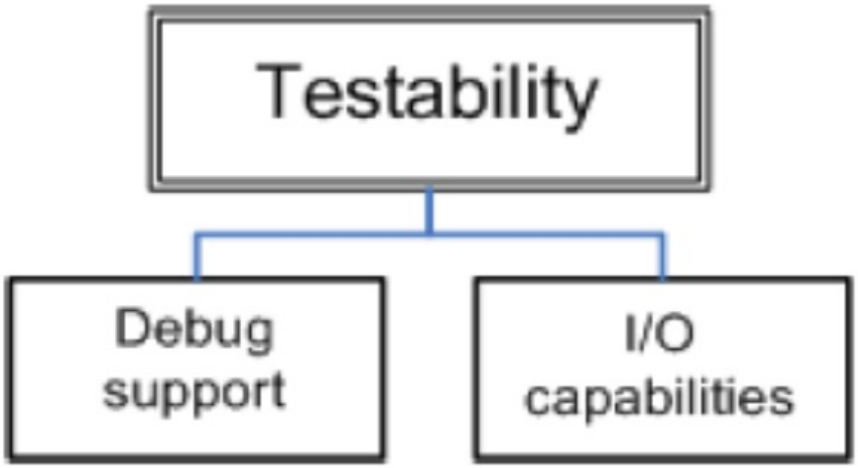
System design



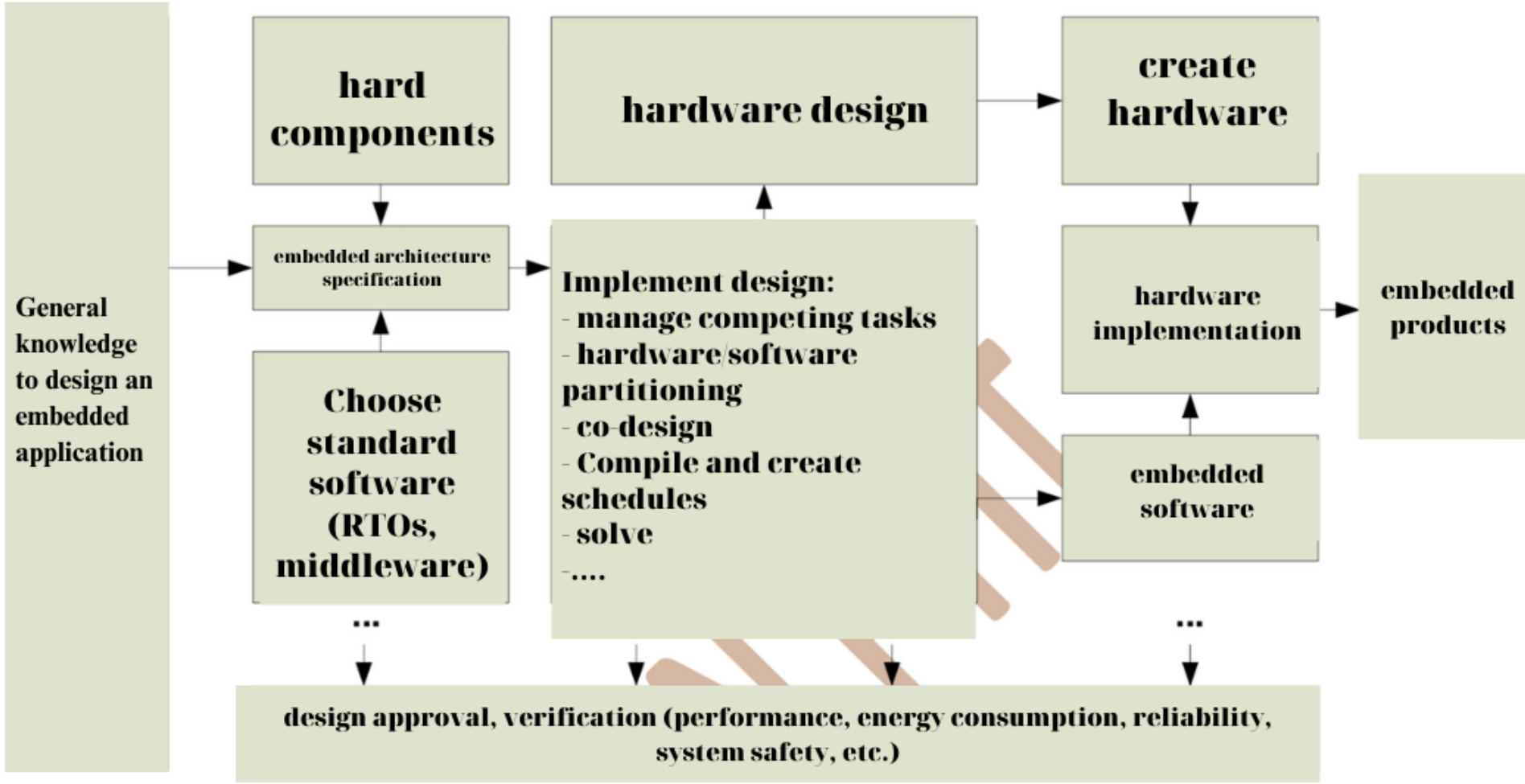
performance attributes



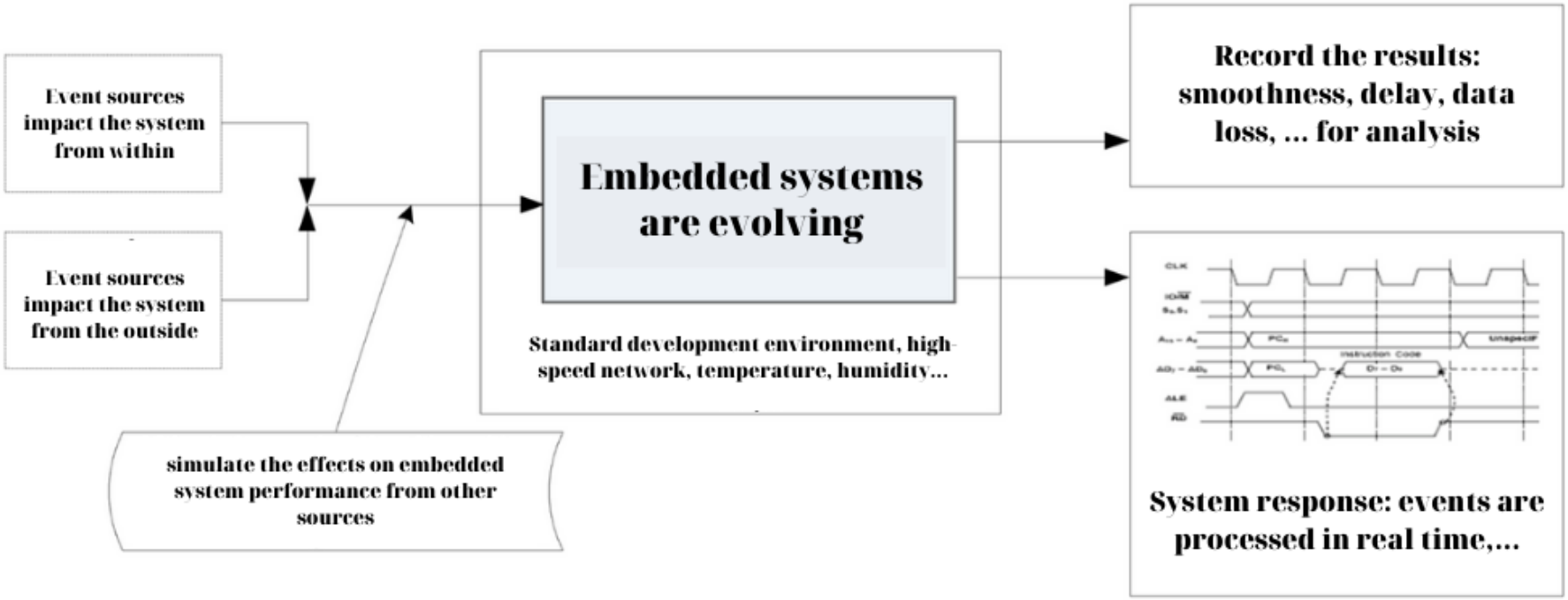
hardware testable properties



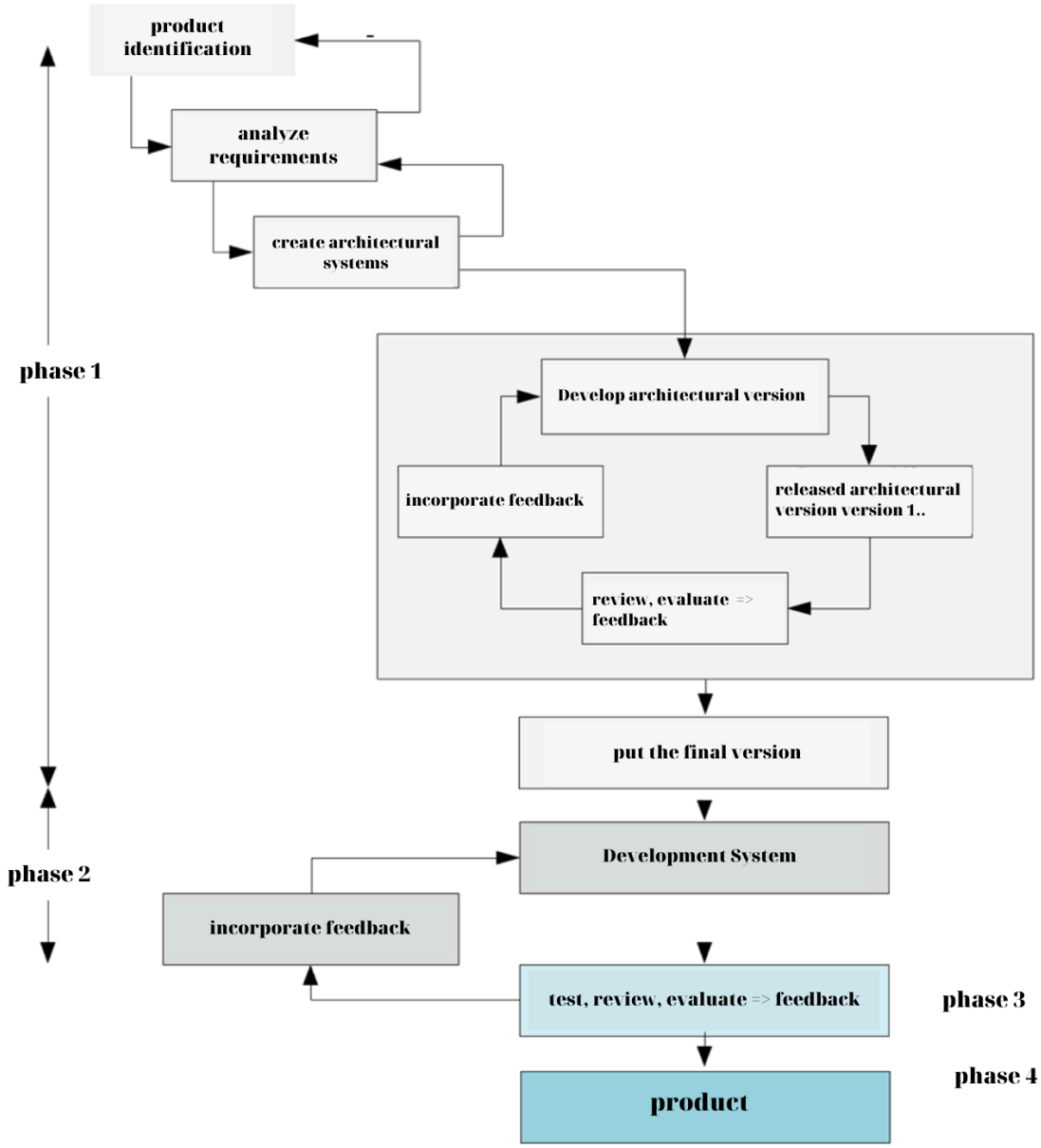
System design



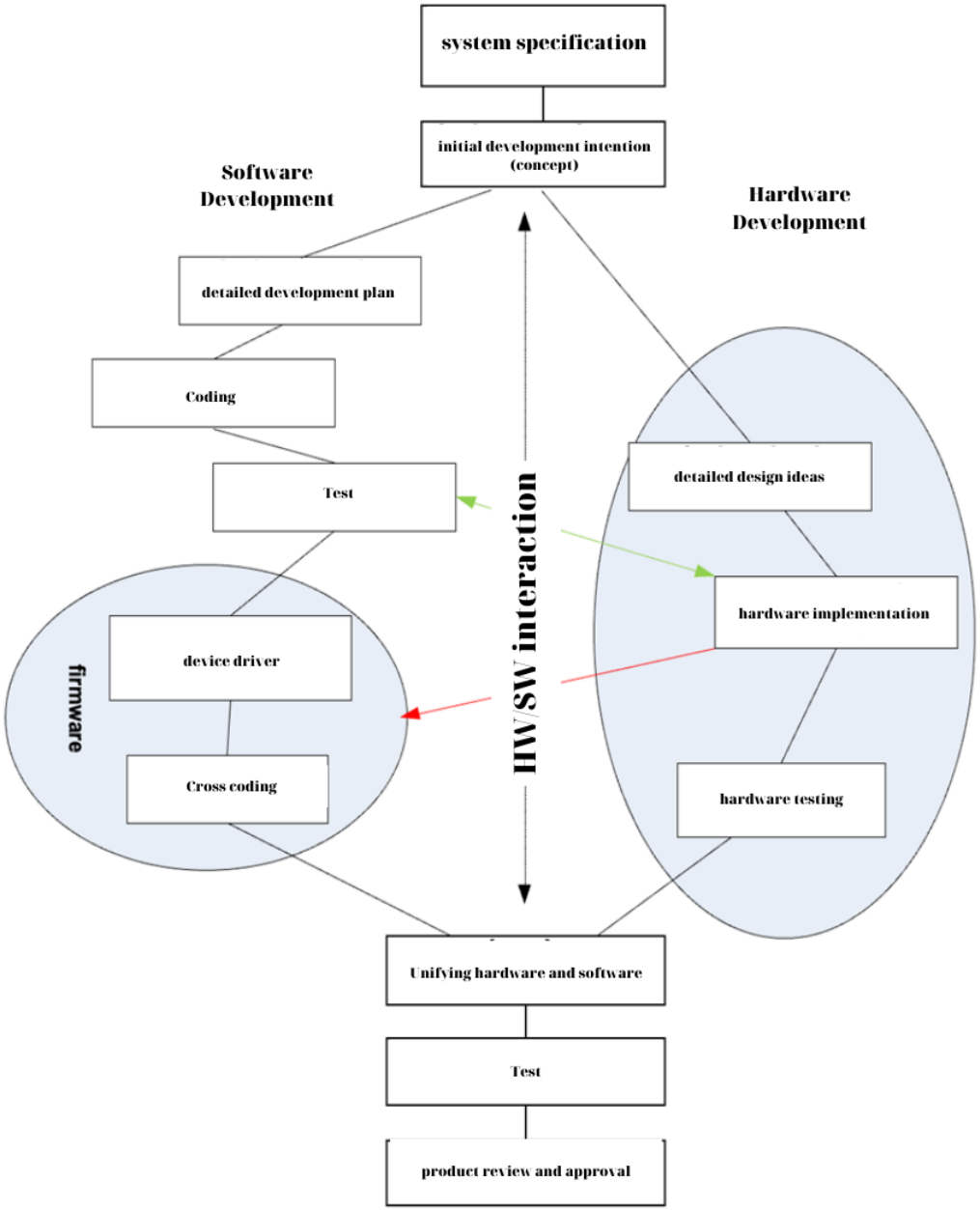
Example of building an embedded system performance scenario :



Embedded System Design Phases



Hardware and software design planning



Design rules

- Knowledge base for HTN design: Computer science and electronic engineering
- Circuit design with Verilog or VHDL hardware language (FPGA, ASIC);
- Technological capabilities and hardware technological limitations.
- The design must target specific application objects with the following trends:
  - Increased program code size: 16 – 64 KB up to 64kB to 512 KB
  - Reusing hardware (CPUs, micro-controllers, DSPs) and software (device drivers) components,
  - There is high integration in one system (DSP, network, RF, 32-bit CPU, Intelligent Input/Output-I2O type IO processors).
- Use available software, reusable software, open source code.
- Programming technology (programming languages, software development systems);
- Circuit design (VLSI, ASIC format), electronic system design (digital, analog);
- Real-time processing system (hard real-time, soft real-time).

Design steps

- Building Embedded System specifications and modeling Embedded systems will design and experiment with algorithms
- Gather and describe basic hardware: base connection, communications, microelectronics application computing technology, memory technology, devices connected to the system.
- The software system will include: device control, middleware, operating system, application software
- Partition and select parts of the design: hardware, software
- Use design simulation tools to run hardware and software simulations
- Critical software and hardware (time constraints) require testing and adjustment
- Test on hardware board (prototype) with selected CPU.
- Troubleshoot and tweak hardware and software;
- Complete the product.

4.2 INSTALLATION AND TESTING

Install and test

❖ *Select CPU for design*

- Based on analyzing HTN requirements => Determine CPU features
- Criteria for CPU selection:
  - What types of peripherals (sensors) will the EMBEDDED SYSTEM connect to?
  - How many programs and data space is needed for the system?
  - How many interrupts does the system need?
  - How many input/output ports will the design use?
  - Which type of processing has the most critical-time constraints that the CPU must perform?
  - What types of development tools (for both hardware and software) are available for the CPU to choose from?
  - In the case of CPU on board, what is the actual cost?
  - Are there any types of available devices that can be integrated into the current system?
  - How will the software work with the hardware and vice versa?….

Install and test

❖ *Select Memory for design*

- Two issues in choosing Memory for embedded systems:
  - Memory type and operating control technique,
  - Content protection mechanism (separation between system memory segments for the Operating System and segments for applications).
- Criteria for selecting Memory for embedded systems:
  - Memory technology
  - CPU compatible

❖ *Pair the device*

- Pairing devices in embedded systems includes:
  - Similar information(analog)
  - Digital information is digitized using functional microchips(ADC/DAC).
  - Data communication
  - Connection techniques (serial, parallel, DMA...)

Install and test

- ***Software Development***
  - Write source code for EMBEDDED SYSTEMS
  - Operating system for EMBEDDED SYSTEMS
  - Download software to hardware
- ***Troubleshooting and simulation***
  - Low level simulation
  - Troubleshooting on the board
  - Troubleshooting at the task level
  - Troubleshooting symbols
  - Optimize code
  - Xray –Look for radiographic errors
  - System development and simulation techniques
  - …

❖ *Embedded System development example*

1) Board design with options:
   - CPU Intel 8085/8086,
   - ROM, RAM
   - Peripheral microcircuits : 74257, 74 244, 74245, 7474, 8253, 8255, 8237, UART 8250/16450/16550
2) http://www.beyondlogic.org/serial/serial2.htm
3) Developed HTN with Intel 8051 micro controller with KEIL Soft software.
4) EMBEDDED SYSTEM with PIC :

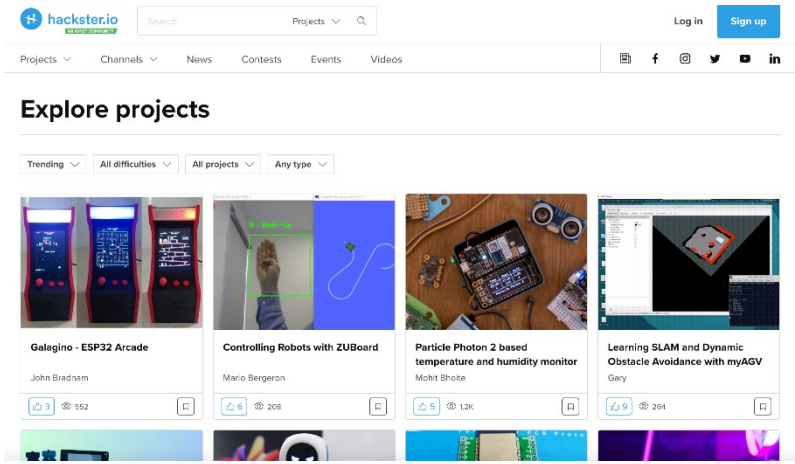
   http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2123&param=en022497

   Currently on the market there is a board with PIC 16F877 with Development System software for sale for 900,000.00 VND, suitable for practicing writing application programs. Once proficient, you can design hardware with discrete microchips and develop specific applications from simple to complex.

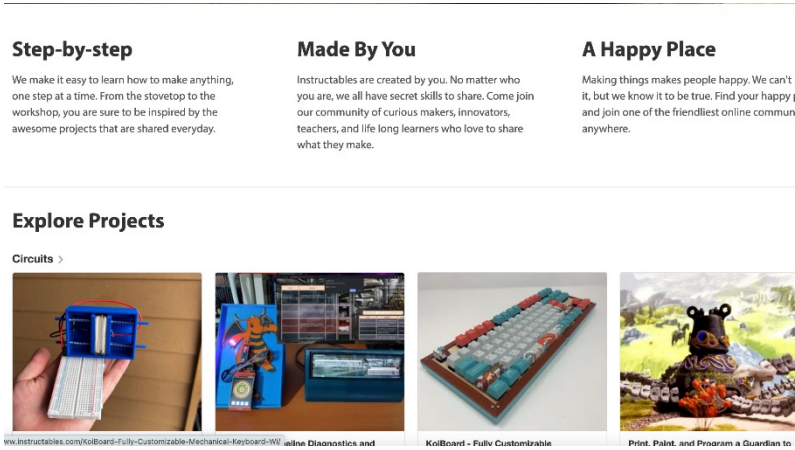
Install and test

- ***Embedded System development example***

1) Hackster.io –Develop embedded system projects:https://www.hackster.io/projects



2) Instructables.com – Community for people who enjoy making and implementing embedded systems



4.3 COMPLETE PRODUCT DESIGN

Design and develop product designs

- Product design needs attention :
  - Product aesthetics
  - Size, weight, style and color
  - Select materials for product shells
  - Protection against external impacts such as drops and water.
  - Limit the cost of the product

Embedded product development lifecycle

- Embedded product development lifecycle
- EDLC - embedded product development life cycle : “analysis-design-installation”