


# EC619 Embedded System

*Presented By*

Dr. Deepa Sharma

# Books

- ▶ Computers as components – principles of embedded computing system design, Wolf & Wayne, Second Edition, Morgan-Kaufmann.
  - ▶ Embedded system design – a unified hardware / software introduction, F. Vahid & T. Givargis, John Wiley.
  - ▶ Embedded systems, Raj Kamal, Tata McGraw Hill.
  - ▶ Introduction to embedded systems, K. V. Shibu, 2nd Ed., Tata McGraw Hill.
- 

Thank You

In class 1 generalized discussion on  
embedded system what are embedded system?

# Introduction to Embedded Systems

## **Several Definitions:**


### **Wayne Wolf:**

- ▶ Any device that includes a programmable computer but is not itself intended to be a general purpose computer.

### **Jonathan W. Valvano:**

- ▶ An embedded computer system includes a microcomputer with mechanical, chemical and electrical devices attached to it, programmed for a specific dedicated purpose, and packaged as a complete system.

### **Raj Kamal:**

- ▶ An embedded system is one that has computer-hardware with software embedded in it as one of its most important component.
- 


## **K.V. Shibu**

- ▶ An embedded system is an electronic/electro-mechanical system designed to perform a specific function and is a combination of both hardware and firmware (software).

## **Lyla B. Das**

- ▶ An embedded system is an electronic system which is designed to perform one or a limited set of functions, using hardware and software.


**In a nutshell: Embedded Systems => Computers inside a product (electronic devices).**




# EMBEDDED SYSTEMS VS. GENERAL COMPUTING SYSTEMS

General Purpose Computing System	Embedded System
A system which is a combination of a generic hardware and a General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OS for executing a specific set of applications
Contains a General Purpose Operating System (GPOS)	May or may not contain an operating system for functioning
Applications are alterable (programmable) by the user (It is possible for the end user to re-install the operating system, and also add or remove user applications)	The firmware of the embedded system is pre-programmed and it is non-alterable by the end-user (There may be exceptions for systems supporting OS kernel image flashing through special hardware settings)
Performance is the key deciding factor in the selection of the system. Always, 'Faster is Better'	Application-specific requirements (like performance, power requirements, memory usage, etc.) are the key deciding factors
Less/not at all tailored towards reduced operating power requirements, options for different levels of power management.	Highly tailored to take advantage of the power saving modes supported by the hardware and the operating system
Response requirements are not time-critical	For certain category of embedded systems like mission critical systems, the response time requirement is highly critical
Need not be deterministic in execution behaviour	Execution behaviour is deterministic for certain types of embedded systems like 'Hard Real Time' systems

# Desirable Features and General Characteristics of Embedded Systems

- ▶ It should have one or a small set of functions which it is expected to perform efficiently.
  - ▶ It should be designed for low-power dissipation, because many systems are battery powered.
  - ▶ It has limited memory and limited number of peripherals.
  - ▶ Applications are not meant to be alterable by the user.
  - ▶ Many of them are not accessible directly, that is, they may be part of the control unit of a larger system, so no interference in operation is possible.
  - ▶ They need to be highly reliable.
  - ▶ Many of them need to operate with time constraints.
- 

# Application Areas of Embedded Systems

- ▶ Consumer electronics: Cameras, music players, TVs, DVD players, microwave ovens, washing machines, refrigerators and remote controls.
  - ▶ Household appliances/home security systems: Air conditioners, intruders and fire alarm systems.
  - ▶ Automobile controls: Anti-lock braking system, engine and transmission control, door and wiper control, etc.
  - ▶ Hand-held devices : Mobile phones, PDAs, MP3 players, digicams, etc.
  - ▶ Medical equipments: Scanners, ECG and EEG units, testing and monitoring equipments.
- 



- ▶ Banking: ATMs, currency counters, etc.
- ▶ Computer peripherals: Printers, scanners, webcams, etc.
- ▶ Networking: Routers, switches, hubs, etc.
- ▶ Factories: Control, automation, instrumentation and alarm systems.
- ▶ Aviation: Airplane controls, guidance and instrumentation systems.
- ▶ Military: Control and monitoring of military equipments.
- ▶ Robotics: Used in factories, household and hobby-related activities.
- ▶ Toys.

And many more....



# Components of Embedded Systems (Hardware)

**Input Devices** (e.g. keys, switches, sensors etc.)

**Interfacing / Driver Circuits** (decoders, signal conditioning, ADCs etc)

**Power Supply,  
Reset  
and Oscillator Circuits**

**Processor**

(8051, 68HC12, PIC16XXX, ARM  
etc.)

**Application**

**Specific Circuits** (serial  
-to-USB / Ethernet  
converter circuit)

**Timer Circuits** (Input  
capture, Output compare  
circuits, RTC etc)

**Program Memory**  
(EEPROMs, Flash etc.) and  
**Data Memory** (RAMs, SRAMs  
etc.)

**Serial  
Communication  
Ports** (SCIs, SPIs)

**Interrupt Controller**

**Parallel Ports** (2/6/8-bits etc.)

**Output Devices** (LEDs, Relays LCDs etc.)


**Interfacing / Driver Circuits** (DACs etc)

# Components of Embedded Systems (Software)


## **Main application software:**

- ▶ performs series of tasks or multiple tasks concurrently.
- ▶ constrained due to low memory, low processing power requirement and power dissipation etc.

## **Real Time Operating System (RTOS):**

- ▶ supervises the application software.
  - ▶ provides a mechanism to let the processor run a process as per scheduling.
  - ▶ performs context-switching between various processes (tasks).
  - ▶ organizes access to a resource in sequence of the series of tasks of the system.
  - ▶ schedules their working and execution by following a plan to control the latencies and to meet the deadlines.
  - ▶ sets the rules during the execution of the application software.
- 

# Classification Of Embedded Systems

- ▶ Based on Generation
  - ▶ Based on Complexity & Performance Requirements
  - ▶ Based on deterministic behavior
  - ▶ Based on Triggering
- 

# **Embedded Systems - Classification based on Generation (on the order in which the embedded processing systems evolved)**


## **First Generation:**

- ▶ Built around 8bit microprocessors like 8085 and Z80, and 4bit microcontrollers.
- ▶ Simple in hardware circuits with firmware developed in Assembly code
- ▶ Ex. Digital telephone keypads, stepper motor control units etc.


## **Second Generation:**

- ▶ Built around 16bit microprocessors and 8 or 16 bit microcontrollers, following the first generation embedded systems.
- ▶ The instruction set were much more complex and powerful than the first generation processors/controllers.
- ▶ Some of the second generation embedded systems contained embedded operating systems for their operation.
- ▶ Ex. Data Acquisition Systems, SCADA systems, etc.

## Third Generation

- ▶ 32bit processors and 16bit microcontrollers
  - ▶ A new concept of application and domain specific processors/controllers like Digital Signal Processors (DSP) and Application Specific Integrated Circuits (ASICs) came.
  - ▶ The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved.
  - ▶ Processors like Intel Pentium, Motorola 68K, etc. gained attention in high performance embedded requirements.
  - ▶ Dedicated embedded real time and general purpose operating systems entered into the embedded market.
  - ▶ Embedded systems spread its areas such as robotics, media, industrial process control, networking, etc.
- 

## Fourth Generation

- ▶ The advent of System on Chips (SoC), reconfigurable processors and multi-core processors are bringing high performance, tight integration and miniaturisation.
  - ▶ The SoC technique implements a total system on a chip by integrating different functionalities
  - ▶ with a processor core on an integrated circuit.
  - ▶ making use of high performance real time embedded operating systems for their functioning.
  - ▶ Ex. Smart phone devices, mobile internet devices (MIDs), etc.
- 

Thanking you




# **Classification Based on Complexity and Performance**


## **Small scale embedded systems**

- Built around low performance and low cost 8 or 16 bit microprocessors/microcontrollers
- Suitable for simple applications, where performance is not time critical
- It may or may not contain OS

## **Medium scale embedded systems**

- Built around medium performance, low cost 16 or 32 bit microprocessors/microcontrollers or digital signal processors.
  - Slightly complex in hardware and firmware
  - usually contain an operating system (either general purpose or real time operating system).
- 

## **Large-scale embedded systems/complex systems**

- built around high performance 32 or 64 bit RISC processors/controllers, reconfigurable system on chip(RSoC) or multi-core processors and programmable logic device (PLD).
  - It requires complex hardware and software.
  - may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor.
  - contains RTOS for scheduling, prioritization and management.
- 


## **Embedded Systems - Classification Based on deterministic behavior**

- It is applicable for Real Time systems.

### **Soft Real time Systems**

- Missing a deadline may not be critical and can be tolerated to a certain degree

### **Hard Real time systems**

- Missing a program/task execution time deadline can have catastrophic consequences (financial, human loss of life, etc.)
- 

## **Embedded Systems - Classification Based on Triggering**

### **Event Triggered**

- Activities within the system (e.g., task run-times) are dynamic and depend upon occurrence of different events .

### **Time triggered**

- Activities within the system follow a statically computed schedule (i.e., they are allocated time slots during which they can take place) and thus by nature are predictable.

Thanking you

# Design Challenges

## ▶ External Constraints

- How much hardware do we need? (too little-> deadlines missed/ too much -> too expensive)
- How do we meet deadlines? (Increased CPU clock-rate may not speed up execution time, maybe due to the memory speed)
- How do we minimize power consumption? (increased consumption -> more heat, slowing down digital systems -> missed deadlines)
- How do we design for upgradeability? (s/w compatibility for different hardware platforms when the s/w itself is not written)
- Does it really work? (how reliable are the developed systems?)

## ▶ Internal Constraints

- challenges posed by the characteristics of the components and systems themselves

# Design Metrics

- ▶ Nonrecurring engineering (NRE) Cost: One time design cost
- ▶ Unit Cost: monetary cost of manufacturing each copy of the system, excluding NRE cost.
- ▶ Size: Physical space required by the system. (s/w -> bytes, h/w -> transistors and gates)
- ▶ Performance: The execution time or throughput of the system.
- ▶ Power Consumption: amount of power consumed by the system (lifetime of a battery, cooling requirements of the IC i.e. more power -> more heat).
- ▶ Flexibility: ability to change the functionality of the system without incurring heavy NRE cost. (e.g. software)
- ▶ Time-to-prototype: time needed to build a working version of a system. Works as a yardstick to verify the product's usefulness and correctness as well as to refine and fine tune the product.
- ▶ Time-to-market: time required to develop a system to the point that it can be released and sold to the customers. (design time + manufacturing time + testing time = time-to-market)

# Design Metrics (Conti..)

- ▶ Maintainability: the ability to modify the system after its initial release.
- ▶ Correctness: measure of the confidence that the built system functions correctly (throughout the design process using test circuitry).
- ▶ **The Performance Design Metric:** measure of how long the systems takes to execute our desired tasks
  - Latency or Response Time: time between the start of the task's execution and end (e.g time to process an image = 0.20 sec).
  - Throughput: number of tasks processed per unit time (e.g a camera may process 4 images/sec)..

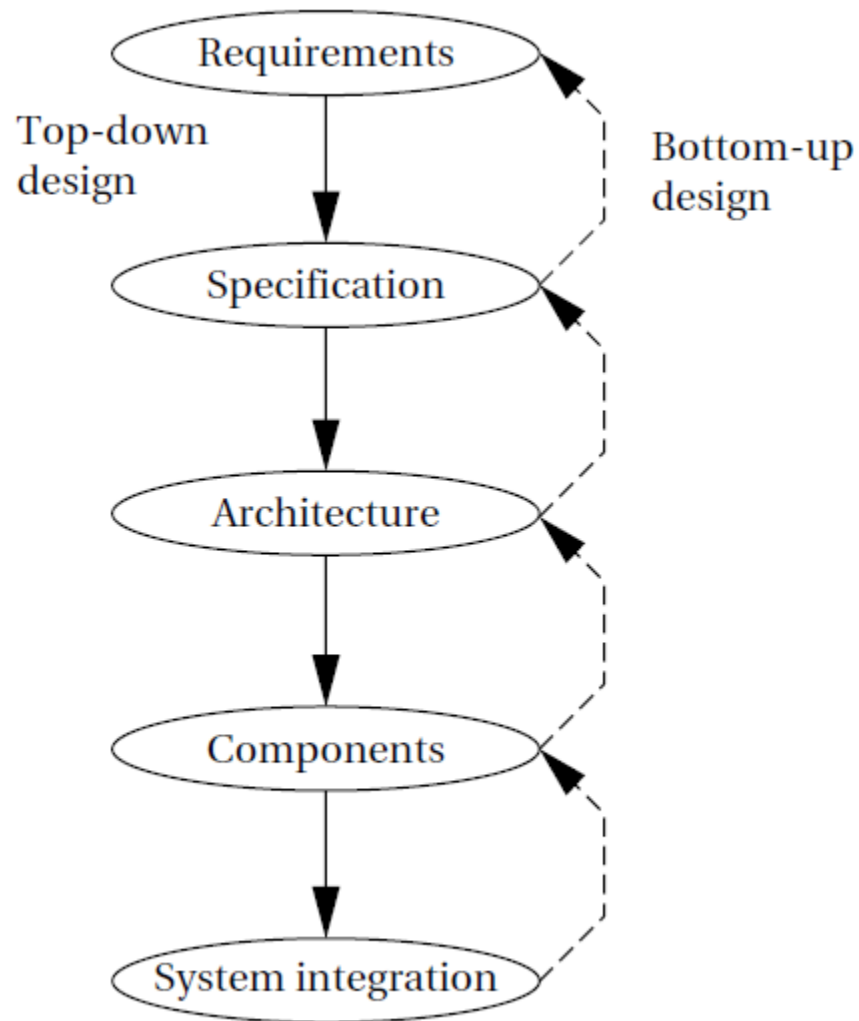


Thanking you


# Design Methodology


- ▶ Why design methodology is important?
  - Allows to **keep a scorecard on a design to ensure** that we have done everything we need to do, such as optimizing *performance or performing* functional tests.
  - Allows us to develop computer-aided design tools. Developing a single program that takes in a concept for an embedded system and emits a completed design would be a daunting (difficult) task, but by first **breaking the process into manageable steps**, we can work on automating (or at least semi-automating) the steps one at a time.
  - A design methodology **makes it much easier for members of a design team to communicate**. By defining the overall process, team members can more easily understand what they are supposed to do, what they should receive from other team members at certain times, and what they are to hand off when they complete their assigned steps.

- The major steps in the embedded system design process



Major levels of abstraction in the design process.

- ▶ In the **top–down design** flow/ view: we will begin with the most abstract description of the system and conclude with concrete details.
  - ▶ The alternative is a **bottom–up** view :in which we start with components to build a system.
  - ▶ We need bottom–up design because we do not have perfect insight into how later stages of the design process will turn out. Decisions at one stage of design are based upon estimates of what will happen later: how fast can we make a particular function run? How much memory will we need? How much system bus capacity do we need? **If our estimates are inadequate, we may have to backtrack and amend our original decisions to take the new facts into account.**
  - ▶ In general, the less experience we have with the design of similar systems, the more we will have to rely on bottom-up design information to help us refine the system.
- 

- ▶ The steps in the design process are only one axis along which we can view embedded system design.
  - ▶ We also need to consider the major goals of the design:
    - Manufacturing cost;
    - Performance (both overall speed and deadlines); and
    - Power consumption.
  - ▶ Also consider the tasks we need to perform at every step in the design process.
    - We must analyze the design at each step to determine how we can meet the specifications.
    - We must then refine the design to add detail.
    - And we must verify the design to ensure that it still meets all system goals, such as cost, speed, and so on.
- 

# Requirements

- ▶ before we design a system, we must know **what we are designing**.
- ▶ The initial stages of the design process **capture this information for use in creating the architecture and components**.
- ▶ We generally proceed in two phases:
  - First, we **gather an informal description from the customers known as requirements**
  - we **refine the requirements into a specification that contains enough information to begin designing the system architecture**.
- ▶ Consumers of embedded systems are usually not themselves embedded system designers or even product designers. Their understanding of the system is based on how they envision (imagine) users' interactions with the system. They may have unrealistic expectations as to what can be done within their budgets; and they may **also express their desires in a language very different from system architects' jargon** ( language/terminology).
- ▶ **Capturing a consistent set of requirements from the customer and then massaging those requirements into a more formal specification is a structured way to manage the process of translating from the consumer's language to the designer's.**

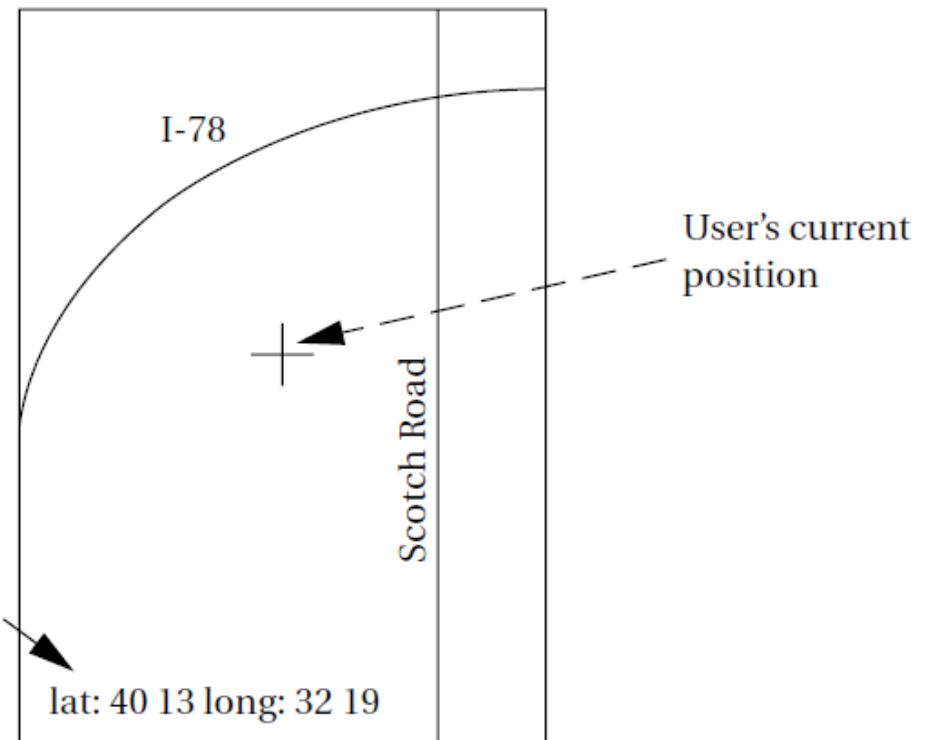
- ▶ Requirements may be functional or nonfunctional (Performance, cost, Physical size and weight, power consumption).
- ▶ Requirement form consists of entries such as name of the project, its purpose, inputs and outputs (type of data, data characteristics, types of I/O devices), its functions, performance, manufacturing cost, power, physical size and weight....

## Example : *Requirements analysis of a GPS moving map*

The moving map is a handheld device that displays for the user a map of the terrain (land) around the user's current position; the map display changes as the user and the map device change position. The moving map obtains its position from the GPS, a satellite-based navigation system. The moving map display might look something like the following figure.



User's lat/long position





- ▶ What requirements might we have for our GPS moving map? Here is an initial list:
  - **Functionality:** This system is designed for highway driving and similar uses, not Nautical/maritime or aviation uses that require more specialized databases and functions. The system should show major roads and other landmarks available in standard topographic databases.
  - **User interface:** The screen should have at least 400X600 pixel resolution. The device should be controlled by no more than three buttons. A menu system should pop up on the screen when buttons are pressed to allow the user to make selections to control the system.
  - **Performance:** The map should scroll smoothly. Upon power-up, a display should take no more than one second to appear, and the system should be able to verify its position and display the current map within 15 s.
  - **Cost:** The selling cost (street price) of the unit should be no more than \$100.
  - **Physical size and weight:** The device should fit comfortably in the palm of the hand.
  - **Power consumption:** The device should run for at least eight hours on four AA batteries.

- Based on this discussion, let's write a requirements chart for our moving map system:

---


Name	GPS moving map
Purpose	Consumer-grade moving map for driving use
Inputs	Power button, two control buttons
Outputs	Back-lit LCD display 400 × 600
Functions	Uses 5-receiver GPS system; three user-selectable resolutions; always displays current latitude and longitude
Performance	Updates screen within 0.25 seconds upon movement
Manufacturing cost	\$30
Power	100 mW
Physical size and weight	No more than 2" × 6, " 12 ounces

---

This chart adds some requirements in engineering terms that will be of use to the designers. For example, it provides actual dimensions of the device. The manufacturing cost was derived from the selling price by using a simple rule of thumb: The selling price is four to five times the ***cost of goods sold*** (the total of all the component costs).


---

## Specification

- ▶ The specification is **more precise** and serves as the contract between the customer and the architects.
  - ▶ The specification must be **carefully written** so that it accurately reflects the customer's requirements and does so in a way that can be clearly followed during design.
  - ▶ The specification should be **understandable enough** so that someone can verify that it meets system requirements and overall expectations of the customer.
  - ▶ UML (*unified modeling language*), a language for describing specifications. UML is an *object-oriented modeling language*.
- 


- ▶ Designers can run into several different types of problems caused by **unclear specifications**. If the behavior of some feature in a particular situation is unclear from the specification, the designer may implement the wrong functionality. If global characteristics of the specification are wrong or incomplete, the overall system architecture derived from the specification may be inadequate to meet the needs of implementation.

## **A specification of the GPS system:**

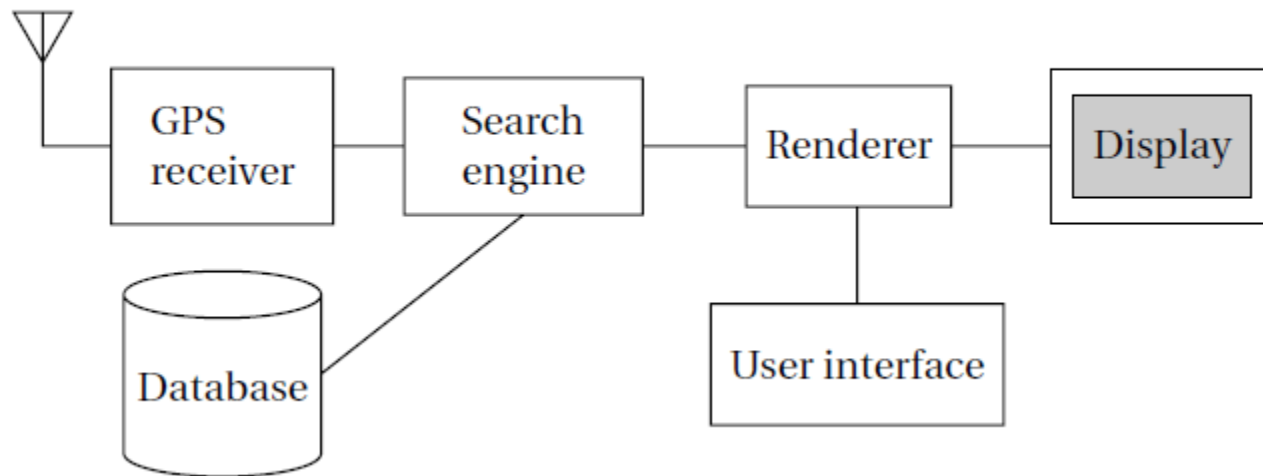
- Data received from the GPS satellite constellation.
  - Map data.
  - User interface.
  - Operations that must be performed to satisfy customer requests.
  - Background actions required to keep the system running, such as operating the GPS receiver.
- 

Thanking you

## Architecture Design

- ▶ The specification only say, **what the system does**.
  - ▶ Describing **how the system implements those functions** is the purpose of the architecture.
  - ▶ The architecture is a **plan for the overall structure of the system that will be used later to design the components** that make up the architecture.
  - ▶ Architectural descriptions must be designed to satisfy both functional and nonfunctional requirements.
- 

## Architecture design for the moving map

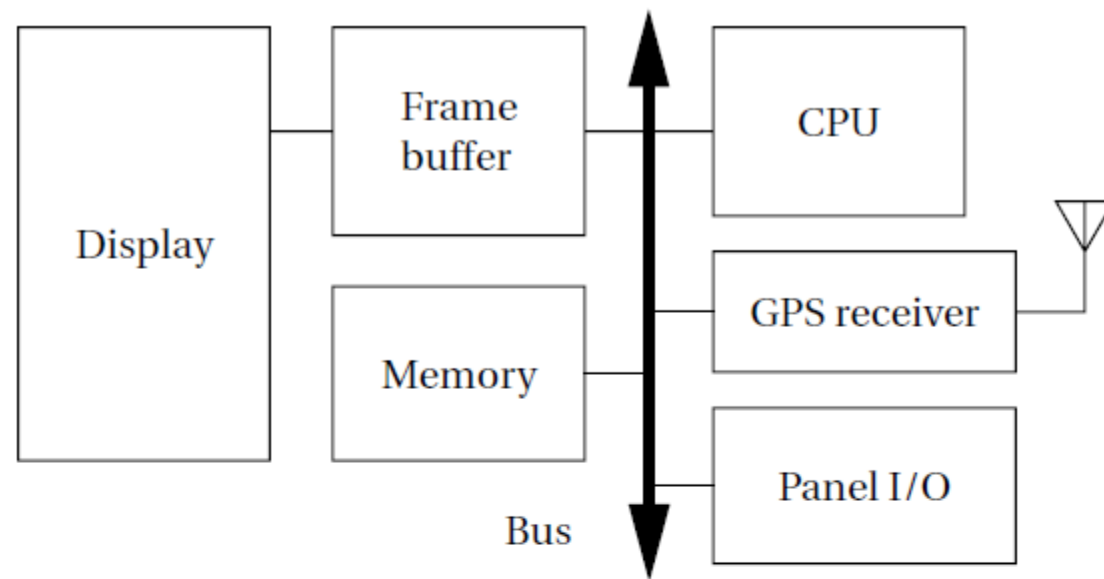


Block diagram illustrate system architecture that shows major Operations and data flows among them.

- ▶ This block diagram is still quite abstract—we have **not yet specified which operations will be performed by software running on a CPU, what will be done by special-purpose hardware**, and so on.

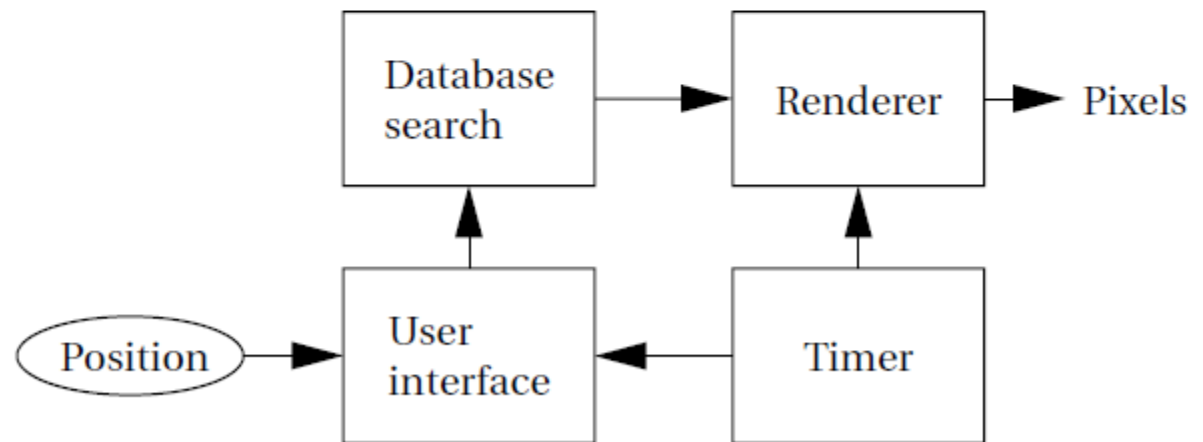


- ▶ Further **we refine the system block diagram** into two block diagrams: one for hardware and another for software.
- ▶ The **hardware block diagram** shows that
  - one central CPU surrounded by memory and I/O devices.
  - Also we have select to use two memories: a frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU.
- ▶ The **software block diagram** closely follows the system block diagram, but we have added a timer to control when we read the buttons on the user interface and give data onto the screen.
- ▶ To have a complete architectural description, we require more detail, such as where **units in the software block diagram will be executed in the hardware block diagram and when operations will be performed in time.**



Hardware and software  
architectures for the  
moving map.

### Hardware




### Software

## Designing hardware and software components

- ▶ The architectural description tells us **what components we need**.
- ▶ The component design effort **builds those components in conformance to the architecture and specification**.
- ▶ The components include both hardware - fpgas, boards, and so on and software modules.
  - Some of the **components will be ready-made**. For example, the CPU will be a standard component in almost all cases, as well memory chips and many other components.
- ▶ **In the moving map**, the GPS receiver is a good example of a specialized component that is a predesigned, standard component.


- We can also make use of **standard software modules**. Example topographic database in the moving map example. Using standard software not only saves design time, but it may give a faster implementation for specialized functions
- You will have to **design some components yourself**. Even if you are using only standard integrated circuits, you may have to design the printed circuit board that Connects them.
- You have to do a lot of custom programming as well. When creating these embedded software modules, ensure that the system runs properly in real time, does not take up more memory space, minimized power consumption (memory accesses are a major source of power consumption).

## System integration

- ▶ Only after the components are built, do we have the satisfaction of putting them together and seeing a working system.
  - ▶ Of course, this phase usually consists of a lot more than just plugging everything together and standing back. Bugs are typically found during system integration, and good planning can help us find the bugs quickly.
  - ▶ By building up the system in phases and running properly chosen tests, we can often find bugs more easily.
  - ▶ System integration is difficult because it usually uncovers problems.
  - ▶ Careful attention to inserting appropriate debugging facilities during design can help ease system integration problems
- 

Thanking You

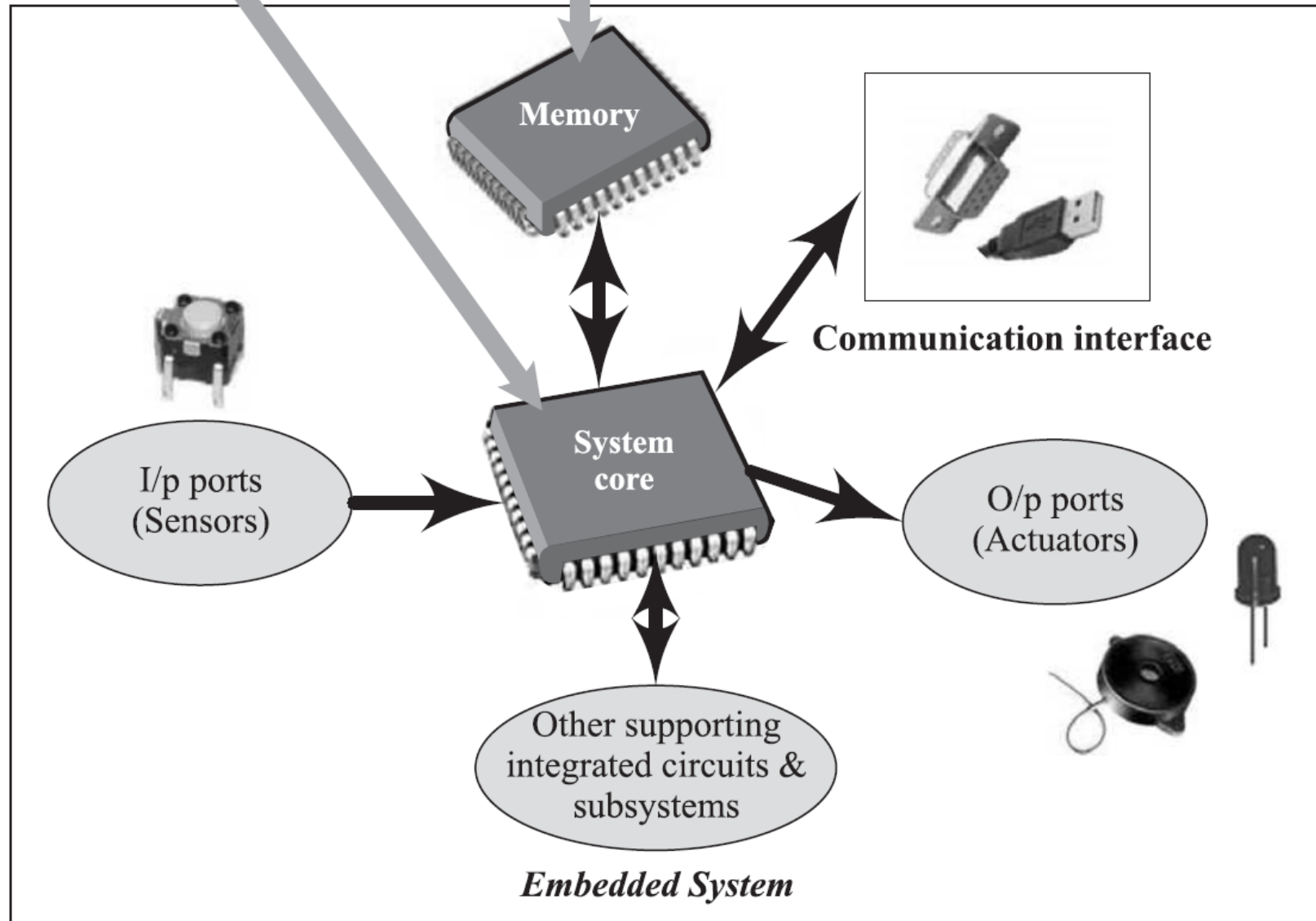
# ELEMENTS OF EMBEDDED SYSTEMS

- ▶ A typical embedded system usually consists of (as shown in figure):
    - FPGA/ASIC/DSP/SoC microprocessor/controller
    - Embedded firmware
    - Memory
    - Communication interfaces
    - I/P ports (Sensors)
    - O/P ports (Actuators)
    - Other supporting integrated circuits & subsystems
- 

**FPGA/ASIC/DSP/SoC  
Microprocessor/controller**




**Embedded  
Firmware**



***Real World***



- ▶ Embedded system is designed to regulate a physical variable (such as Microwave Oven) or to manipulate the state of some devices by sending some signals to the actuators or devices connected to the output port system (such as temperature in Air Conditioner), in response to the input signal provided by the end users or sensors which are connected to the input ports.
- ▶ Examples of user interface input devices : Key boards, push button switches, etc.
- ▶ Examples for user interface output devices: LEDs, liquid crystal displays, piezoelectric buzzers, etc.
- ▶ Type of I/O user interfaces needed for embedded system is depend on the type of the application for which the embedded system is designed. For example, if the embedded system is designed for any handheld application, such as a mobile handset application, then the system should contain user interfaces like a keyboard for performing input operations and display unit for providing users the status of various activities in progress.

- ▶ Some embedded systems do not require any manual intervention for their operation. They automatically sense the variations in the input parameters from the changes in the real world, through the sensors which are connected to the input port of the system. The sensor information is passed to the processor after signal conditioning and digitisation. Upon receiving the sensor data the processor performs some predefined operations with the help of the firmware embedded in the memory and sends some actuating signals to the actuator connected to the output port of the embedded system, which in turn acts on the controlling variable to bring the controlled variable to the desired level to make the embedded system work in the desired manner.
- 


## Core of the embedded system

- ▶ Embedded systems are domain and application specific and are built around a central core. The core of the embedded system may be either:
  - General purpose and domain specific processors
    - Microprocessors
    - Microcontrollers
    - Digital signal processors
  - Application specific integrated circuits (ASICs)
  - Programmable logic devices (PLDs)
  - Commercial off-the-shelf components (COTS)

# Embedded System Processor Chip or Core

General Purpose Processor (GPP)	Application Specific System Processor (ASSP)	Multi - processor System	Embedded System-on-chip (SOC) or Embedded System in VLSI circuit
<p>Microprocessor (<math>\mu</math>P)  Microcontroller (<math>\mu</math>C)  Embedded Processor  Digital Signal Processor (DSP)  Media Processor</p>	<p>Additional Processor or companion to the main processor.  (Processing unit for specific task)</p>	<p>GPPs + Application Specific System Processor (ASIPs)</p>	<p>GPP core(s) or ASIP core(s) integrated into either  An Application Specific Circuit (ASIC)  or  A Very Large Scale Integrated Circuit (VLSI) circuit  or  An Field Programmable Gate Array (FPGA) core integrated with processor unit (s) in a VLSI (ASIC) chip.</p>
<ul style="list-style-type: none"> <li>-General purpose instruction set</li> <li>-readily available compilers for fast s/w development</li> <li>-readily available APIs</li> <li>-H/W platform is reusable by changing s/w</li> </ul>	<p>e.g. image processing chip integrated through the buses with the main processor</p>	<p>e.g. real-time video processing and multimedia applications require multi-processing units</p>	<p>Used in large embedded systems (e.g. security applications, killer applications, PDA, satellite and guided missiles systems etc.)  In a cell phone a number of tasks such as speech signal-compression and coding, dialing, modulating and transmitting, demodulating and receiving, signal decoding and decompression, SMSing etc.</p>

## **General purpose and domain specific processors**

- ▶ Around 80% of the embedded systems are processor/controller based. Depending on the domain and application, the processor may be a microprocessor/ a microcontroller/ a digital signal processor.
  - ▶ Most of the embedded systems in the industrial control and monitoring applications make use of the commonly available microprocessors /microcontrollers.
  - ▶ while the applications which require signal processing such as speech coding, speech recognition, etc. make use of digital signal processors supplied by manufacturers like, Analog Devices, Texas Instruments, etc.
- 

# Microprocessor

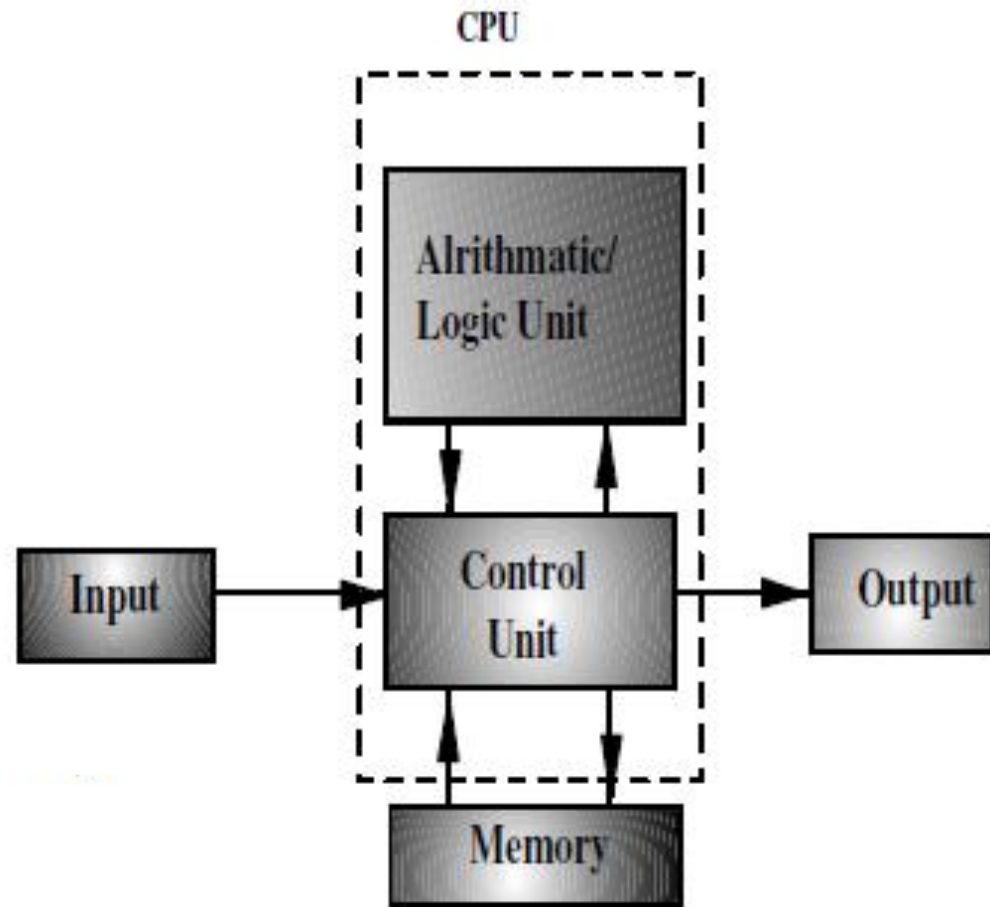
## What is a Microprocessor ?

- ▶ A microprocessor is a multipurpose, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to the instructions given and provides results as output.

Ex: 8085, 8086, Z80, 6800, Pentium processors etc

## Why do we use microprocessors ?

- ▶ efficiently implements digital systems (executes programs very efficiently)
- ▶ easier to design families of products and can be extended to meet rapidly changing market demands. (optimized design in terms of achieving greater speed i.e. processing power)





Instruction set includes instructions for performing	Other functional units besides CPU	Embedded Systems incorporated in GHz processors
<ul style="list-style-type: none"> <li>-Data Transfer Operations</li> <li>- ALU Operations</li> <li>- Stack Operations</li> <li>- Input and Output (I/O) Operations</li> <li>- Program Control Operations</li> <li>- Sequencing Operations</li> <li>- Supervising Operations</li> </ul>	<ul style="list-style-type: none"> <li>- cache memories</li> <li>- floating point processing arithmetic unit</li> <li>- pipelining unit</li> <li>- super-scaling unit</li> </ul>	<ul style="list-style-type: none"> <li>- Gbps transceiver</li> <li>- Encryption engine</li> <li>- Graphic accelerator</li> <li>- Disk controllers</li> <li>- Network Interface Cards (NICs)</li> </ul>

A **microprocessor** is used when **large embedded software** is to be located in the **external memory chips**.

**RISC core** microprocessors are used when **intensive computations** are to be performed as it provides speedy processing of instructions (in a single clock-cycle)

# Microcontroller

## What is a Microcontroller?

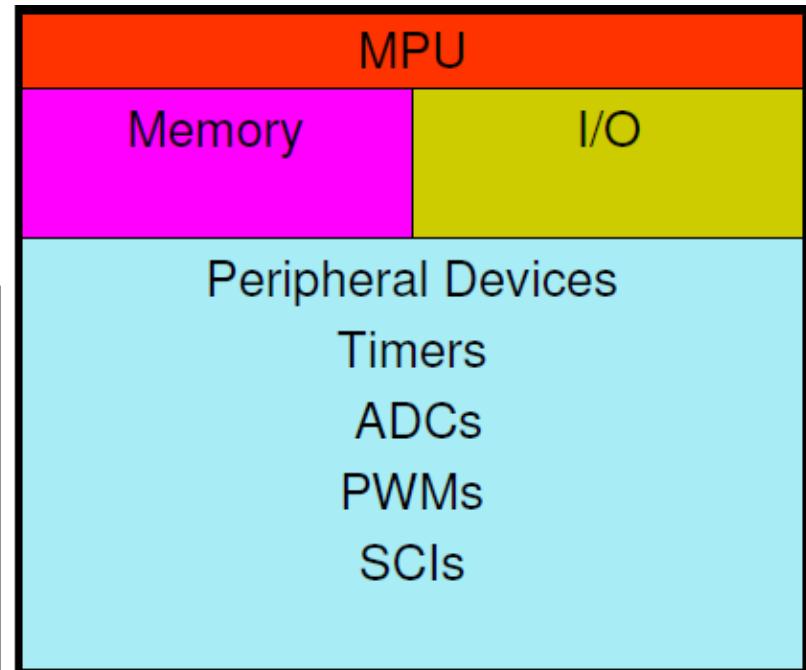
- ▶ A microcontroller is essentially an entire computer on a single chip.

Ex: Intel's 8051, 8096, Motorola M68HC11 XX /M68HC12XX, PIC 16XX series etc.

Most essential component of a control or communication unit

### Commonly used microcontrollers

Small scale embedded systems	Medium scale embedded systems	Large scale embedded systems
- 68HC05 / 08 - PIC 16F8X - 8051	- 8051 / 251 - 80x86 - 68HC11XX / 12XX - 80196	-Intel 80960CA -ARM 7 -Power PC MPC 604





## Functional circuits in a chip or core of a microcontroller

<ul style="list-style-type: none"><li>- Processor</li><li>- Data and Stack in internal RAM</li><li>- Timers and Watch dog timers</li><li>- External Memories Interface</li><li>- Non volatile PROM / ROM / EPROM</li><li>- Interrupt controller</li><li>- I/O ports control and Interface / Drivers</li><li>- Serial UART communication port</li><li>- Serial synchronous communication port</li></ul>	Application Specific Circuits in Specific Versions
	<ul style="list-style-type: none"><li>--DMA Controllers</li><li>-- Network Driver Stack and Interface</li><li>-- A/D Converter</li><li>-- LAN controller</li><li>-- PWM circuit for D/A</li><li>-- Printer Controller</li><li>-- Modem</li><li>-- DTMF circuit</li></ul>

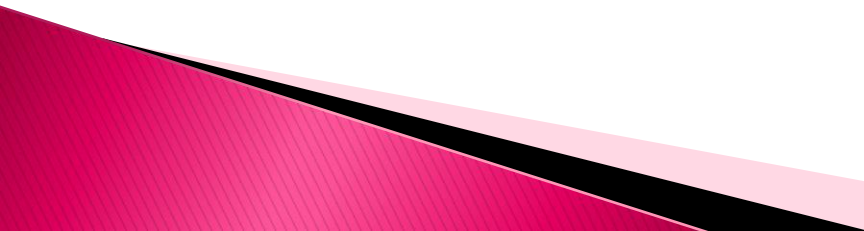
A **microcontroller** is used when a **small part of the embedded software** has to be **located in internal memory** and when the **on-chip functional units** like interrupt handler, port, timer, ADC and PWM are **needed**.

# Microprocessor verses Microcontroller

Microprocessor	Microcontroller
<p>A silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a predefined set of instructions</p> <p>It is a dependent unit. It requires the combination of other chips like timers, program and data memory chips, interrupt controllers, etc. for functioning</p>	<p>A microcontroller is a highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports</p> <p>It is a self-contained unit and it doesn't require external interrupt controller, timer, UART, etc. for its functioning</p>
Most of the time general purpose in design and operation	Mostly application-oriented or domain-specific
Doesn't contain a built in I/O port. The I/O port functionality needs to be implemented with the help of external program-mable peripheral interface chips like 8255	Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins
Limited power saving options compared to microcon-trollers	Includes lot of power saving features

Thank you


## General-purpose processor

- ▶ Suitable for a variety of applications, to maximize the number of devices sold.
  - ▶ The designer does not know what program will run on the processor, so cannot build the program into the digital circuit.
  - ▶ General datapath – the datapath must be general enough to handle a variety of computations, so typically has a large register file and one or more general-purpose arithmetic-logic units (ALUS).
  - ▶ An embedded system designer uses a general-purpose processor, by programming the processor's memory to carry out the required functionality.
- 

Using a general-purpose processor in an embedded system may result in several **design-metric benefits**:

- ▶ Design time and NRE cost are low, because the designer must only write a program, but need not do any digital design.
- ▶ Flexibility is high, because changing functionality requires only changing the program.
- ▶ Unit cost may be relatively low in small quantities, since the processor manufacturer sells large quantities to other customers and hence distributes the NRE cost over many units.
- ▶ Performance may be fast for computation-intensive applications, if using a fast processor, due to advanced architecture features and leading edge IC technology.

Some **design-metric drawbacks**:

- ▶ Unit cost may be too high for large quantities.
  - ▶ Performance may be slow for certain applications.
  - ▶ Size and power may be large due to unnecessary processor hardware.
- 

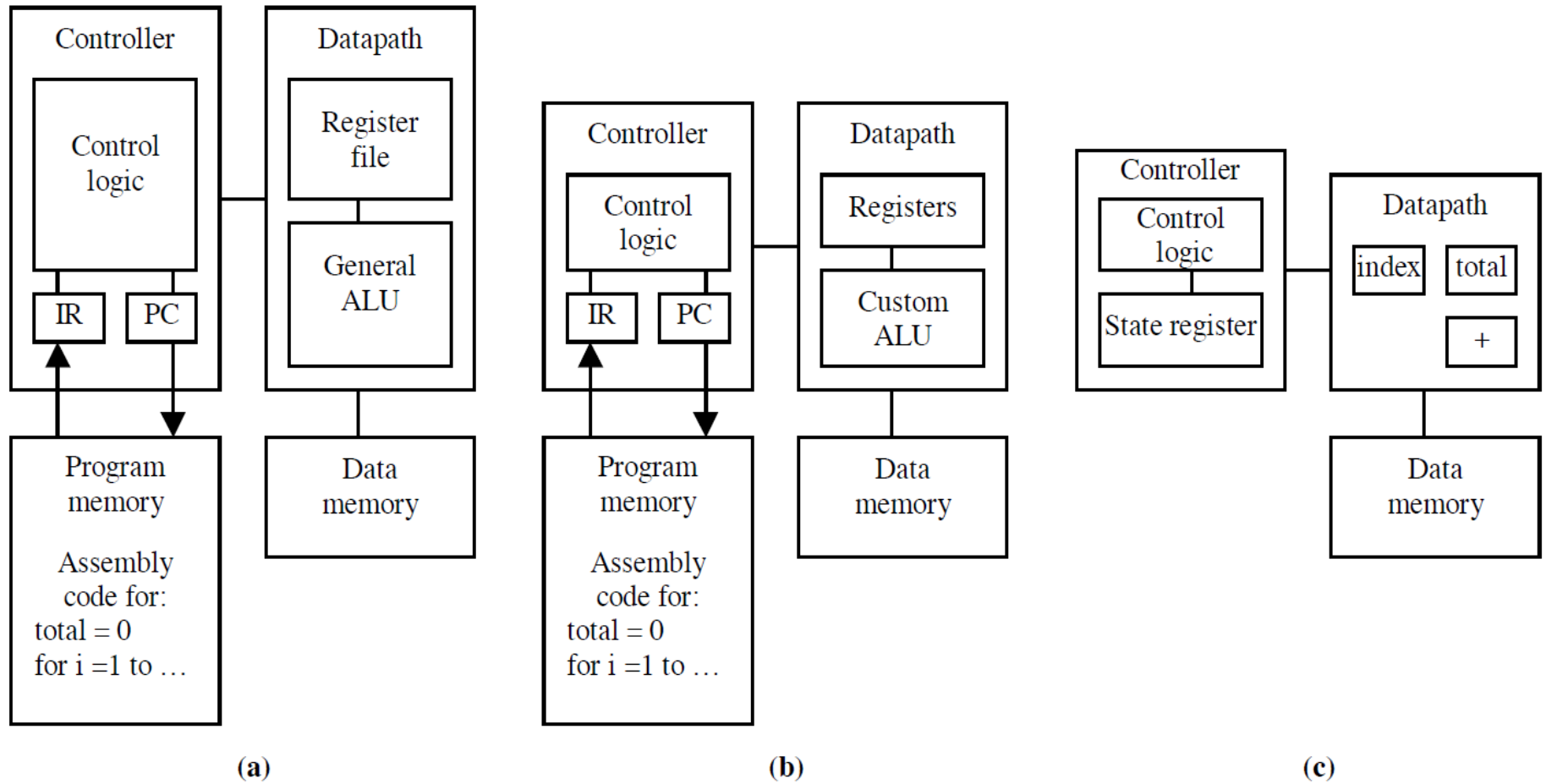
## **Single-purpose processor**

- ▶ Digital circuit designed to execute exactly one program. For example, digital camera :all of the components other than the microcontroller are single-purpose processors. The JPEG codec, for example, executes a single program that compresses and decompresses video frames.
- ▶ An embedded system designer creates a single-purpose processor by designing a custom digital circuit.

## **Design metric benefits and drawbacks:**

- ▶ Performance may be fast, size and power may be small, and unit-cost may be low for large quantities,
- ▶ While design time and NRE costs may be high, flexibility is low, unit cost may be high for small quantities, and performance may not match general-purpose processors for some applications.

Figure 1.6: Implementing desired functionality on different processor types: (a) general-purpose, (b) application-specific, (c) single-purpose.




## **Application-specific instruction-set processor (or ASIP)**

- ▶ Can serve as a compromise between the general purpose and single purpose processor.
- ▶ An ASIP is designed for a particular class of applications with common characteristics, such as digital-signal processing, telecommunications, embedded control, etc.
- ▶ The designer of such a processor can optimize the datapath for the application class, perhaps adding special functional units for common operations, and eliminating other infrequently used units.
- ▶ Using an ASIP in an embedded system can provide the benefit of flexibility while still achieving good performance, power and size.
- ▶ However, such processors can require large NRE cost to build the processor itself, and to build a compiler, if these items don't already exist.



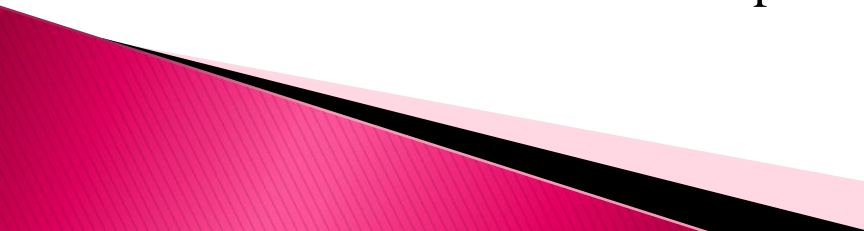
## Digital-signal processors (DSPS)

- ▶ A DSP is a processor designed to perform common operations on digital signals, which are the digital encodings of analog signals like video and audio. These operations carry out common signal processing tasks like signal filtering, transformation, or combination.
- ▶ Such operations are usually math-intensive, including operations like multiply and add or shift and add.
- ▶ To support such operations, a DSP may have special purpose datapath components such a multiply-accumulate unit, which can perform a computation like  $t = t + m[i] * k$  using only one instruction.
- ▶ Because DSP programs often manipulate large arrays of data, a DSP may also include special hardware to fetch sequential data memory locations in parallel with other operations, to further speed execution.

- ▶ Digital signal processors are 2 to 3 times faster than the general purpose processors because of the architectural difference between them. DSPs implement algorithms in hardware which speeds up the execution whereas general purpose processors implement the algorithm in firmware and the speed of execution depends primarily on the clock for the processors.
  - ▶ Audio video signal processing, telecommunication, and multimedia applications are typical examples where DSP is employed.
  - ▶ Digital signal processing employs a large amount of real-time calculations. Sum of products (SOP) calculation, convolution, fast fourier transform (FFT), discrete fourier transform (DFT), etc, are some of the operations performed by digital signal processors.
- 

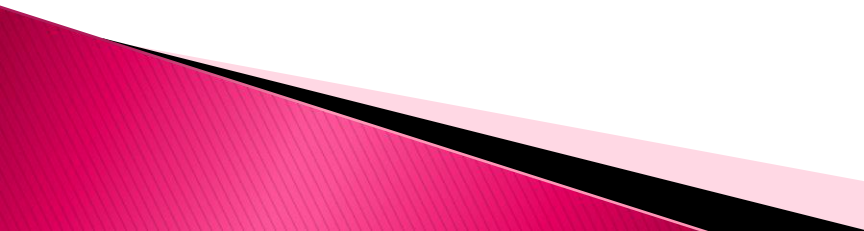
## RISC vs. CISC Architecture

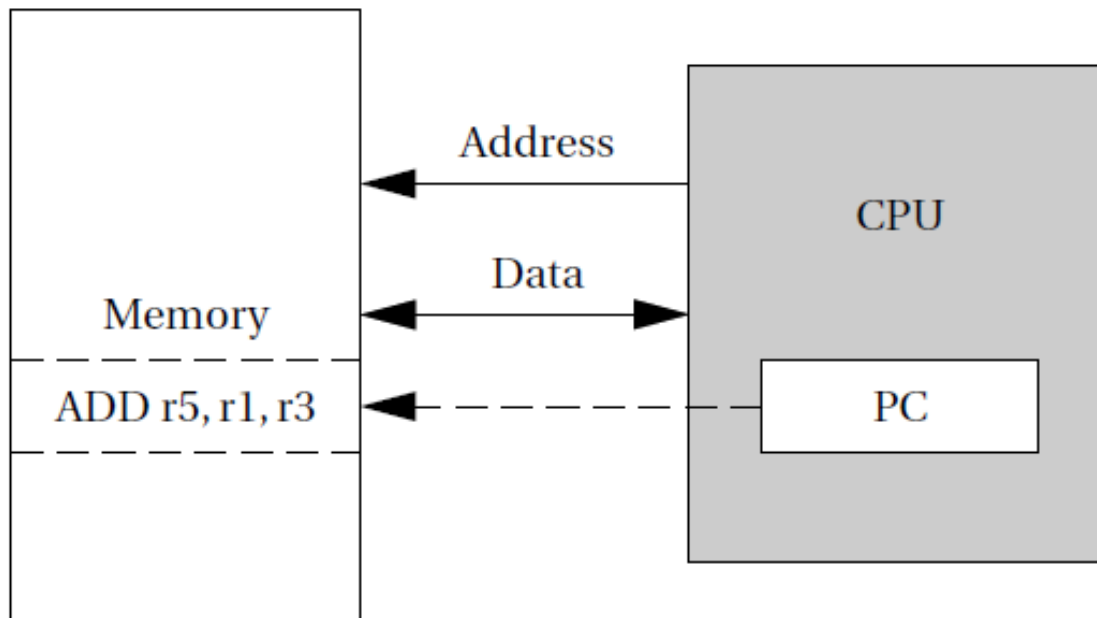
- ▶ RISC stands for Reduced Instruction Set Computer and CISC means Complex Instruction Set Computer.
- ▶ In the early days of microprocessor development, the trend was to have complex instructions implemented fully using hardware. For example, the multiply instruction is a complex instruction which needs a dedicated hardware multiplier. Because hardware is fast, execution is fast, but with lots of such complex instructions, the hardware budget is naturally high.
- ▶ RISC on the other hand, views this matter in a different way. On an average, the number of complex instructions a computer uses is relatively less. So, why not realize a complex instruction using a set of simple instructions? This is possible, and the advantage is that the hardware budget is much less. The instruction set is also small. However, software is to be written to realize complex instructions with simple instructions. This amounts to trading software for hardware.

- ▶ As the name implies, all RISC processors/controllers possess simple and lesser number of instructions while CISC processors/controllers have the complex and large number of instructions.
  - ▶ From a programmers point of view RISC processors are comfortable since s/he needs to learn only a few instructions, whereas for a CISC processor s/he needs to learn more number of instructions and should understand the context of usage of each instruction .
  - ▶ **Atmel AVR microcontroller** is an example for a RISC processor and its instruction set contains only 32 instructions. The original version of 8051 microcontroller (e.g. **AT89C51**) is a CISC controller and its instruction set contains 255 instructions. (Remember it is not the number of instructions that determines whether a processor/controller is CISC or RISC.)
- 

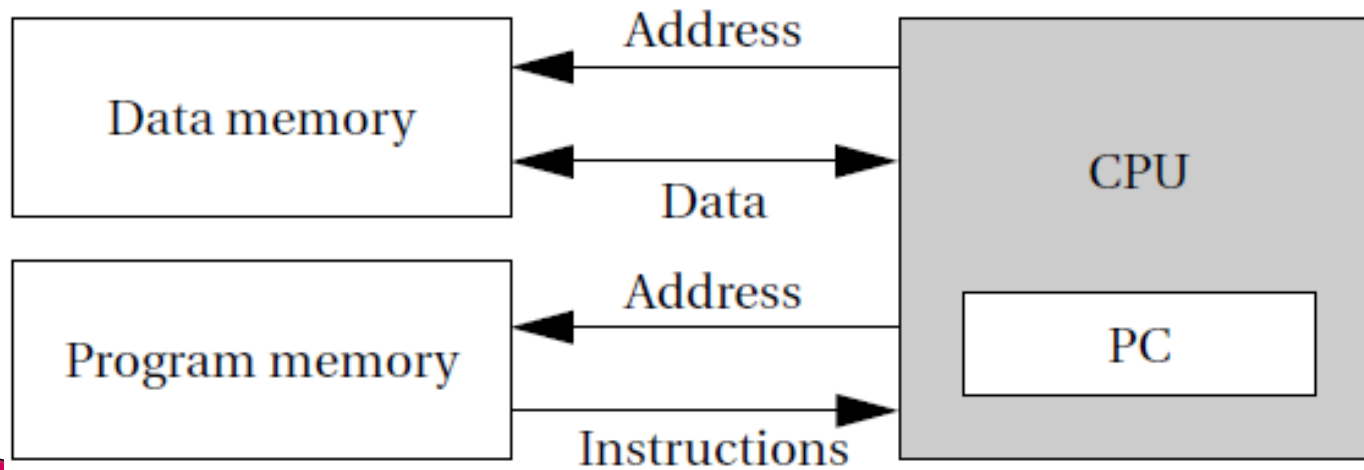
RISC	CISC
Lesser number of instructions	Greater number of Instructions
Instruction pipelining and increased execution speed	Generally no instruction pipelining feature
Orthogonal instruction set (Allows each instruction to operate on any register and use any addressing mode)	Non-orthogonal instruction set (All instructions are not allowed to operate on any register and use any addressing mode. It is instruction-specific)
Operations are performed on registers only, the only memory operations are load and store	Operations are performed on registers or memory depending on the instruction
A large number of registers are available	Limited number of general purpose registers
Programmer needs to write more code to execute a task since the instructions are simpler ones	Instructions are like macros in C language. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instructions in RISC
Single, fixed length instructions	Variable length instructions
Less silicon usage and pin count	More silicon usage since more additional decoder logic is required to implement the complex instruction decoding.
With Harvard Architecture	Can be Harvard or Von-Neumann Architecture

## Harvard / Von-Neumann Processor/Controller Architecture


- ▶ The computing system consists of a *central processing unit (CPU) and a memory*.
  - ▶ **Von-Neumann** architecture shares a single common bus for fetching both instructions and data. Program instructions and data are stored in a common main memory.
  - ▶ Von-Neumann architecture based processors/controllers first fetch an instruction and then fetch the data to support the instruction. The two separate fetches slows down the controller's operation.
  - ▶ Von-Neumann architecture is also referred as **Princeton architecture**, since it was developed by the Princeton University.
- 



Von Neumann Architecture




Harvard Architecture

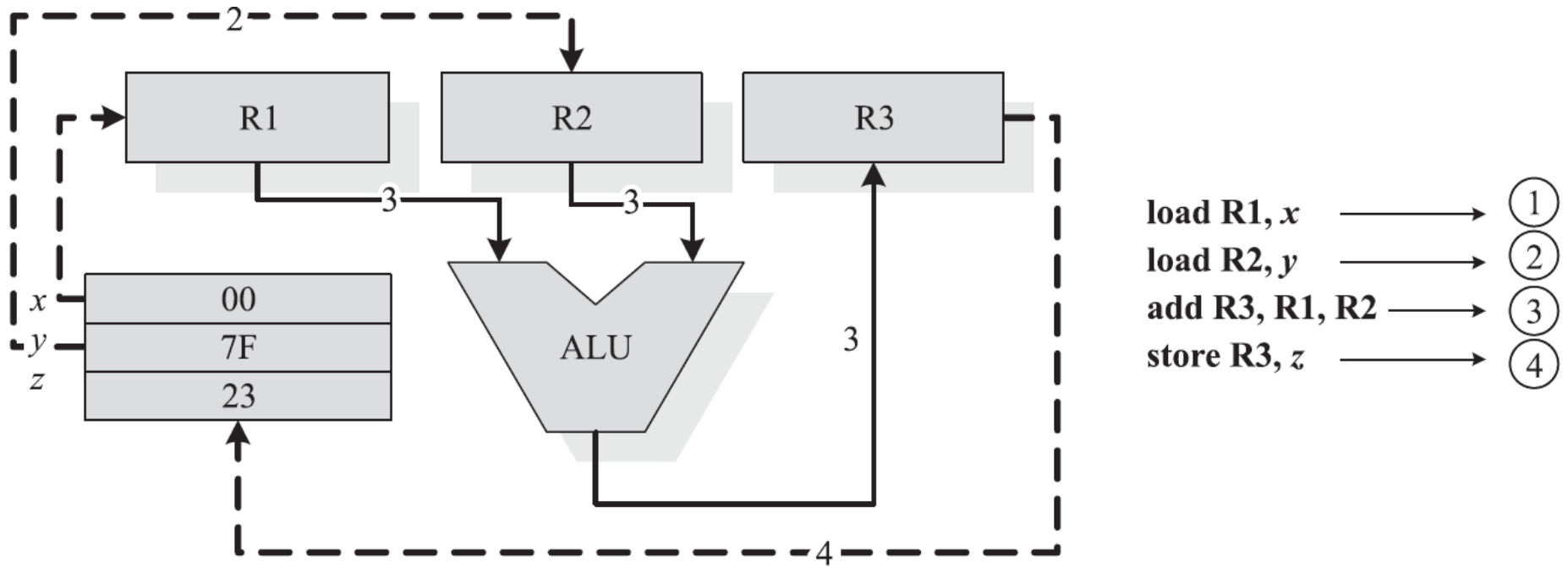
- ▶ **Harvard architecture** will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both buses.
  - ▶ With Harvard architecture, the data memory can be read and write while the program memory is being accessed. These separated data memory and code memory buses allow one instruction to execute while the next instruction is fetched (“prefetching”). The prefetch theoretically allows much faster execution than Von-Neumann architecture.
  - ▶ Some additional hardware logic is required for the generation of control signals for this type of operation it adds silicon complexity to the system.
- 



<b>Harvard Architecture</b>	<b>Von-Neumann Architecture</b>
Separate buses for instruction and data fetching	Single shared bus for instruction and data fetching
Easier to pipeline, so high performance can be achieved	Low performance compared to Harvard architecture
Comparatively high cost	Cheaper
Since data memory and program memory are stored physically in different locations, no chances for accidental corruption of program memory	Since data memory and program memory are stored physically in the same chip, chances for accidental corruption of program memory

## Load Store Operation and Instruction Pipelining

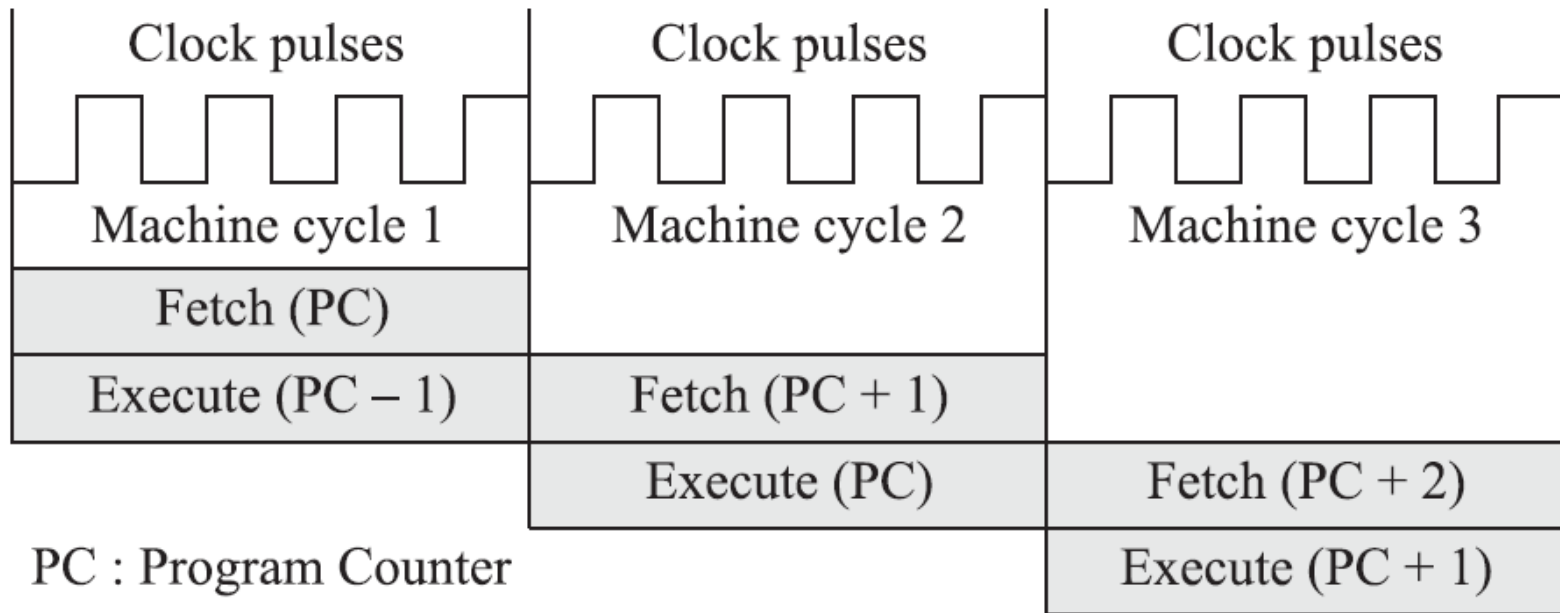
- ▶ The RISC processor instruction set is orthogonal i.e. it operates on registers. The memory access related operations are performed by the special instructions load and store.
  - ▶ If the operand is specified as memory location, the content of it is loaded to a register using the load instruction. The instruction store stores data from a specified register to a specified memory location.
  - ▶ The concept of Load Store Architecture is illustrated with the following example: suppose x, y and z are memory locations and we want to add the contents of x and y and store the result in location z. Under the load store architecture the same is achieved with 4 instructions.
- 



The concept of load store architecture


- ▶ The first instruction load R1, x loads the register R1 with the content of memory location x, the second instruction load R2, y loads the register R2 with the content of memory location y. The instruction add R3, R1, R2 adds the content of registers R1 and R2 and stores the result in register R3. The next instruction store R3, z stores the content of register R3 in memory location z.
- ▶ The conventional instruction execution by the processor follows the **fetch-decode-execute sequence**. Where
  - the 'fetch' part fetches the instruction from program memory or code memory
  - the decode part decodes the instruction to generate the necessary control signals
  - The execute stage reads the operands, perform ALU operations and stores the result.


- ▶ In conventional program execution, the fetch and decode operations are performed in sequence. Let's consider decode and execution together. During the decode operation the memory address bus is available and if it is possible to effectively utilize it for an instruction fetch, the processing speed can be increased.
- ▶ **Instruction pipelining refers to the overlapped execution of instructions.** Under normal program execution flow it is meaningful to fetch the next instruction to execute, while the decoding and execution of the current instruction is in progress.
- ▶ If the current instruction in progress is a **program control flow transfer instruction like jump or call instruction**, there is no meaning in fetching the instruction following the current instruction. In such cases the instruction fetched is flushed and a new instruction fetch is performed to fetch the instruction.
- ▶ Whenever the current instruction is executing the program counter will be loaded with the address of the next instruction. In case of jump or branch instruction, the new location is known only after completion of the jump or branch instruction.
- ▶ Depending on **the stages involved in an instruction** (fetch, read register and decode, execute instruction, access an operand in data memory, write back the result to register, etc.) there can be **multiple levels of instruction pipelining.**



The single-stage pipelining concept

# Programmable Logic Devices

- ▶ Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform.
  - ▶ Logic devices can be classified into two broad categories—fixed and programmable.
  - ▶ The circuits in a **fixed logic device** are permanent, they perform one function or set of functions—once manufactured, they cannot be changed.
  - ▶ Programmable Logic Devices ( **PLDs**) offer customers a wide range of logic capacity, features, speed, and voltage characteristics—and these devices can be re-configured to perform any number of functions at any time.
- 

- ▶ With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit.
  - ▶ There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device.
  - ▶ During the design phase of PLDS, customers can change the circuitry as often as they want until the design operates to their satisfaction. That's because PLDs are based on re-writable memory technology—to change the design, the device is simply reprogrammed.
- 



## CPLDs and FPGAs

The **two major types of PLDs** are Field Programmable Gate Arrays ( FPGAs) and Complex Programmable Logic Devices ( CPLDs).

- ▶ FPGAs offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA now shipping, part of the Xilinx Virtex™§ line of devices, provides eight million “system gates” (the relative density of logic). These advanced devices also offer features such as built-in hardwired processors (such as the IBM power PC), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies.
- ▶ FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing.

- ▶ CPLDs, offer much smaller amounts of logic—up to about 10,000 gates. But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications.
- ▶ CPLDs such as the Xilinx CoolRunner™† series also require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

## Commercial Off-the-Shelf Components (COTS)

- ▶ COTS products are designed in such a way to provide easy integration and interoperability with existing system components. The COTS component itself may be developed around a general purpose or domain specific processor or an Application Specific Integrated circuit or a programmable logic device.
- ▶ Typical examples of COTS hardware unit are remote controlled toy car control units including the RF circuitry part, high performance, high frequency microwave electronics (2–200 GHz), high bandwidth analog-to-digital converters, devices and components for operation at very high temperatures, electro-optic IR imaging arrays, UV/IR detectors, etc.
- ▶ The major advantage of using COTS is that they are readily available in the market, are cheap and a developer can cut down his/her development time to a great extent. This in turn reduces the time to market your embedded systems.

- ▶ **Ex.** The TCP/IP plug-in module available from various manufactures like 'WIZnet', 'HanRun', 'Viewtool', etc. gives the TCP/IP connectivity to the system you are developing.
- ▶ What you need to do is identify the COTS for your system and give the plug-in option on your board according to the hardware plug-specifications

of the COTS.

- ▶ The major **problem** faced by the end- user is that there no operational and manufacturing standards follow by COTS vendors. A COTS component manufactured by a vendor need not have hardware plug-in and firmware

interface compatibility with one manufactured by a second vendor for the same application. This restricts the end-user to stick to a particular vendor for a particular COTS. This greatly affects the product design.



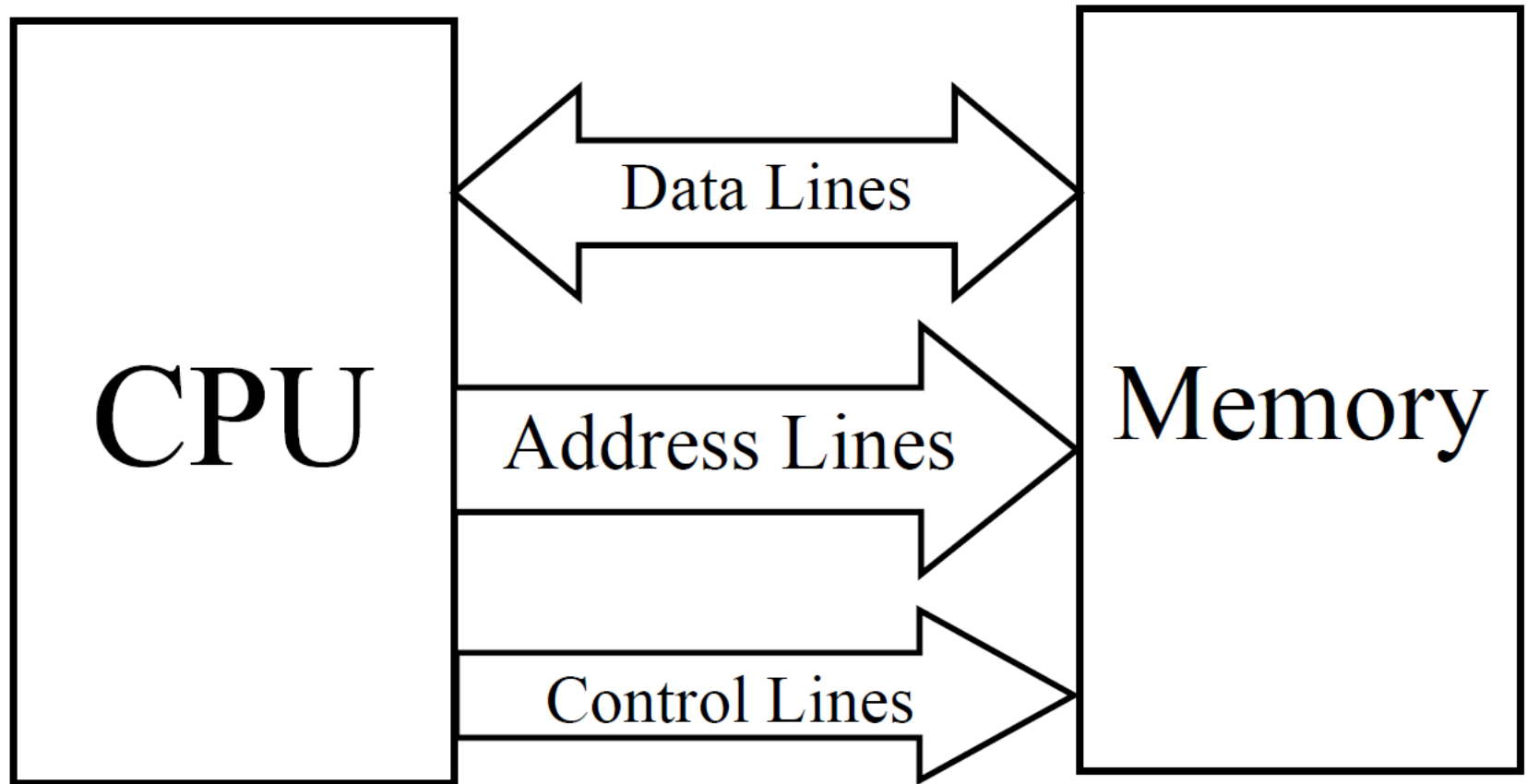
**An example of a COTS product for TCP/IP plug-in from WIZnet**


Major **drawback** of using COTS components in embedded design is that the manufacturer of the COTS component may withdraw the product or discontinue the production of the COTS at any time if a rapid change in technology occurs, and this will adversely affect a commercial manufacturer of the embedded system which makes use of the specific COTS product.



# Memory

- ▶ *Memory serves processor short and long-term information storage requirements while registers serve the processor's short-term storage requirements.*
- ▶ The memory may be Read-Only-Memory or Random Access Memory (RAM). It may exist on the same chip with the processor itself or may exist outside the chip. The on-chip memory is faster than the off-chip memory.
- ▶ The memory is connected to the CPU through the following lines
  - Address
  - Data
  - Control



- ▶ In a memory **read operation** the CPU loads the address onto the address bus. Most cases these lines are fed to a decoder which selects the proper memory location. The CPU then sends a read control signal. The data is stored in that location is transferred to the processor via the data lines.
  - ▶ In the memory **write operation** after the address is loaded the CPU sends the write control signal followed by the data to the requested memory location.
  - ▶ The memory can be **classified in various ways** i.e. based on the location, power consumption, way of data storage etc
  - ▶ The memory at the basic level can be classified as
    - Processor Memory (Register Array)
    - Internal on-chip Memory
    - Primary Memory
    - Cache Memory
    - Secondary Memory
- 



The memory can also be divided into:

- Volatile Memory
- Non-volatile Memory

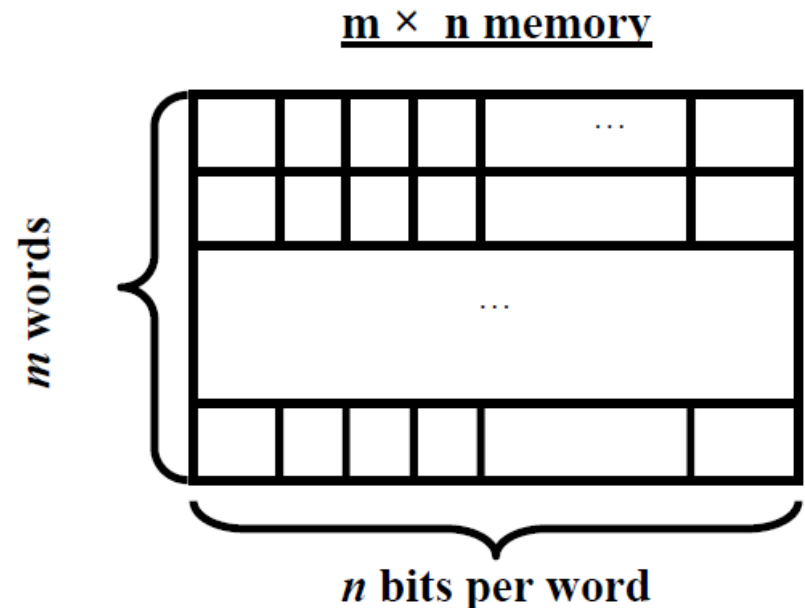
## Data Storage

- An  $m$  word memory can store  $m \times n$ :  $m$  words of  $n$  bits each. One word is located at one address therefore to address  $m$  words we need.

$k = \text{Log}_2(m)$  address input signals

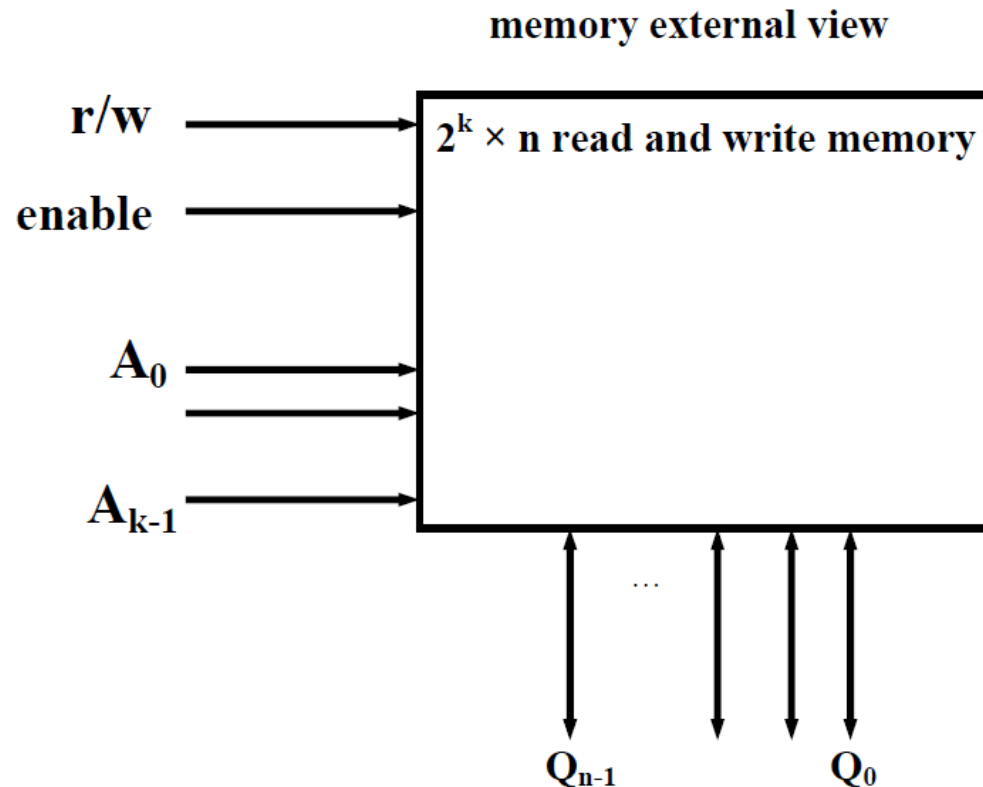
or  $k$  number address lines can address  $m = 2^k$  words

**Example 4,096 x 8 memory:**




## Memory access

- ▶ The memory location can be accessed by placing the address on the address lines. The control lines read/write selects read or write. Some memory devices are multi-port i.e. multiple accesses to different locations simultaneously .



## Memory Specifications

- The storage capacity: The number of bits/bytes or words it can store
  - The memory access time (read access and write access): How long the memory takes to load the data on to its data lines after it has been addressed or how fast it can store the data upon supplied through its data lines. This reciprocal of the memory access time is known as *Memory Bandwidth*
  - The Power Consumption and Voltage Levels: The power consumption is a major factor in embedded systems. The lesser is the power consumption the more is packing density.
  - Size: Size is directly related to the power consumption and data storage capacity.
- 

There are two important specifications for the Memory as far as Real Time Embedded Systems are concerned.

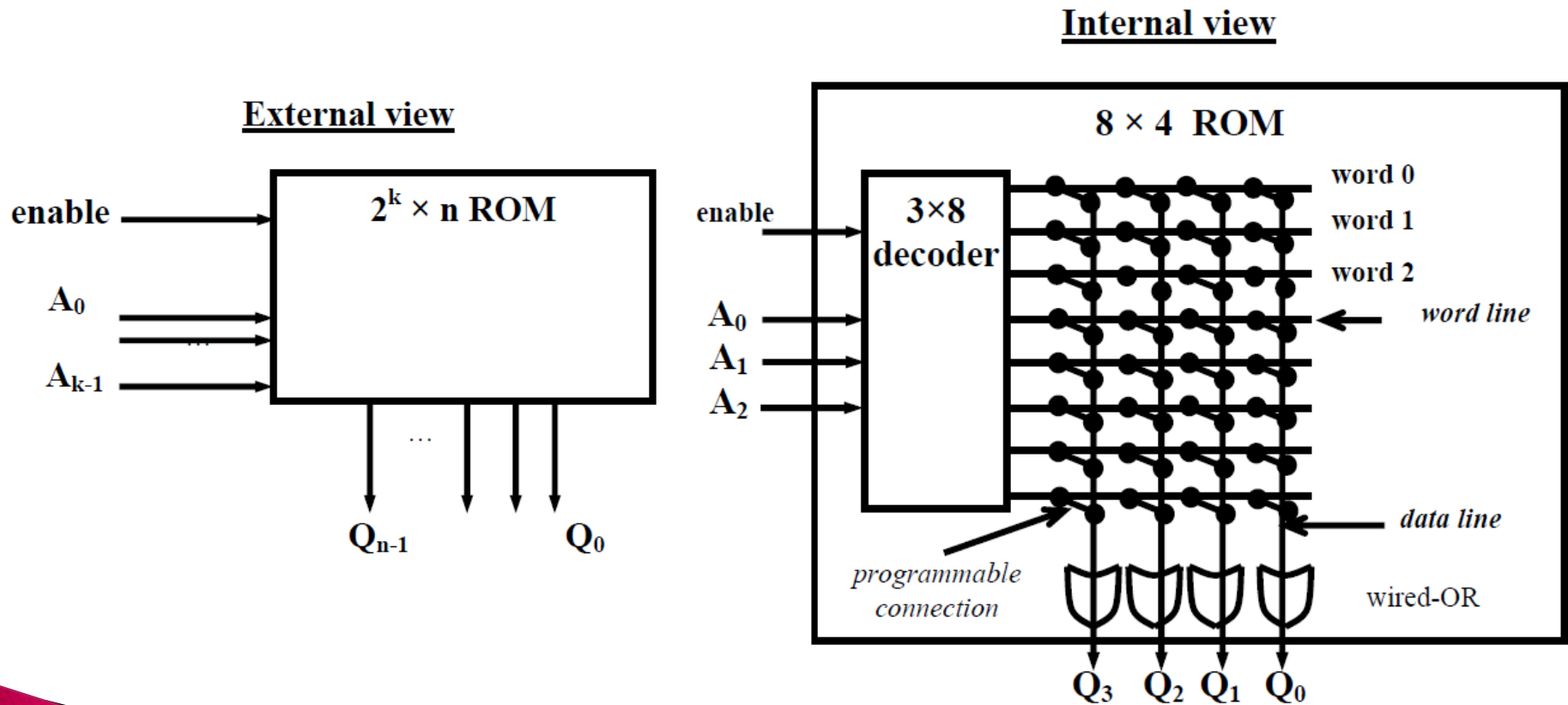
- Write Ability :It is the manner and speed that a particular memory can be written
- Ranges of write ability :
  - High end
    - processor writes to memory simply and quickly e.g., RAM
  - Middle range
    - processor writes to memory, but slower e.g., FLASH, EEPROM (Electrically Erasable and Programmable Read Only Memory)
  - Lower range
    - special equipment, “programmer”, must be used to write to memory e.g., EPROM, OTP ROM (One Time Programmable Read Only Memory)
  - Low end
    - bits stored only during fabrication e.g., Mask-programmed ROM
- ▶ In-system programmable memory
  - Can be written to by a processor in the embedded system using the memory
  - Memories in high end and middle range of write ability

- Storage permanence :It is the ability to hold the stored bits.

Range of storage permanence :

- High end
  - essentially never loses bits  
e.g., mask-programmed ROM
- Middle range
  - holds bits days, months, or years after memory's power source turned off  
e.g., NVRAM
- Lower range
  - holds bits as long as power supplied to memory  
e.g., SRAM
- Low end
  - begins to lose bits almost immediately after written  
e.g., DRAM

- ▶ Common Memory Types
- ▶ Read Only Memory (ROM)




- ▶ Mask-programmed ROM :
  - written only once (in the factory)
  - stores data for ever
  - has the highest storage permanence
  - typically used for final design of high-volume systems.
- ▶ OTP ROM (One-time programmable ROM ):
  - Each programmable connection is a fuse and programmed by blowing fuses where connections should not exist.
  - Typically written only once and have high storage permanence .
  - Commonly used in final products

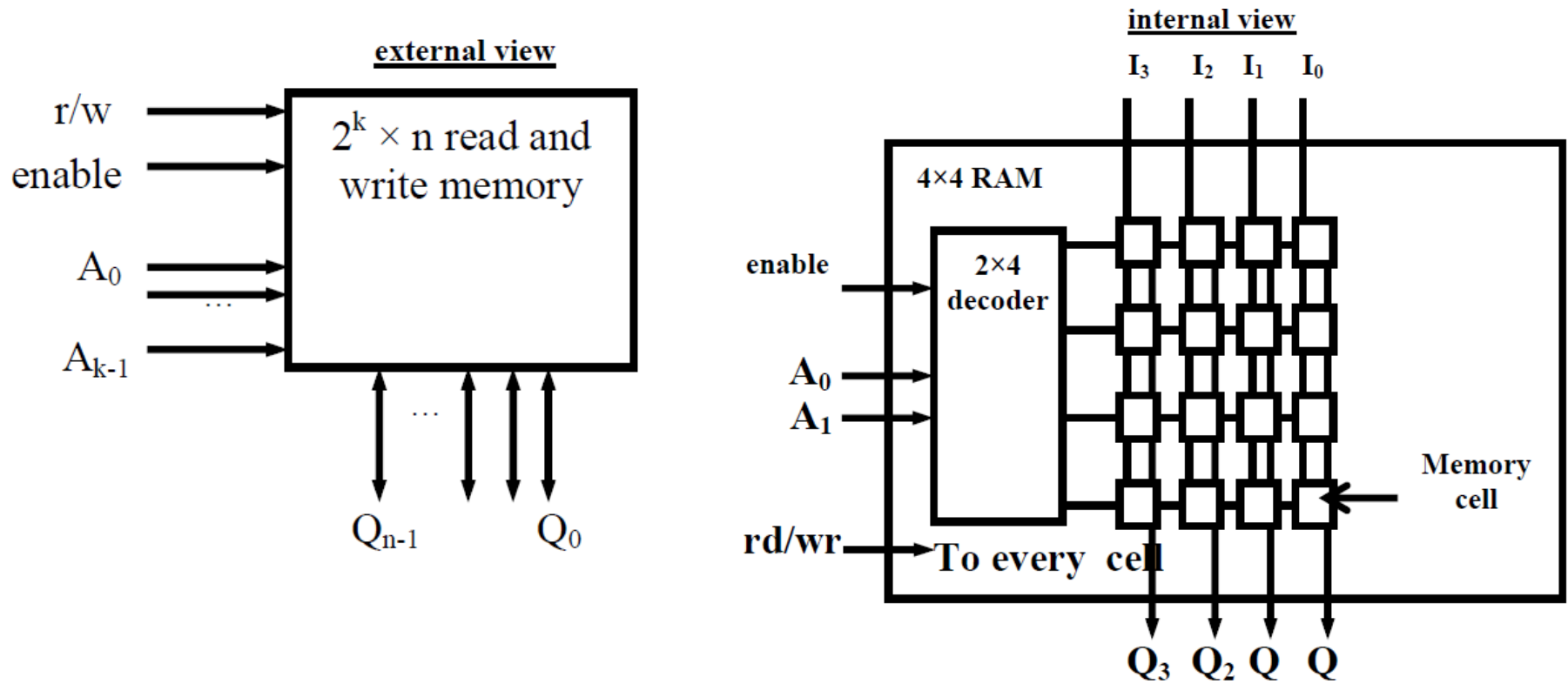
## EPROM (Erasable programmable ROM):

- The programmable component is a MOS transistor and has a “floating” gate surrounded by an insulator. The Negative charges form a channel between source and drain storing a logic 1. The Large positive voltage at gate causes negative charges to move out of channel and get trapped in floating gate storing a logic 0.
- The (Erase) Shining UV rays on surface of floating-gate causes negative charges to return to channel from floating gate restore the logic 1.
- Better write ability (can be erased and reprogrammed thousands of times )
- Reduced storage permanence ( program lasts about 10 years but is susceptible to radiation and electric noise)
- Typically used during design development



- ▶ EEPROM (Electrically Erasable and Programmable ROM) :
    - erased typically by using higher than normal voltage. It can program and erase individual words
    - Better write ability
    - Similar storage permanence to EPROM (about 10 years)
    - Far more convenient than EPROMs, but more expensive
  - ▶ Flash Memory
    - extension of EEPROM. It has the same floating gate principle and same write ability and storage permanence.
    - It can be erased at a faster rate i.e. large blocks of memory erased at once, rather than one word at a time. The blocks are typically several thousand bytes large.
- 

## ▶ RAM:



## ► Types of RAM

SRAM cell	DRAM cell
Made up of 6 CMOS transistors (MOSFET)	Made up of a MOSFET and a capacitor
Doesn't require refreshing	Requires refreshing
Low capacity (Less dense)	High capacity (Highly dense)
More expensive	Less expensive
Fast in operation. Typical access time is 10ns	Slow in operation due to refresh requirements. Typical access time is 60ns. Write operation is faster than read operation.

### ► PSRAM: Pseudo-static RAM

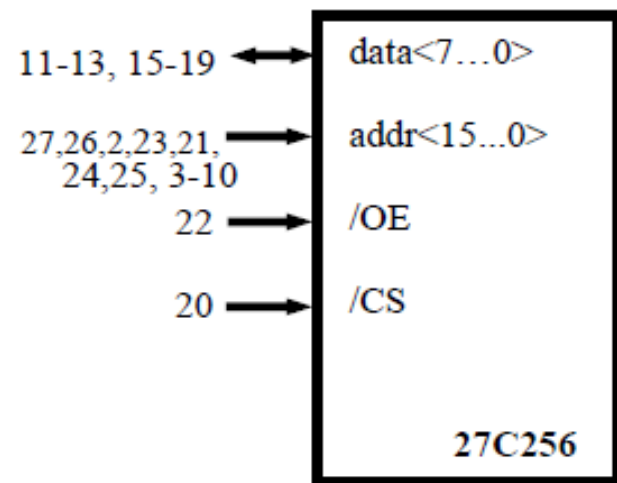
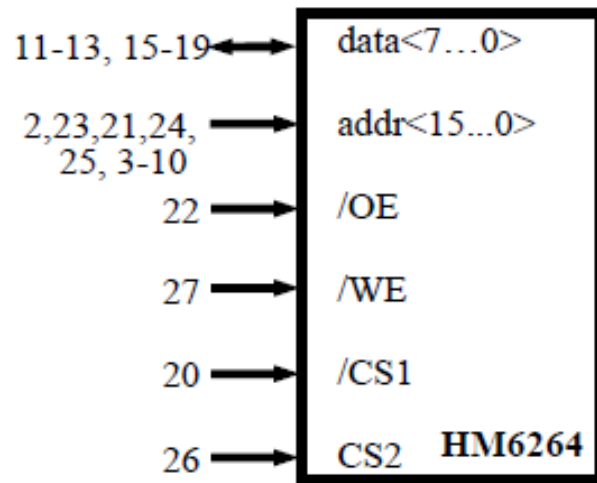
- DRAM with built-in memory refresh controller
- Popular low-cost high-density alternative to SRAM

### ► NVRAM: Nonvolatile RAM

- Holds data after external power removed
- Battery-backed RAM
- SRAM with own permanently connected battery
  - writes as fast as reads
  - no limit on number of writes unlike nonvolatile ROM-based memory

## Example: HM6264 & 27C256 RAM/ROM devices

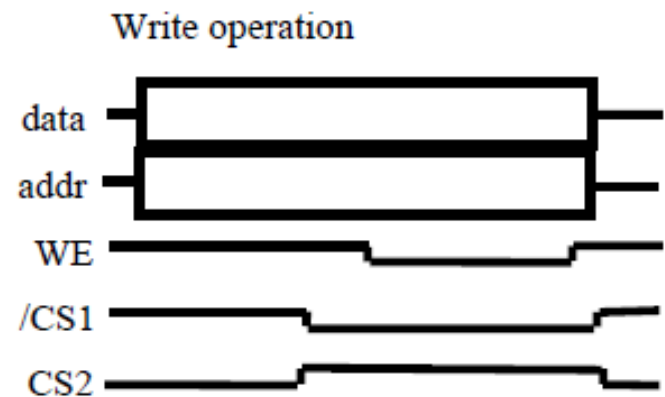
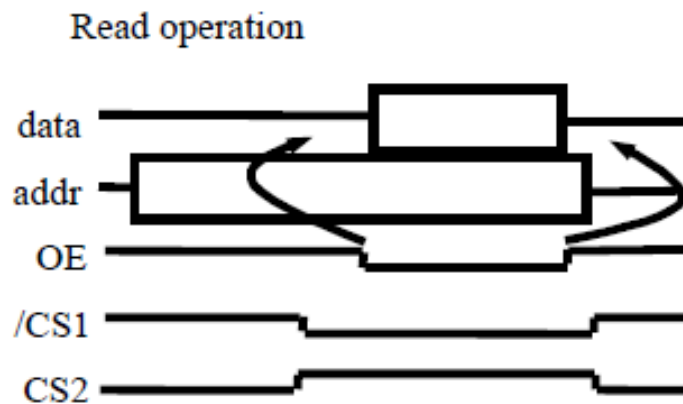
- Low-cost low-capacity memory devices
- Commonly used in 8-bit microcontroller-based embedded systems
- First two numeric digits indicate device type
  - RAM: 62
  - ROM: 27
- Subsequent digits indicate capacity in kilobits



block diagrams

Device	Access Time (ns)	Standby Pwr. (mW)	Active Pwr. (mW)	Vcc Voltage (V)
HM6264	85-100	.01	15	5
27C256	90	.5	100	5

device characteristics

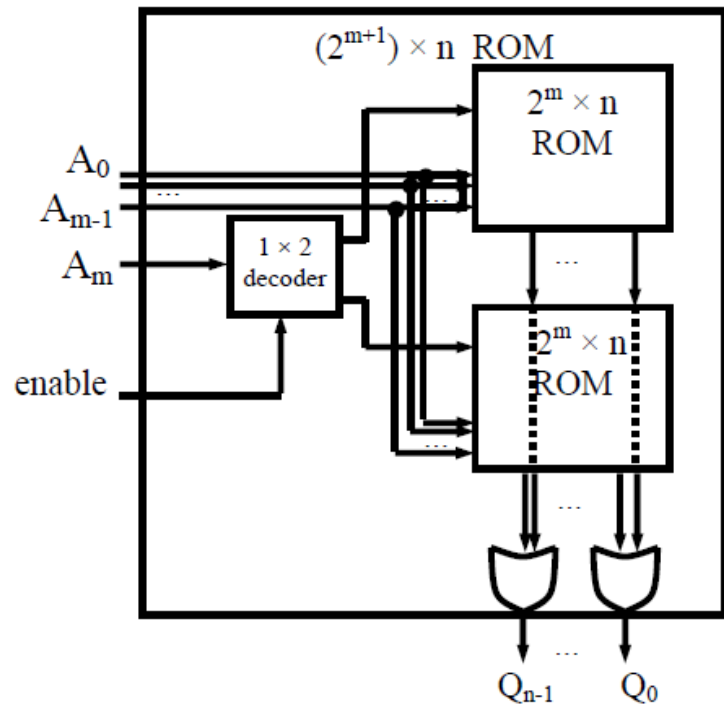


timing diagrams

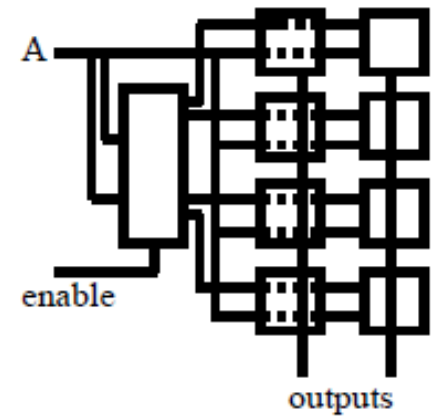
## Composing memory

- ▶ Memory size needed often differs from size of readily available memories
  - When available memory is larger, simply ignore unneeded high-order address bits and higher data lines
  - When available memory is smaller, compose several smaller memories into one larger memory
    - Connect side-by-side to increase width of words
    - Connect top to bottom to increase number of words
  - added high-order address line selects smaller memory containing desired word using a decoder
  - Combine techniques to increase number and width of words

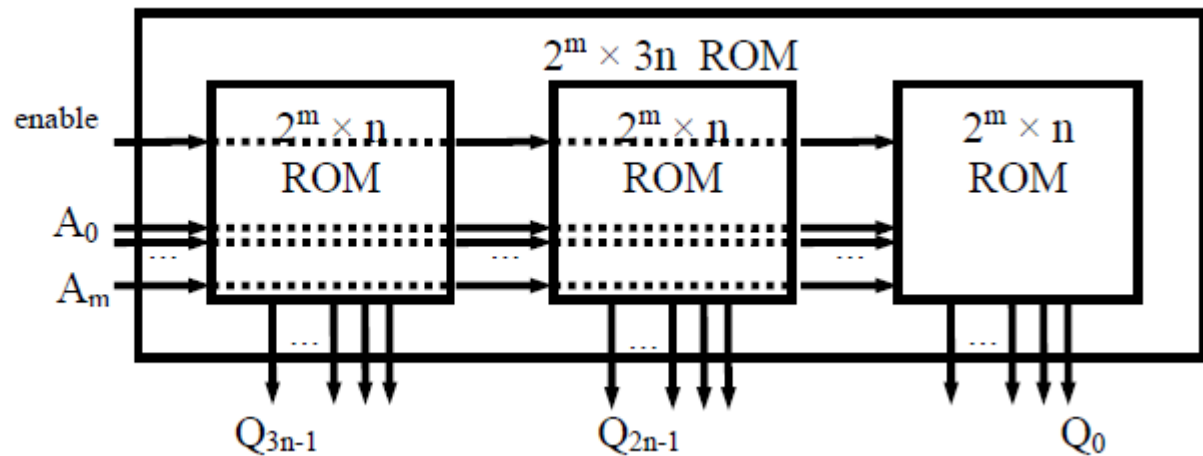
**Increase number of words**



**Increase number**  
**and width of**  
**words**

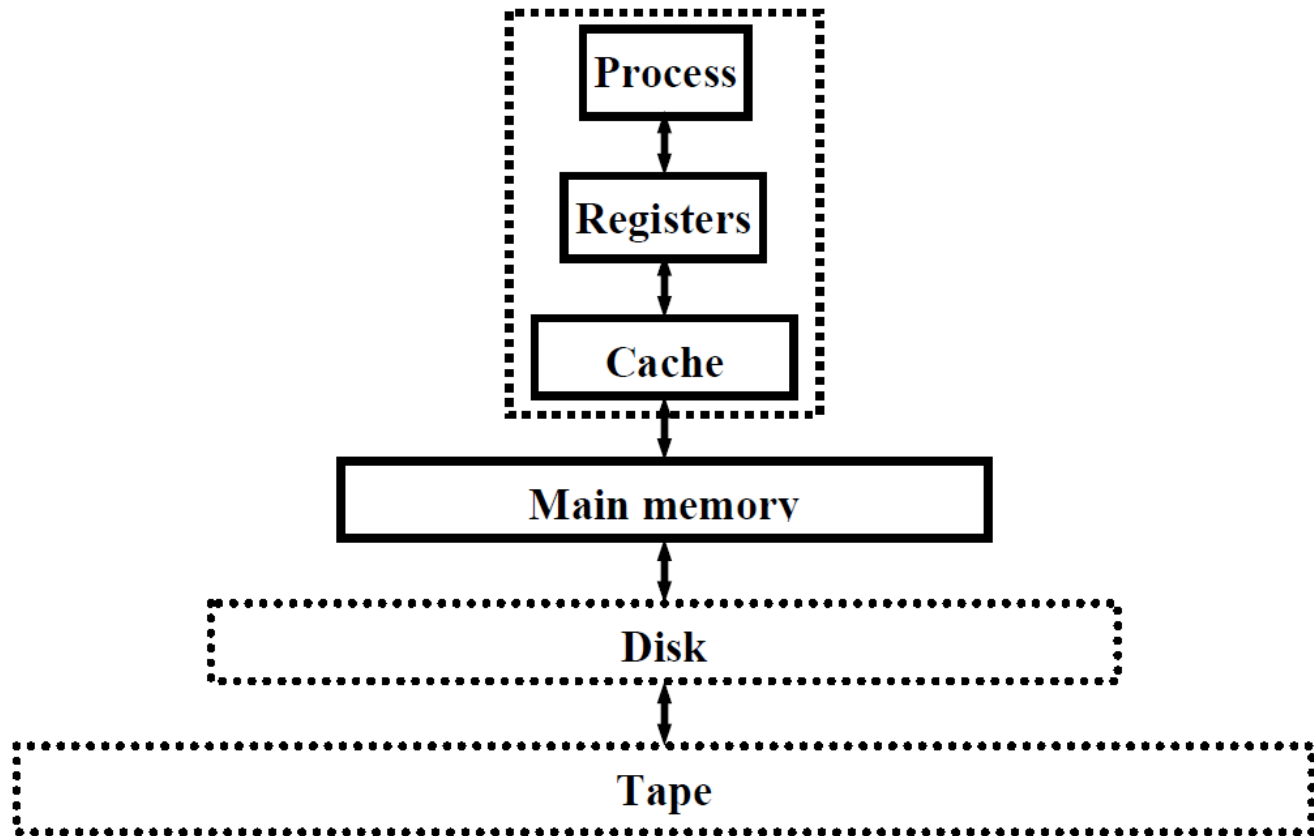


**Increase width**  
**of words**



# Memory hierarchy

- ▶ Main memory
  - Large, inexpensive, slow memory stores entire program and data
- ▶ Cache
  - small, expensive, fast memory stores copy of likely accessed parts of larger memory
  - can be multiple levels of cache






## Cache memory:


- ▶ Processors perform operations on operands faster than the access time of main memory.

*How can we increase the speed of access to the main memory?*

- ▶ A main memory made from fast ram chips would be financially expensive, take up a lot of board space, and consume a large amount of energy.
- ▶ Increase in access speed can be achieved by the use of a relatively small, but Fast, additional memory, called a cache.
- ▶ A **cache memory** is made of fast, static RAM, and be large enough to hold the currently active clusters of instruction code and data that the microprocessor needs.

- ▶ The microprocessor will access the cache rather than main memory and so will have faster access to the code and data that it requests.
  - ▶ To further improve the access time to the cache, we will either place the cache on the same chip as the microprocessor, or provide a separate, short-length, high speed, bus between the microprocessor and the cache.
  - ▶ This will avoid the delays inherent in driving signals on long bus wires.
- 

## Basic operation of cache

- ▶ Assume that the cache has been partly filled by copying parts of main memory into it. When the microprocessor makes a request to read a particular word (instruction code, or data) in main memory, the address of the word appears both at main memory and at the cache.
  - ▶ If the requested word is present in the cache, the cache hardware will assert a signal, *hit*. If *hit* is asserted, the required word will be read from the cache, otherwise, the word will be read from main memory.
- 

- ▶ Consider the execution of the following loop written in G80 code and assuming we have a cache.

```
        ...form sum of 200 numbers at (hl)
        ld b, 200
        xor a
again:   add a, (hl)
        inc hl
        djnz again
```

Improve execution speed by:

- ▶ Temporal locality of reference property
  - Loop consists of 3 instruction which is executed 200 times.
- ▶ Spatial locality of reference property
  - Cache a block or line of main memory location

## Average access time of memory

Let:  $t_s$  = average access time of memory system

$t_c$  = access time of cache

$t_M$  = access time of main memory

The *hit ratio*,  $h$ :

$$h = \frac{\text{number of times required word is in the cache}}{\text{total number of memory references}}$$

- ▶ Then, a cache hit gives access in  $t_c$  while a cache miss gives access in  $t_c + t_M$  since both the cache and main memory are accessed.

Therefore the average access time of memory is

$$t_s = h.t_c + (1 - h).(t_c + t_M) = t_c + (1 - h).t_M.$$


For example, say  $t_C = 10\text{ns}$  and  $t_M = 50\text{ ns}$ .

For a hit ratio,  $h = 0.80$ :  $t_S = 10 + 0.2 \times 50 = 20\text{ ns}$


- ▶ The average access time of memory system is reduced from 50 ns with no cache to 20 ns with the cache.
- ▶ If the hit ratio is 0.90, the average access time is .....
- ▶ when a hit ratio is 0.98, the average access time is .....

*The engineering challenge is to make the hit ratio as close to 1 as possible, since then the average access time approaches that of the cache memory.*

## Design issues:

- How is the data to be organized in the cache, and how is the computer to determine which parts of main memory to store in the cache?
  - If the microprocessor writes to the cache, so modifying its contents, when should the main memory be updated?
- 

## Cache Mapping


- is necessary as there are far fewer number of available cache addresses than the memory
  - Are address' contents in cache?
  - Cache mapping used to assign main memory address to cache address and determine hit or miss
  - Three basic techniques:
    - Direct mapping
    - Fully associative mapping
    - Set-associative mapping
- 



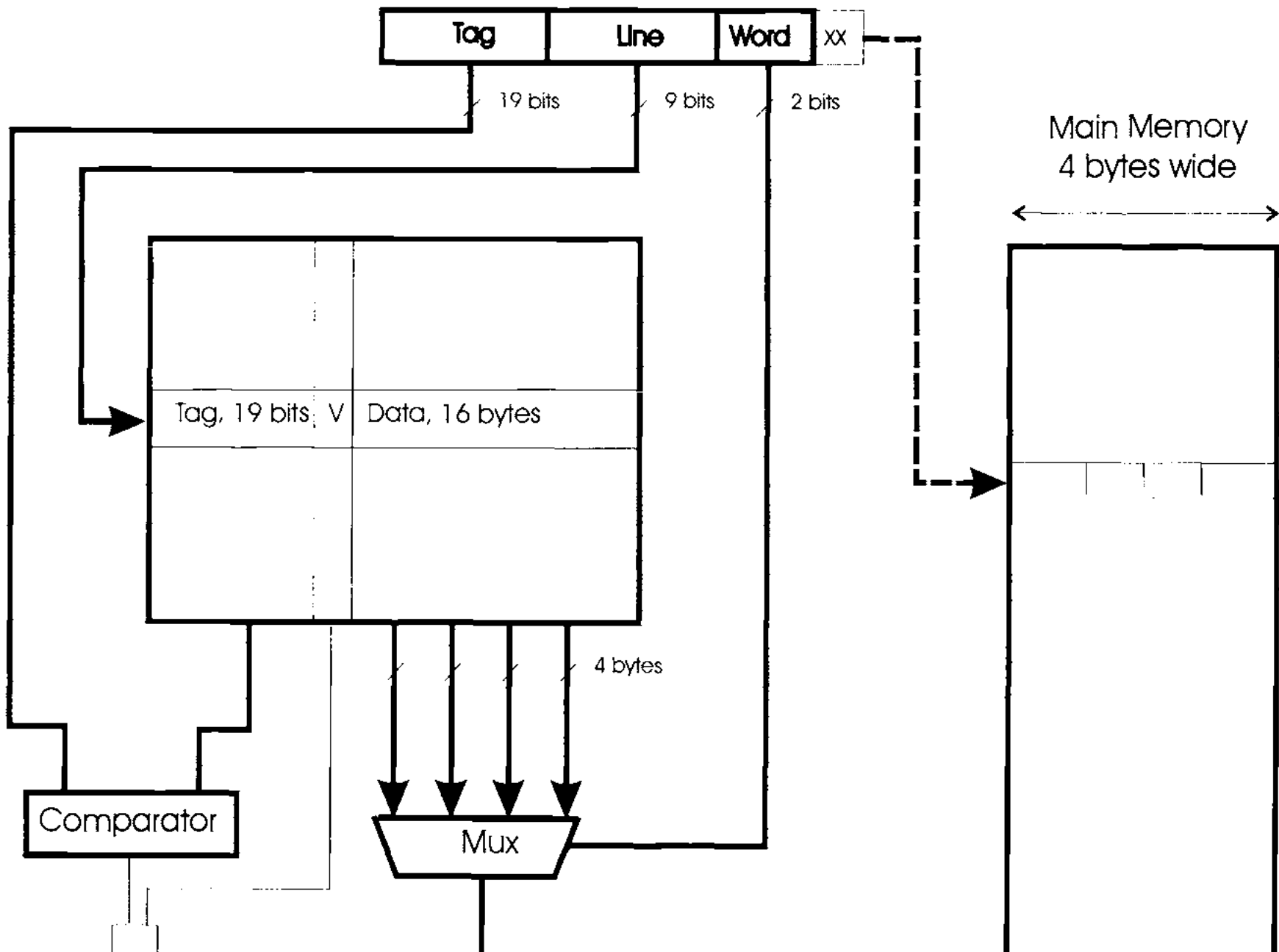
## Cache organization -direct mapping

We assume that the cache contains 8 KB of data. Now, since following a Cache miss we will copy a line of 16 bytes from main memory into the cache, we let each cache storage location hold 16 bytes. Therefore, the cache contains  $8K/16 = 512$  lines, each line containing 16 bytes.


Our computer has a 32-bit address bus and can access a 4-byte word over its 32-bit data bus. Only the upper 30 bits of an address appear on the address bus since main memory is organized into 4-byte words. How are we to use the upper 30 bits of the address from the microprocessor to select one of the 512 lines in the cache?



Address from microprocessor



In **direct mapping** of the address from the microprocessor to a cache address:

- ▶ The 9 bits, A12 to A4, of the address from the microprocessor are used as the cache address.
  - ▶ **Tag:** the upper 19 bits of the address microprocessor is in the cache in addition to the line of data.
  - ▶ The address from the microprocessor is regarded as comprising a 19-bit tag, a 9-bit line number, and a 2-bit word selector, A1 and A0 not being used.
  - ▶ On power up or reset, the cache is flushed by setting all the valid bits to zero; when the line in the cache is filled with data, the valid bit for the line is set to one. This bit is ANDed with the output of the comparator thereby causing the hit signal to be de-asserted if the cache line is invalid.
- 

A31.....A13	A12.....A4	A3.A2
Tag 19 bits	Line 9 bits	Word 2 bits

Partitions of address

Tag 9 bits	Valid 1 bit	Data Four, 4-byte words = 16 bytes.
---------------	----------------	--

Cache line entry

- ▶ When the microprocessor **generates an address**, the line number part of the address is used as the **cache address**.
- ▶ The content of the addressed line in the cache is read out and a hardware comparator compares the tag in the address from the microprocessor with the tag stored in the cache. If **they are the same**, the data in the cache is that requested by the microprocessor, and the **hit signal is asserted**.
- ▶ At the same time, the **2-bit word selector** is input to a 4-way selector that selects the required 4-byte word from the 16 bytes stored in the cache line. Thus, the requested word is made available to the microprocessor.
- ▶ If hit is not asserted (**a cache miss**), the word is read from main memory, and the 16-byte aligned 4 memory line that contains the word is read in a single burst from main memory and stored in a cache line.

## Critique

- ▶ The direct mapping of address from the microprocessor to a cache address is **simple, which implies that it can be implemented at low cost in high-speed hardware**. It has been used in many commercial computers.
- ▶ However, consider two frequently occurring addresses from the microprocessor both of which map to the same cache location. For example, 0000 0000 0000 0000 0001 0000 0000 0000 and 0000 0000 1111 0000 0001 0000 0000 0000 both map to cache line 256.
- ▶ The cache line will be **replaced every time one of these addresses occurs resulting in a dramatic drop in the hit ratio**. We *could reduce* the probability of these repeated collisions by making the cache larger; however, we will seek out other ways of organizing a cache.

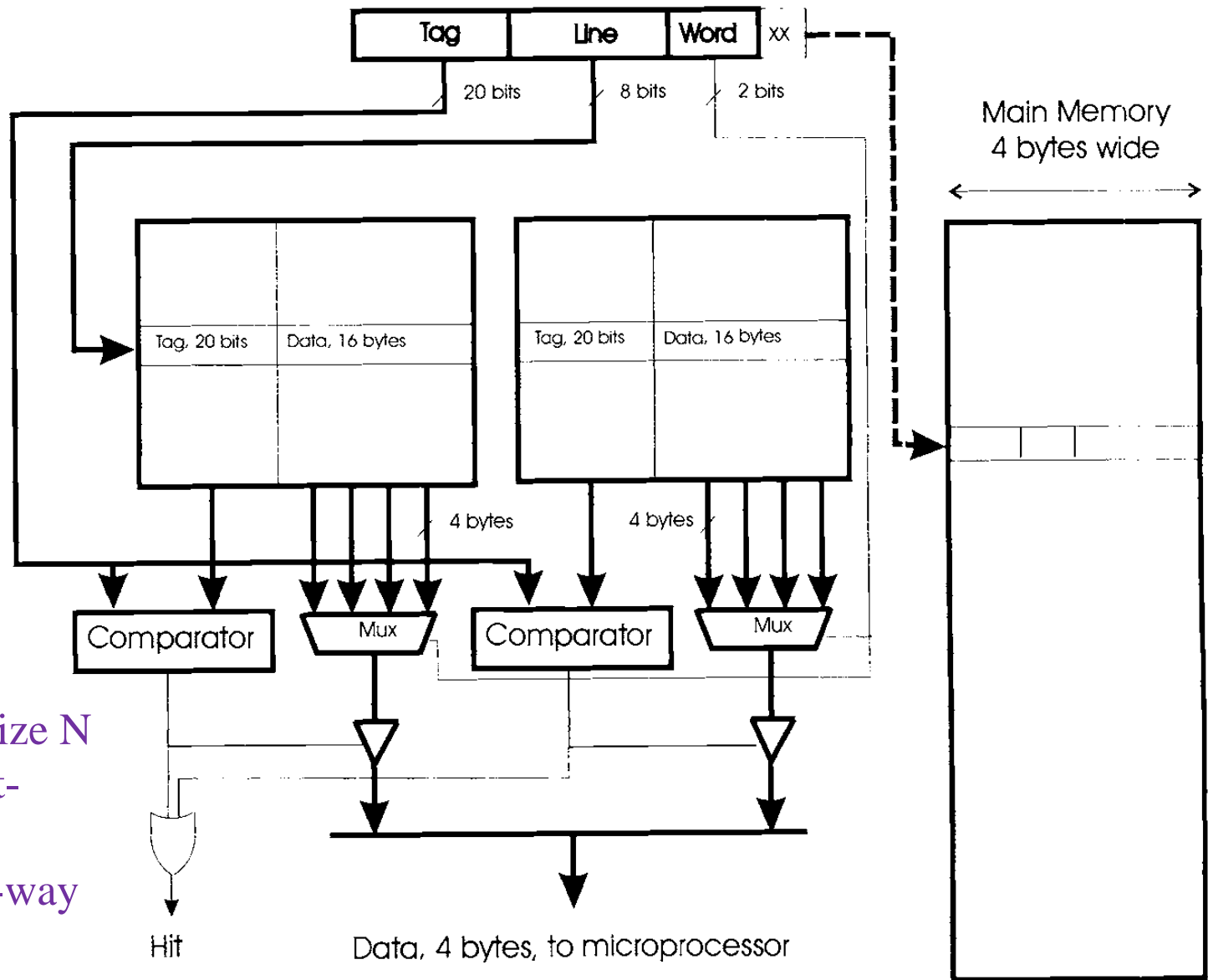
## Cache organization - set-associative mapping

- ▶ The drawback of direct mapping is overcome by placing a number of directly mapped caches side by side.
- ▶ Assume that a line of data comprises 16 bytes, but now two lines of data are stored in each cache location. **The two lines in each cache location are collectively called a set.** Now, when two addresses from the microcomputer map to the same cache set, both lines of data can be stored in the set.
- ▶ There will still be a drop in hit ratio if more than two frequently occurring addresses from the microcomputer map to the same cache set; the solution is to increase the number of lines in each set to, say, four, giving a four-way set.
- ▶ The operation of this cache is the same as for the directly mapped cache except that there are two *hit signals* and *not more than one* of these will be asserted and so can be used to determine which of the two lines in the set contains the required data.

Address from microprocessor

**Fig. Two way set  
associative  
cache**

Cache with set size N  
called N-way set-  
associative  
2-way, 4-way, 8-way  
are common





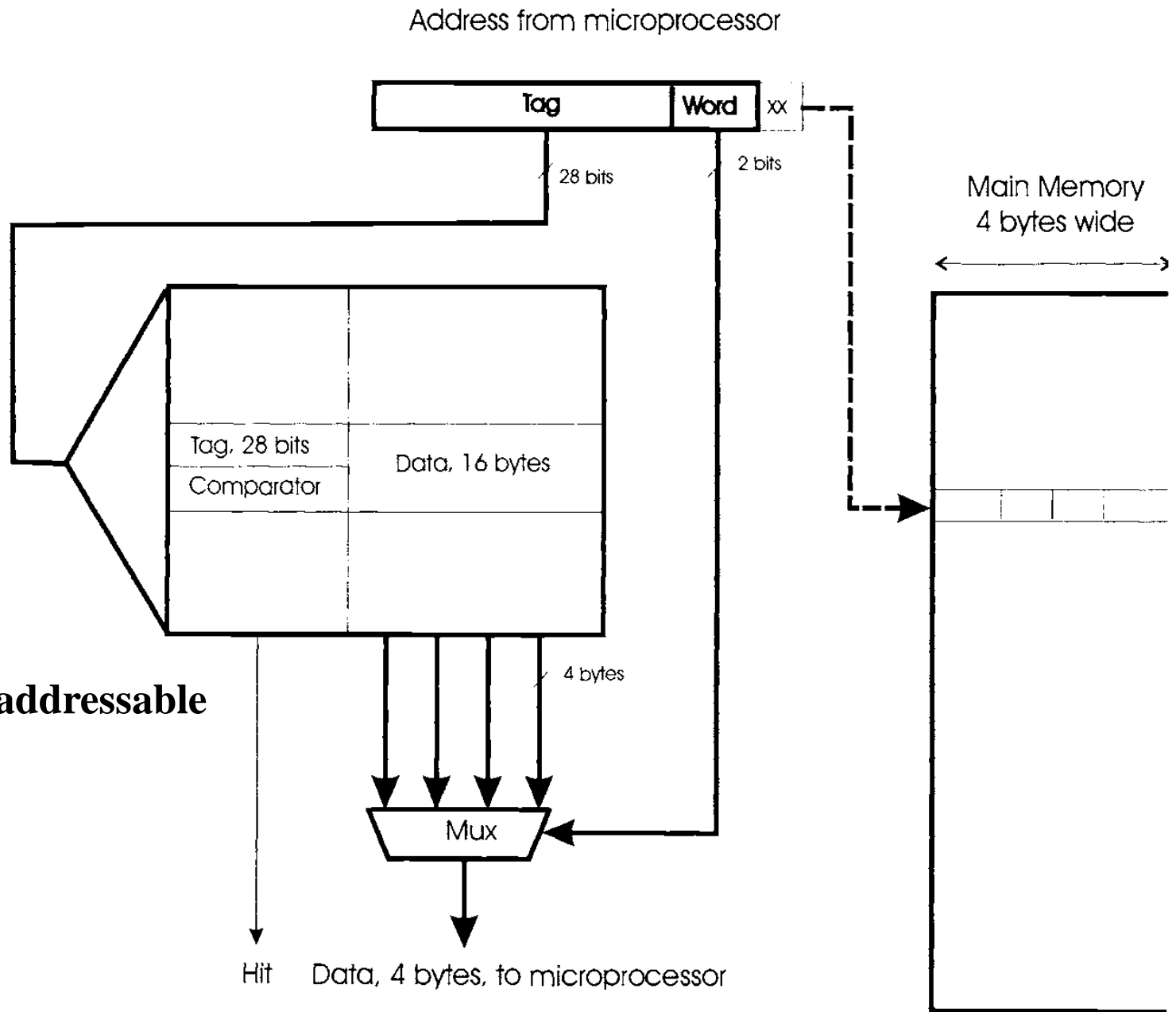
A31.....A11	A10.....A4	A3.A2
Tag	Set	Word
21 bits	7 bits	2 bits

### Partitions of address

- ▶ Assume a **four-way set**, each location in the cache stores four lines of data, and each line contains 16 bytes.
- ▶ For a total cache data size of 8 KB, there will be **128 locations, since 128 locations x 4 lines/location x 16Bytes/line = 8 KB.**
- ▶ The address of one of 128 locations in the cache will be obtained from the 7 bits, A10 to A4, in the address from the microprocessor. Thus, the address from the microprocessor is regarded as comprising a 21-bit tag, a 7-bit line number, and a 2-bit word selector, A1 and A0 not being used.

## Cache organization - fully associative mapping

- ▶ Store the address in the cache along with its associated data.
- ▶ Assume , the microprocessor produces a 32-bit address and that each cache location stores a 16-byte line of data. Then, the address from the microprocessor is regarded as comprising a 28-bit tag, and a 2-bit word selector, A1 and A0 not being used.
- ▶ *Very flexible* since any address from the microprocessor can be stored anywhere in the cache.
- ▶ Associative memory is very expensive in terms of transistors per location, because of the comparator circuit for each location.
- ▶ Therefore this form of cache organization is currently *used only for the small caches that are used in other parts of the microprocessor.*



**Fig. Content-addressable memory**

A31.....A4	A3.A2
Tag 28 bits	Word 2 bits

### Partitions of address

Valid 1 bit	Tag 28 bits	Data 16 bytes
----------------	----------------	------------------

### Cache Entry

- ▶ when the microprocessor requests a memory address, we see if the required data is in the cache by **comparing all the tags stored in the cache with the upper 28 bits of the address** from the microprocessor. If there is a match, we have a cache hit and the data in the cache may be accessed.
- ▶ The comparison of all the tags stored in the cache with the requested main memory address would be unacceptably slow unless done by hardware.
- ▶ This requires a special form of memory called an **associative memory or content addressable memory, CAM** in which each cache location has a comparator circuit attached to it.
- ▶ The upper 28 bits of the address from the microprocessor are presented to the CAM and all the comparators in CAM simultaneously compare these 28 bits with the tags stored in the CAM. Where there is a match, the *hit signal is asserted and the data in the cache location* is output from the CAM.

## Memory write operations


- ▶ When the microprocessor performs a memory write operation, and **the word is not in the cache**, the new data is simply written into main memory.
- ▶ When the **word is in the cache**, both the word in main memory and the cache must be written in order to keep them the same.

*when is the main memory to be written?*

- ▶ The simplest answer is to “**write to both the cache and main memory at the same time. This is called a write through policy**”. The main memory then always contains the same data as the cache.
- ▶ This is important if there are any other devices in the computer that also access main memory.
- ▶ We can overcome **the slowing down** due to the main memory write operation by providing that subsequent cache reads proceed concurrently with the write to main memory. Since more than 70% of memory references are read operations, it is likely that the cache can continue to be read while the write to main memory proceeds.

- ▶ An alternative policy, called **write-back or copy-back**, is to write the new data to the cache only. At the same time, **a flag in the cache line is set** to indicate that the line has been modified. Immediately before the cache location is **replaced with new words from main memory**, if the flag is set, **the line will be copied back into main memory**. Of course, if the flag is not set, this copying is unnecessary.

## Line replacement/cache-replacement policy

- ▶ When a requested word is not in the cache, a new line of data is copied from main memory into the cache.
  - ▶ Assuming a four-way set, bits A10 to A4 of the address from the microprocessor indicate the cache location where the new line is to be stored, but **which of the four lines in the set stored at that location is to be replaced?** This decision must be made entirely by hardware because software will be much too slow.
  - ▶ Clearly, if the valid bit in a line indicates that the line is not in use, that line is the one to be replaced. However, **when the valid bits indicate that all four of the lines are in use, a policy for the replacement of a line is needed.**
- 



## Random replacement policy:

- **Randomly** a line within the selected set is chosen.
- Since we need a random number between 0 and 3, the policy can be implemented by having a single 2-bit counter that is incremented whenever a particular operation occurs. The current count is used to select the line within the set.

## Least recently used policy:

- A counter is associated with each line (four counters in our four-way set cache). When a line is referenced, its counter is set to zero while the other three counters are incremented. Each line count indicates the age of the line since it was last referenced; **the line with the highest count is the oldest and is the one to be replaced.**
- This is expensive to implement in hardware

## Approximation to the LRU policy that is simpler to implement.

- ▶ For ex. In two-way set-associative cache: store a single bit, B, in the set to indicate which line was last used. Call the two lines in the set L0 and L1. Then, when **L0 is accessed, bit B is set to 1**, else it is set to 0. The bit thus indicates which line was most/least recently accessed.
- ▶ In four-way sets divide the four lines into two pairs. Let there be three LRU bits, B0, B1, and B2. These are all set to 0 when the cache is flushed, and are updated on every cache hit or replacement. Call the four lines in the set, L0, L1, L2, and L3. Divide these four lines into two pairs of lines, pair **L01 comprising lines L0 and L1**, and pair L23 comprising lines L2 and L3. Let bit B0 indicate whether pair L01 or L23 was last accessed. That is, if **either L0 or L1 is accessed, B0 is set to 1**, while if either L2 or L3 is accessed, B0 is set to 0. Let bit B1 indicate which line in pair L01 is accessed. That is, if **L0 is accessed, B1 is set to 1**, else B1 is set to 0. Similarly, bit B2 indicates which line in pair L23 is accessed. That is, if **L2 is accessed, B2 is set to 1**, else B2 is set to 0.

## Cache Performance Trade-Offs

- Improving cache hit rate without increasing size
  - Increase line size
  - Change set-associativity

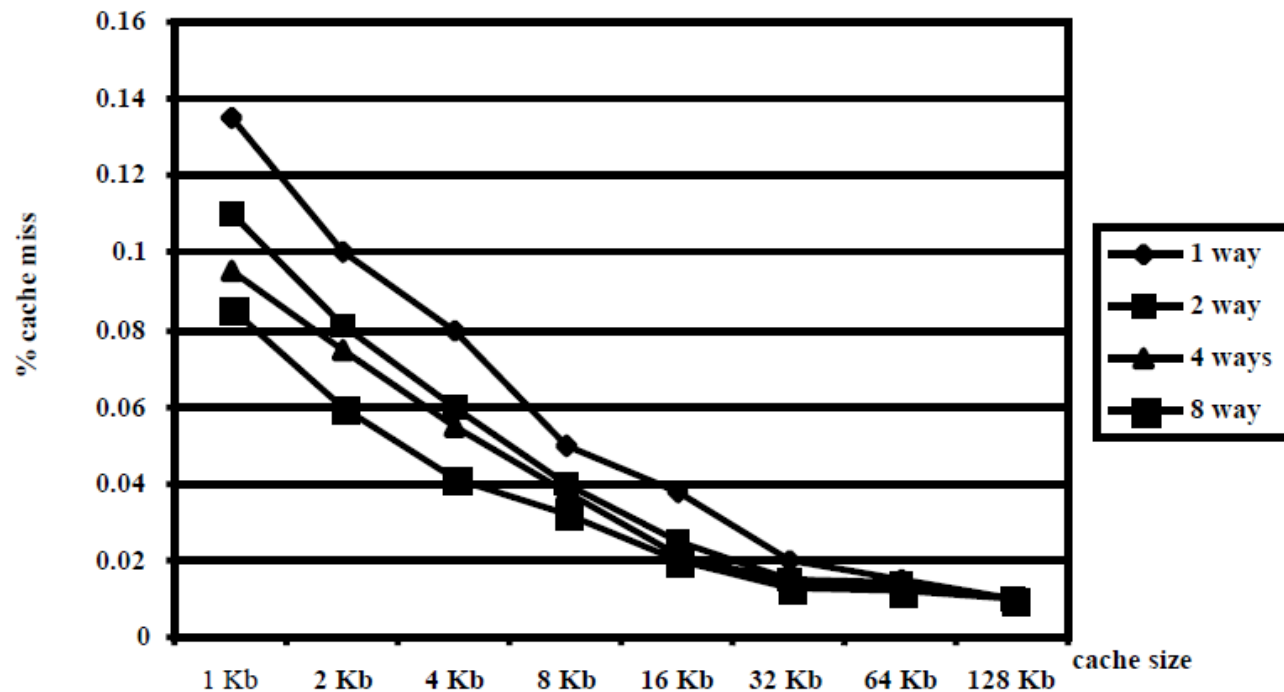


Fig. 6.5 Cache Performance

# Memory Management Units and Address Translation



## ► What is memory management?

- Memory management describes how access to memory in a system is **controlled**. The hardware performs memory management every time that memory is accessed by either the OS or applications. *Memory management is a way of dynamically allocating regions of memory to applications.*

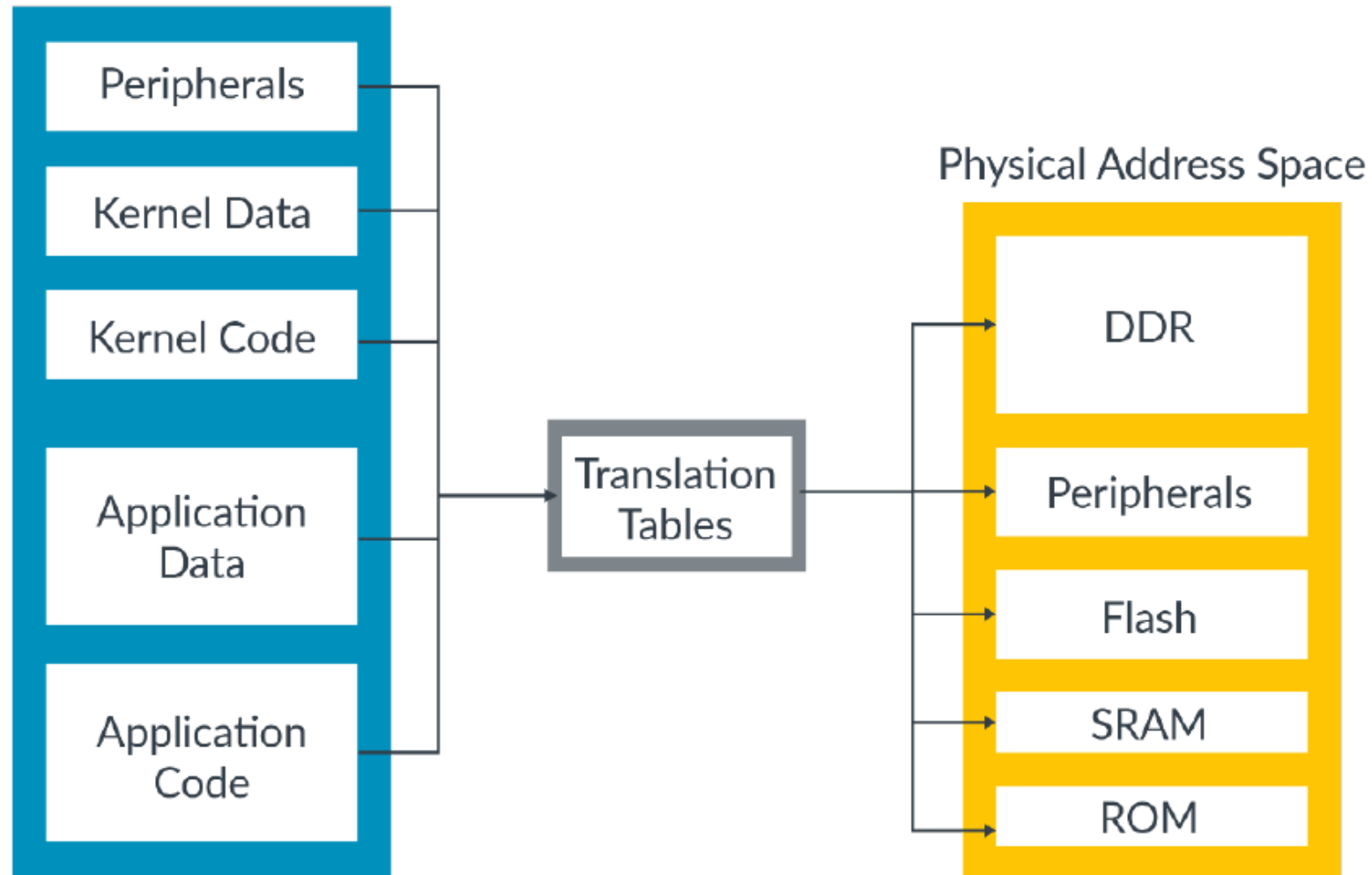
## ► Why is memory management needed?

- Application processors are designed to run a rich OS, such as Linux, and to *support virtual memory systems*. *Software that executes on the processor only sees virtual addresses, which the processor translates into physical addresses.* These physical addresses are presented to the memory system and point to the actual physical locations in memory.

## ► Benefits of using virtual addresses :

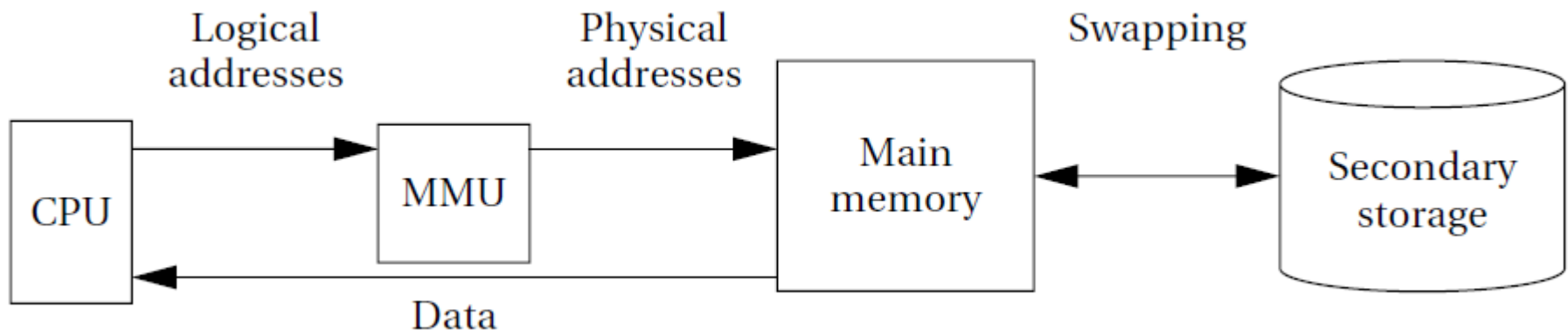
- It allows management software, such as an operating system (OS), to **control the view of memory** that is presented to software. The OS can control **what memory is visible**, the virtual address at which that memory is visible, and **what accesses are permitted to that memory**.
- An OS can present **multiple fragmented physical regions of memory as a single, contiguous virtual address space to an application**.
- Software developers, who will not know a system's exact memory addresses when writing their application. With virtual addresses, software developers do not need to concern themselves with the physical memory. **The application knows that it is up to the OS and the hardware to work together to perform the address translation**

## Virtual Address Space (Addresses seen by software)



Translation tables are in memory and are managed by software, typically an OS. The translation tables are not static, and the tables can be updated as the needs of software change. This changes the mapping between virtual and physical addresses.

- ▶ A MMU translates addresses between the CPU or application software and physical memory. This translation process is often known as *memory mapping* since *addresses are mapped from a* logical space into a physical space.
- ▶ MMUs in embedded systems appear primarily in the host processor.
- ▶ MMUs allowed software to manage multiple programs in a single physical memory, each with its own address space.

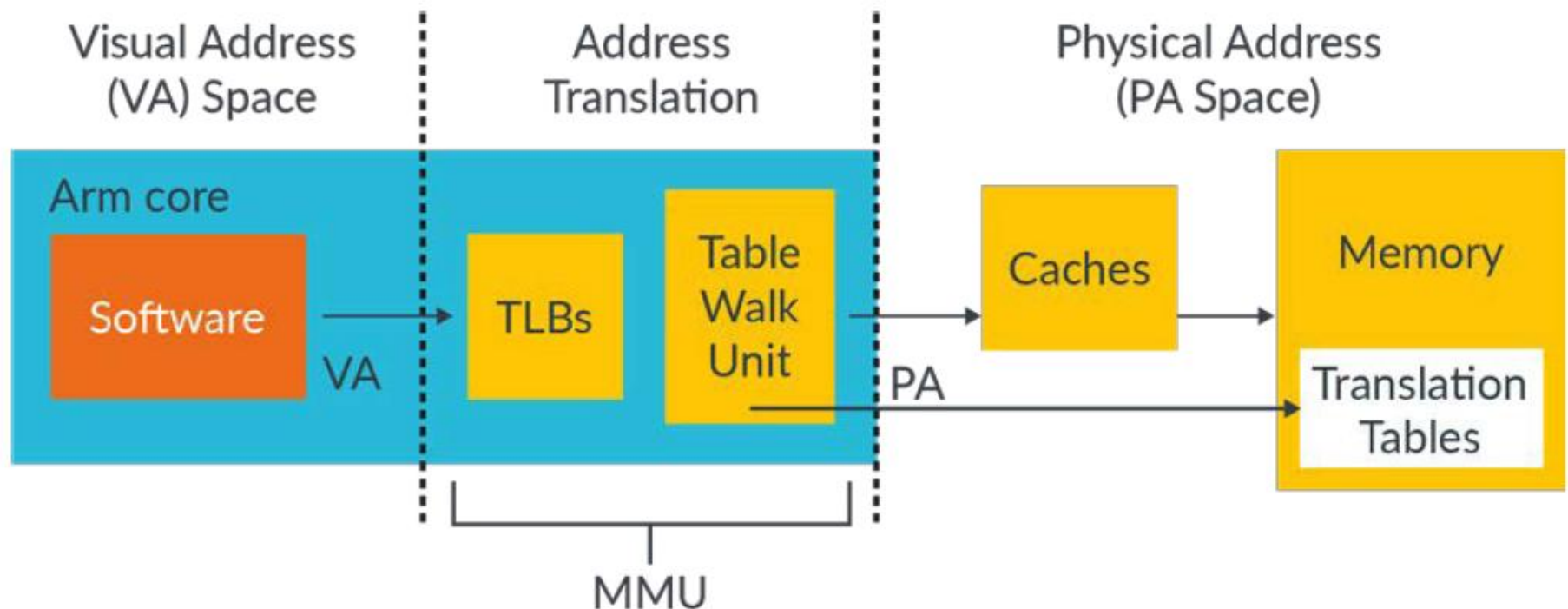


A virtually addressed memory system




The MMU contains the following:

- The table walk unit, which contains logic that reads the translation tables from memory.
- Translation Lookaside Buffers (TLBs), which cache recently used translations.



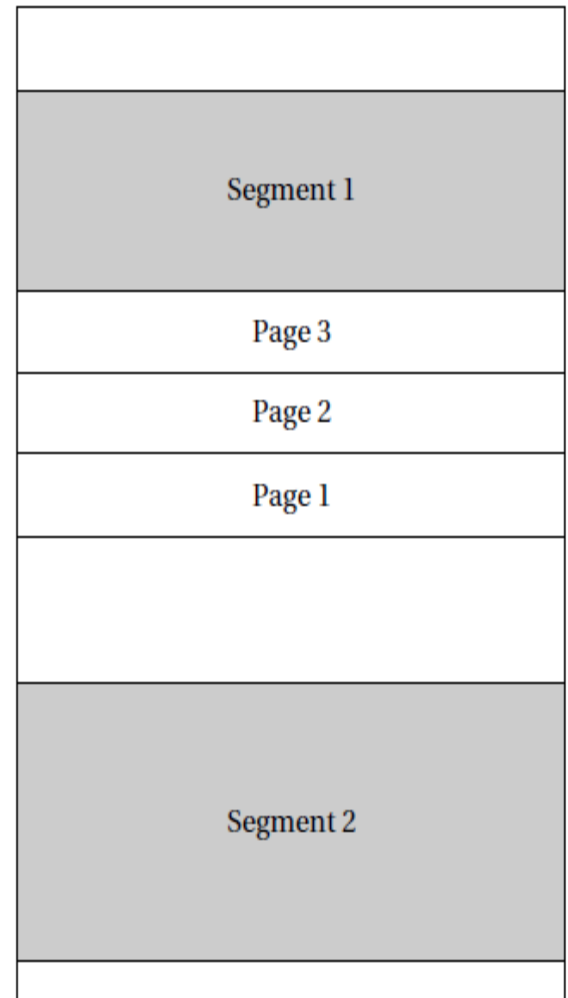
- ▶ All memory addresses that are issued by software are virtual. These memory addresses are passed to the MMU, which checks the TLBs for a recently used cached translation. If the MMU does not find a recently cached translation, the table walk unit reads the appropriate table entry, or entries, from memory

- ▶ MMUs are used to provide *virtual addressing*.
- ▶ In a **virtual memory system**, the MMU keeps track of which logical addresses are actually resident in main memory; those that do not reside in main memory are kept on the secondary storage device.
- ▶ When the CPU requests an address that is not in main memory, the MMU generates an exception called a *page fault*.
- ▶ *The handler for this exception executes code that* reads the requested location from the secondary storage device into main memory.
- ▶ The program that generated the page fault is restarted by the handler only after
  - the required memory has been read back into main memory, and
  - the MMU's tables have been updated to reflect the changes.

- ▶ Of course, loading a location into main memory will usually require throwing something out of main memory. The displaced memory is copied into secondary storage before the requested location is read in.
  - ▶ There are two styles of address translation:
    - *segmented*
    - *Paged*
    - *Each has* advantages and the two can be combined to form a *segmented, paged addressing scheme*.
- 


- ▶ **Segment** is designed to support a large, arbitrarily sized region of memory, while **pages** describe small, equally sized regions.
- ▶ A segment is usually described by its start address and size, allowing different segments to be of different sizes.
- ▶ Pages are of uniform size, which simplifies the hardware required for address translation.
- ▶ **A segmented, paged scheme** is created by dividing each segment into pages and using two steps for address translation.

Physical  
memory

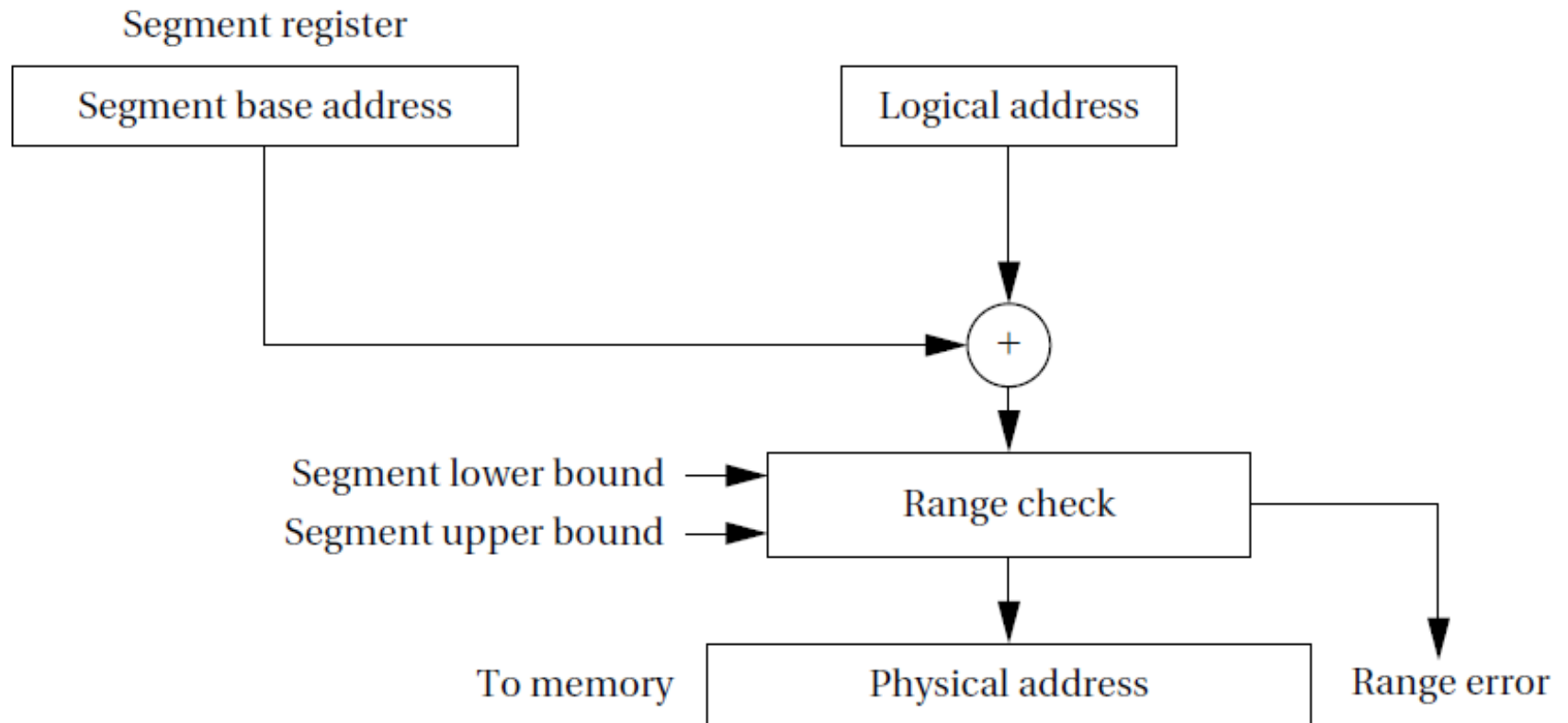


Segments and pages


- ▶ Virtual memory is typically implemented in a paging or segmented, paged scheme. So that only page-sized regions of memory need to be transferred on a page fault.
- ▶ Some extensions to both segmenting and paging are useful for virtual memory:
  - At minimum, a **present bit** is necessary to show whether the logical segment or page is currently in physical memory.
  - A **dirty bit** shows whether the page/segment has been written to. This bit is maintained by the MMU, since it knows about every write performed by the CPU.
  - **Permission bits** are often used. Some pages/segments may be readable but not writable. If the CPU supports modes, pages/segments may be accessible by the supervisor but not in user mode.

- ▶ In a segmenting scheme, the MMU maintain a segment register that describes the currently active segment. This register would point to the base of the current segment.
  - ▶ The logical address provide the offset for the address.
  - ▶ The physical address is formed by adding the segment base to the offset.
  - ▶ Most segmentation schemes also check the physical address against the upper limit of the segment by extending the segment register to include the segment size and comparing the offset to the allowed size.
- 

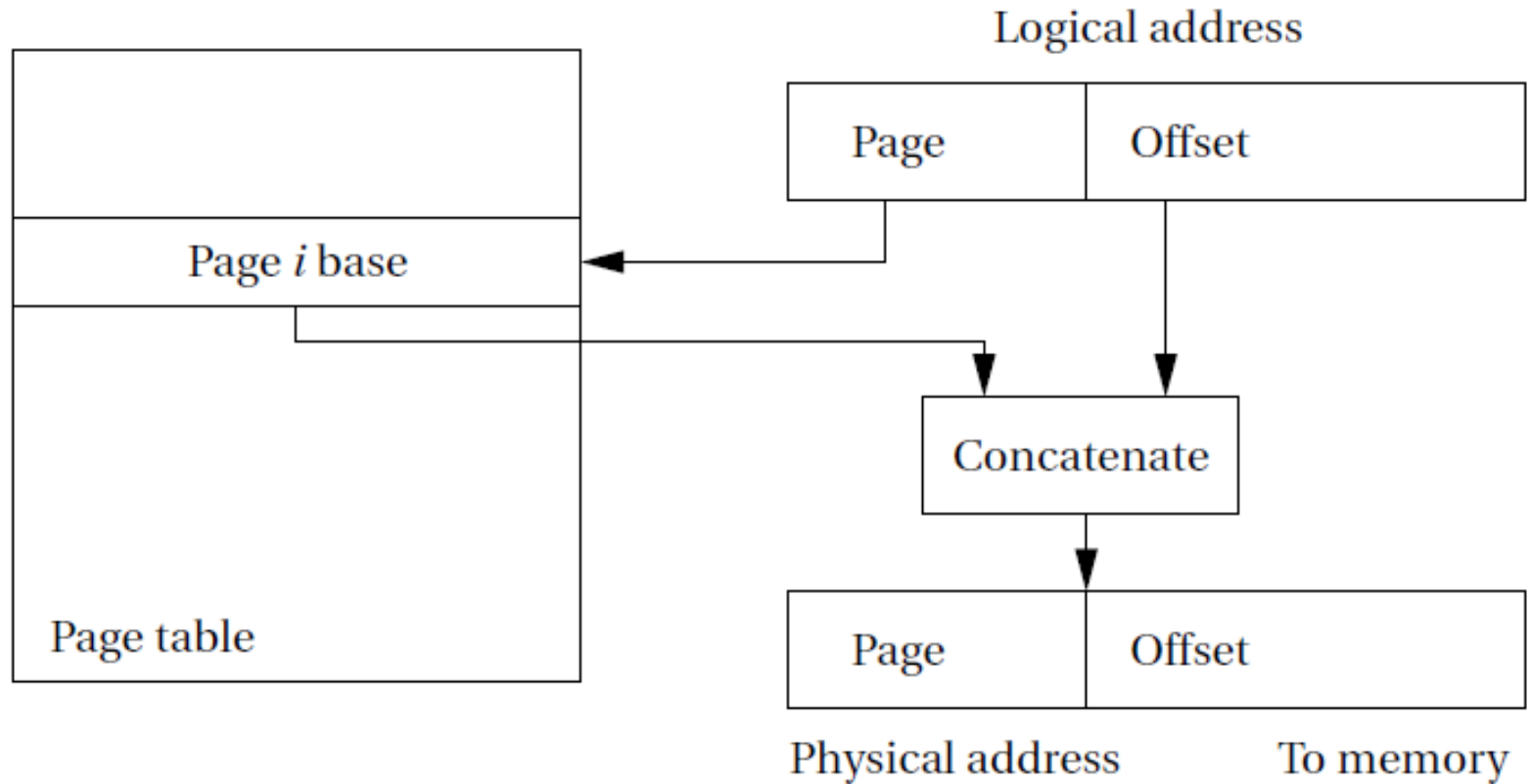
# Address translation for a segment.






- ▶ In Page scheme, the logical address is divided into two sections : a page number and an offset. The page number is used as an index into a page table, which stores the physical address for the start of each page.
  - ▶ since all pages have the same size, the MMU just concatenate the top bits of the page starting address with the bottom bits from the page offset to form the physical address.
  - ▶ Pages are small, typically between 512 bytes and 4 KB. As a result, the page table is large for an architecture with a large address space.
  - ▶ The page table is normally kept in main memory , which means that an address translation requires memory access.
- 

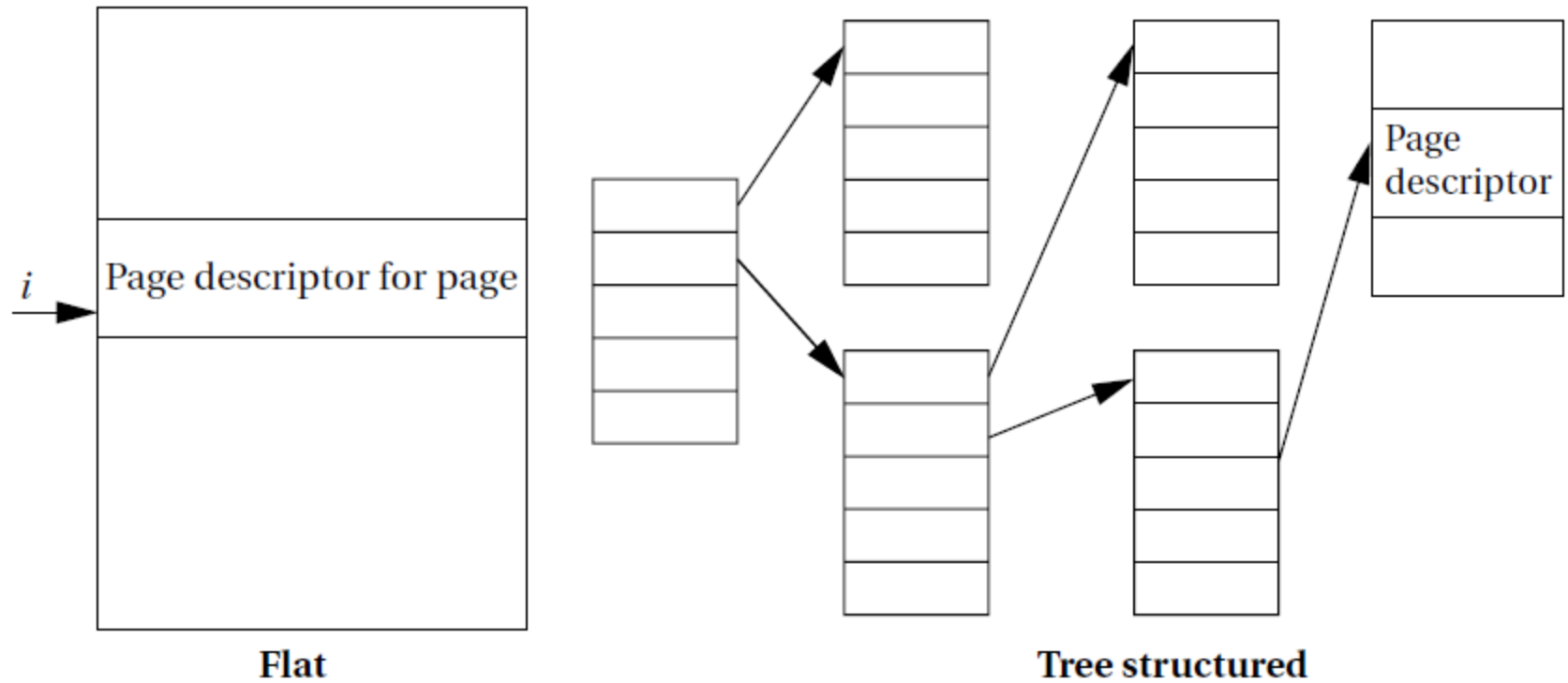
# Address translation for a page.



The page table may be organized in several ways:

- The simplest scheme is a flat table. The table is indexed by the page number and each entry holds the page descriptor.
  - A more sophisticated method is a tree. The root entry of the tree holds pointers to pointer tables at the next level of the tree; each pointer table is indexed by a part of the page number. We eventually (after three levels, in this case) arrive at a descriptor table that includes the page descriptor we are interested in. A tree-structured page table incurs some overhead for the pointers, but it allows us to build a partially populated tree. If some part of the address space is not used, we do not need to build the part of the tree that covers it.
- 

# Alternative schemes for organizing page tables.



# Tutorial1

Q1. A computer has a single cache (off-chip) with a 2 ns hit time and a 98% hit rate. Main memory has a 40 ns access time. What is the computer's effective or average access time? If we add an on-chip cache with a .5 ns hit time and a 94% hit rate, what is the computer's effective access time? How much of a speedup does the on-chip cache give the computer?

Q2. A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses word level addressing. What is the address format? If we change the cache to a 4-way set associative cache, what is the new address format?

Q3. A computer employs a 32-bit address space, a 32-bit data bus, and a cache that holds 4 KB of data.

Determine the number of bits in the tag, line, and byte fields of the address from the microprocessor, assuming the following organizations:

(a) Direct mapping with a line size of 8 bytes.

(b) Direct mapping with a line size of 16 bytes.

*Note: the block or line of main memory consists of 4bytes or 32bit of data.*

Thank You