L	T	P	C
3	0	3	4

19CSE303 – Embedded Systems

L	T	P	C
3	0	3	4

Course Introduction

Unit 1

Basics of Embedded Systems –Definition, Characteristics, Architecture of Microprocessors: General definitions of computers, micro-processors, micro controllers and digital signal processors. ARM Architecture: RISC Machine, Architectural Inheritance, Programmers model, CPSR, ARM Organization and Implementation. 3 Stage pipeline, 5 Stage pipeline, ARM Instruction execution, Co-processor interface, ARM Assembly language Programming, Data processing instructions, Data Transfer Instructions, Control flow instructions, Architectural support for high level programming, Thumb Instruction set.

Unit 2

Interrupt structure -Vector interrupt table, Interrupt service routines, Asynchronous and Synchronous data transfer schemes, ARM memory interface, AMBA interface, Microcontroller ARM7-LPC2148 ports: GPIO, A/D Converters, PWM, timer / counter, UART and its interfacing – Embedded Programming Concepts: Role of Infinite loop, Compiler, Assembler, Interpreter, Linker, Loader, Debugger-Application development using Keil IDE.

Unit 3

Introduction to ARM Cortex M4 Microcontroller –GPIO and other Peripherals -System development process: Requirements, Design, Development, Testing and Deployment. Prototyping: Analog circuit design and construction on a solderless breadboard, Hardware and Software design, Programming simple logic and testing PLL and Systick Timers, Design strategy for building Finite State machine.

Course Introduction

- **Text Book(s):**

Furber SB. ARM system-on-chip architecture. pearson Education; 2000.Martin T. The Insider's guide to the Philips ARM7-based microcontrollers. Coventry, Hitex, UK, Ltd. 2005.

- **Reference(s):**

- Valvano JW. Embedded Systems: Introduction to ARM Cortex-M Microcontrollers. Jonathan W. Valvano; 2016.
- Valvano JW. Embedded microcomputer systems: real time interfacing. Cengage Learning; 2012.

Course Introduction

- **Evaluation Pattern (65 : 35)** **(TENTATIVE)**

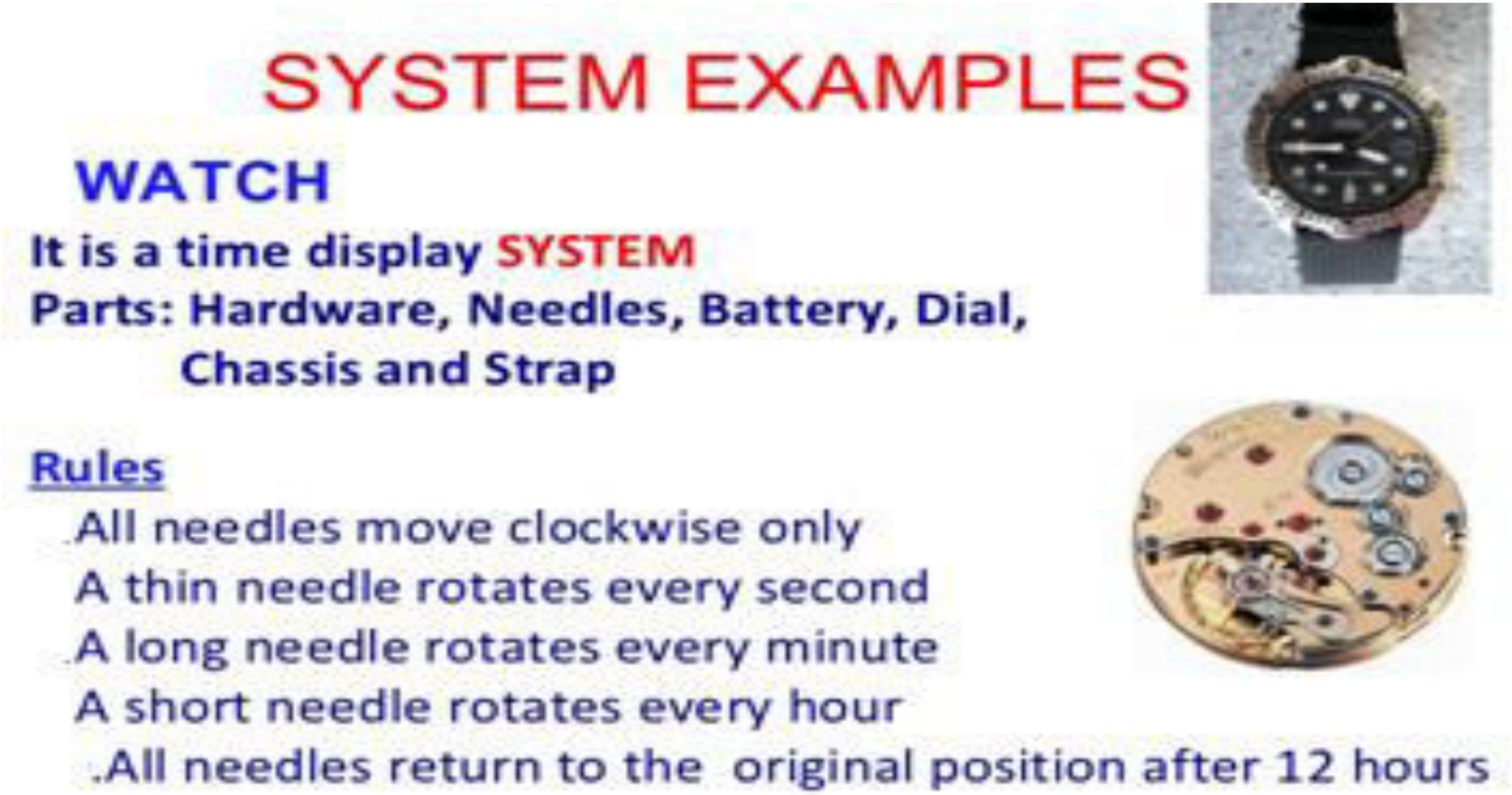
19CSE303 Embedded Systems 3-0-3-4	Internal	Components	Max Marks	Weightage	
		Quiz-1	10	7.5	
		Quiz-2	10	7.5	
		Lab Evaluation-1	15	15	
		Lab Evaluation-2	15	15	
		Periodical Exam 1	50	10	
		Periodical Exam 2	50	10	
	External				
		End Semester Exam	100	35	

Introduction

- What is a System?

A system is a way of working organizing or doing one or many tasks according to a fixed plan, program or set of rules

A system is also an arrangement in which all its units assemble and work together according to the plan or program



SYSTEM EXAMPLES

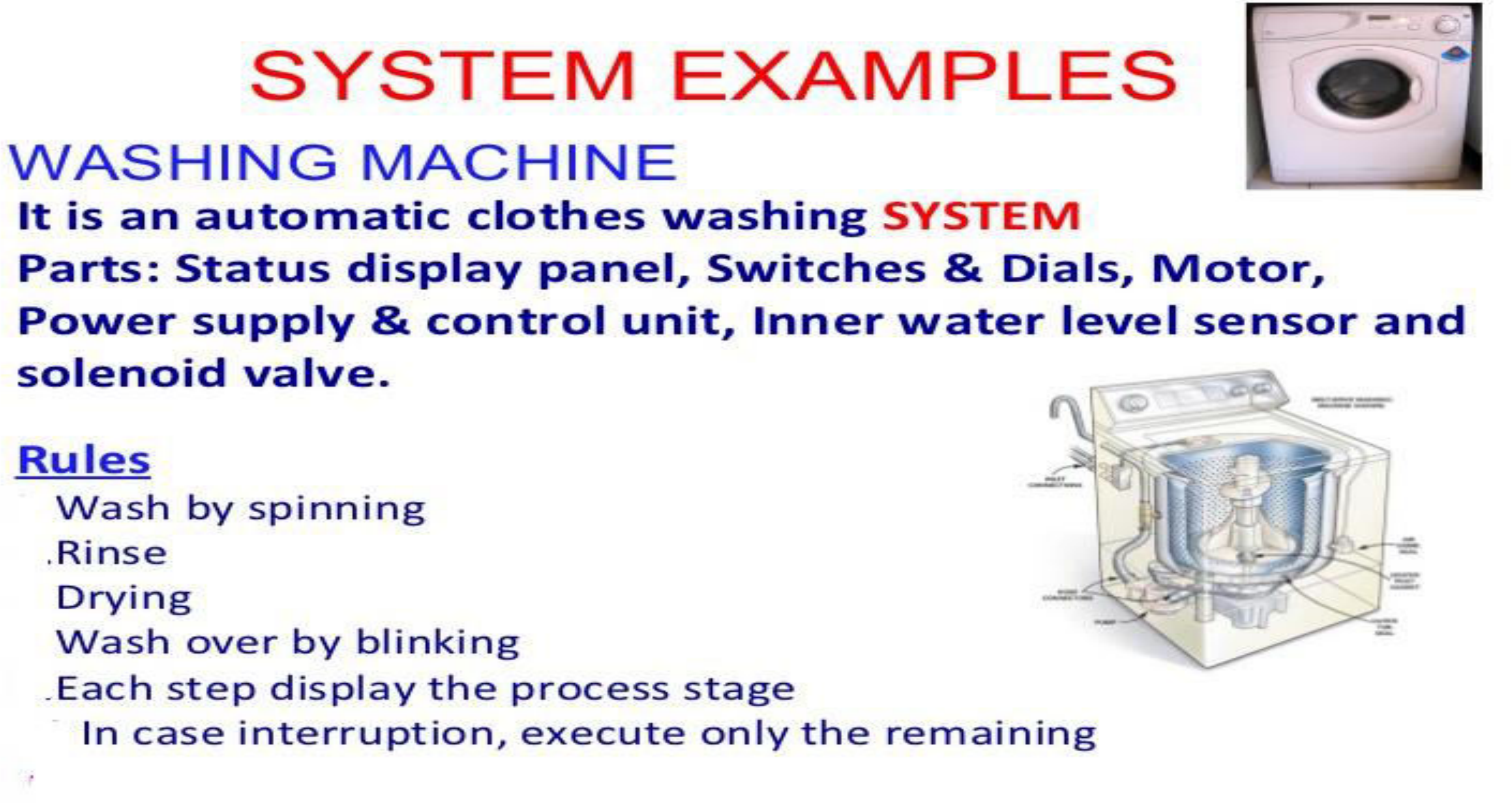
WATCH

It is a time display **SYSTEM**

Parts: Hardware, Needles, Battery, Dial,
Chassis and Strap

Rules

All needles move clockwise only
A thin needle rotates every second
A long needle rotates every minute
A short needle rotates every hour
.All needles return to the original position after 12 hours



SYSTEM EXAMPLES

WASHING MACHINE

It is an automatic clothes washing SYSTEM

Parts: Status display panel, Switches & Dials, Motor, Power supply & control unit, Inner water level sensor and solenoid valve.

Rules

Wash by spinning

.Rinse

Drying

Wash over by blinking

.Each step display the process stage

In case interruption, execute only the remaining

Computing Systems

Computing systems are everywhere

Most of us think of “desktop” computers

- PC’s
- Laptops
- Mainframes
- Servers

designed to be flexible and to meet a wide range of end-user needs.

But there’s another type of computing system

- Far more common...

Embedded Systems

- We are surrounded by Embedded Systems
  - Cell Phones
  - Automatic Washing Machines
  - Traffic Signals with Timers
  - Automobile Electronics

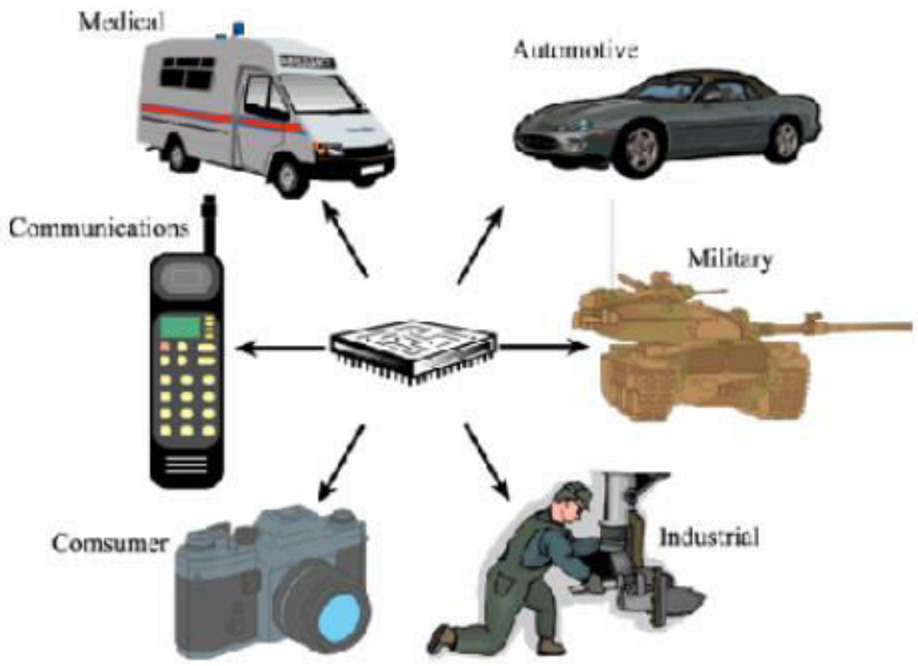
Find a System that contains no electronic system

How an electronic system improve the functionality / efficiency of that system

Over 95% of software systems are actually embedded !!!

Embedded Systems

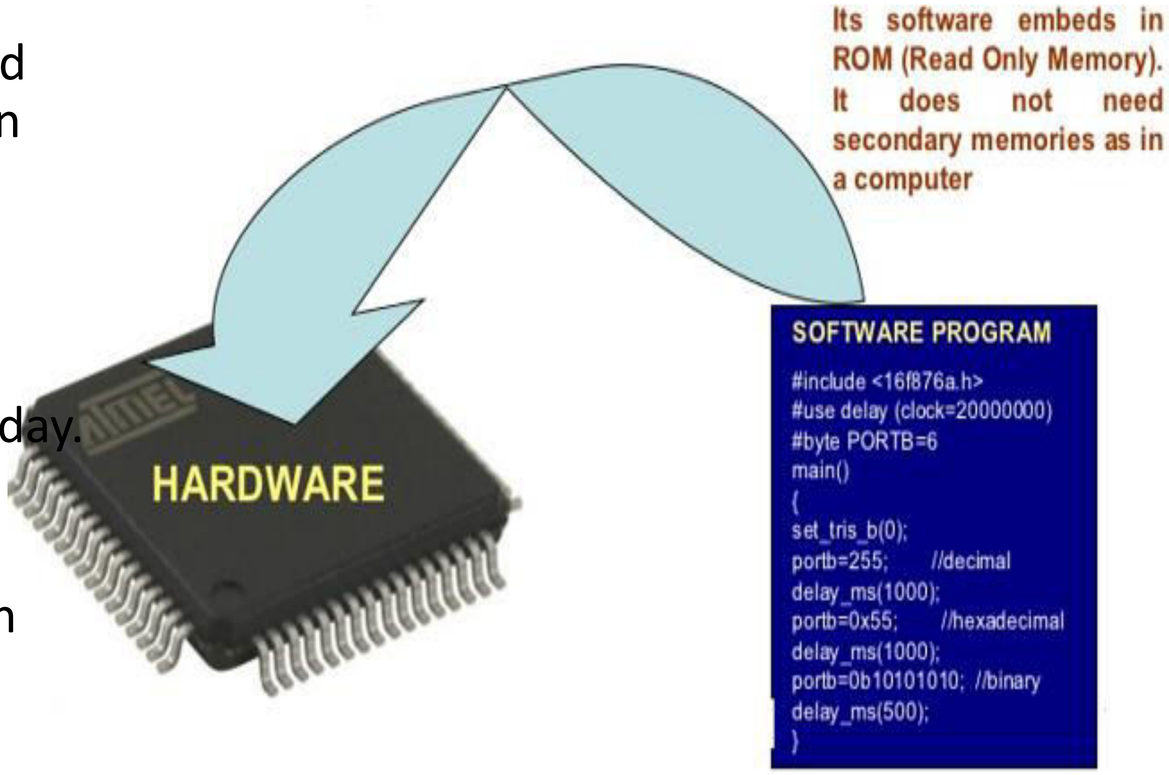
- A Special purpose computer designed to perform certain dedicated functions
- Embedded Systems are everywhere
- Hidden (computer inside)
- Dedicated purpose



Embedded Systems

- An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.
- It is *embedded* as part of a complete device often including hardware and mechanical parts.
- Embedded systems control many devices in common use today.
- **Definition:**

An Embedded System is one that has computer hardware with software embedded in it as one of its important components.



Its software embeds in ROM (Read Only Memory). It does not need secondary memories as in a computer

SOFTWARE PROGRAM

```
#include <16f876a.h>
#use delay (clock=20000000)
#byte PORTB=6
main()
{
set_tris_b(0);
portb=255;      //decimal
delay_ms(1000);
portb=0x55;     //hexadecimal
delay_ms(1000);
portb=0b10101010; //binary
delay_ms(500);
}
```

Embedded Systems - Characteristics

- Single-functioned –

  An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager.

- Tightly constrained –

  All computing systems have constraints on design metrics, but those on an embedded system can be especially tight.

  Design metrics is a measure of an implementation's features such as its cost, size, power, and performance.

  It must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.

Embedded Systems - Characteristics

- Reactive and Real time −

  Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay.

  Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors.

  It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to control of the car.

- Microprocessors based −

  It must be microprocessor or microcontroller based.

Embedded Systems - Characteristics

- Memory −

  It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer.

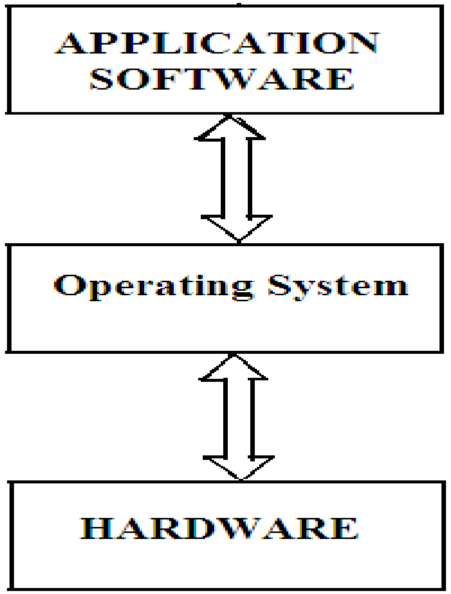
- Connected −

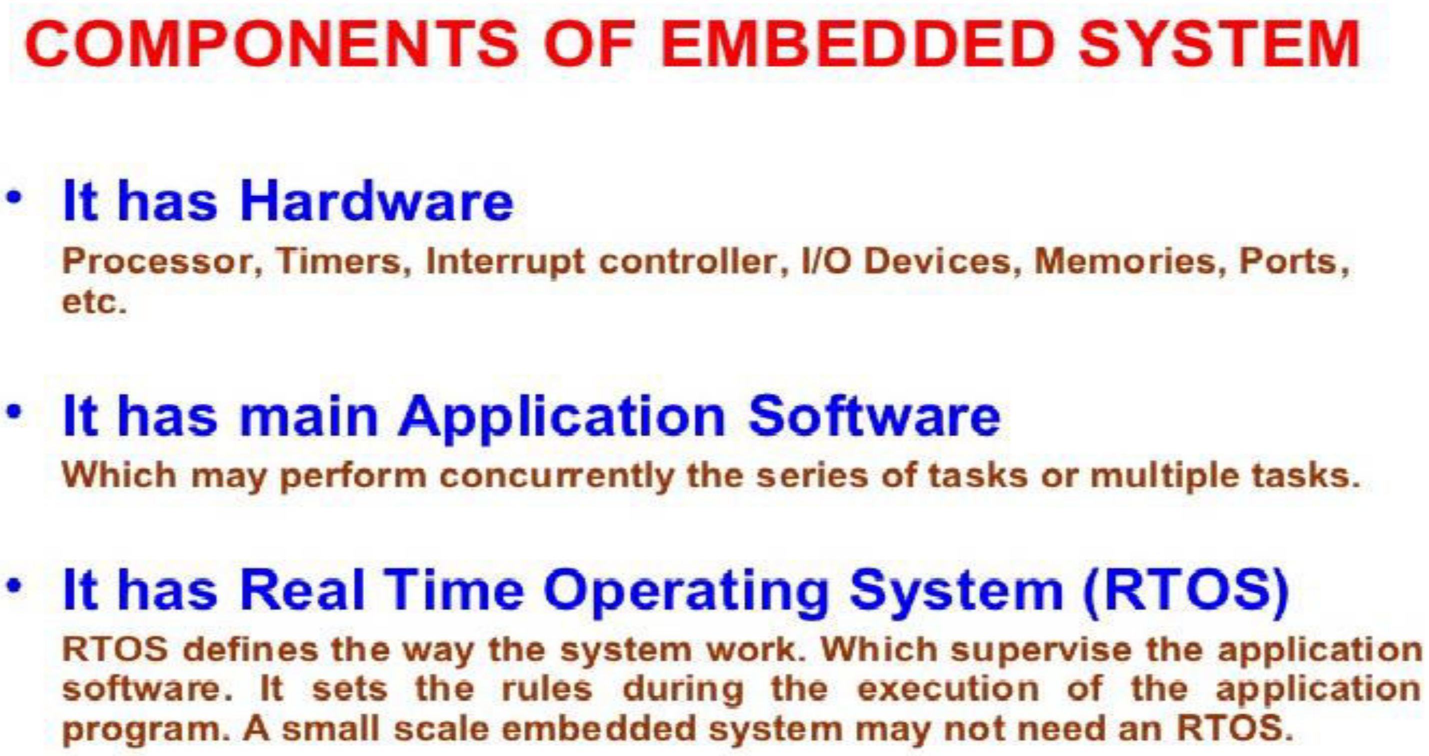
  It must have connected peripherals to connect input and output devices.

- HW-SW systems −

  Software is used for more features and flexibility. Hardware is used for performance and security.

What makes an Embedded Systems





COMPONENTS OF EMBEDDED SYSTEM

- **It has Hardware**

  Processor, Timers, Interrupt controller, I/O Devices, Memories, Ports, etc.

- **It has main Application Software**

  Which may perform concurrently the series of tasks or multiple tasks.

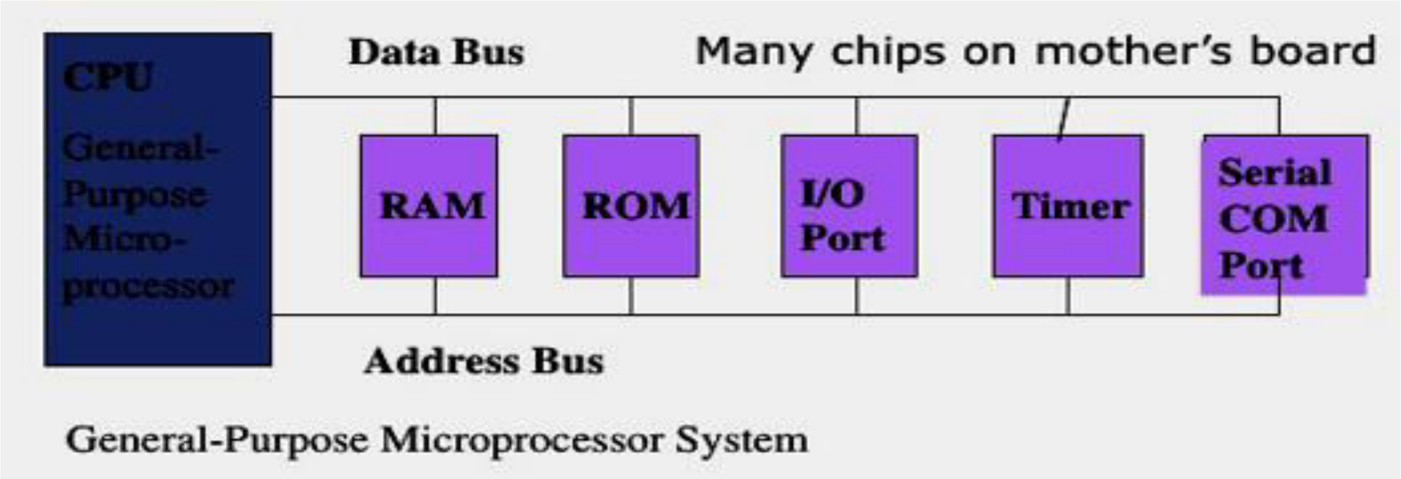
- **It has Real Time Operating System (RTOS)**

  RTOS defines the way the system work. Which supervise the application software. It sets the rules during the execution of the application program. A small scale embedded system may not need an RTOS.

Embedded Systems – Microcontrollers vs Microprocessors

Microprocessors:

- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example: Intel's x86, Motorola's 680x0

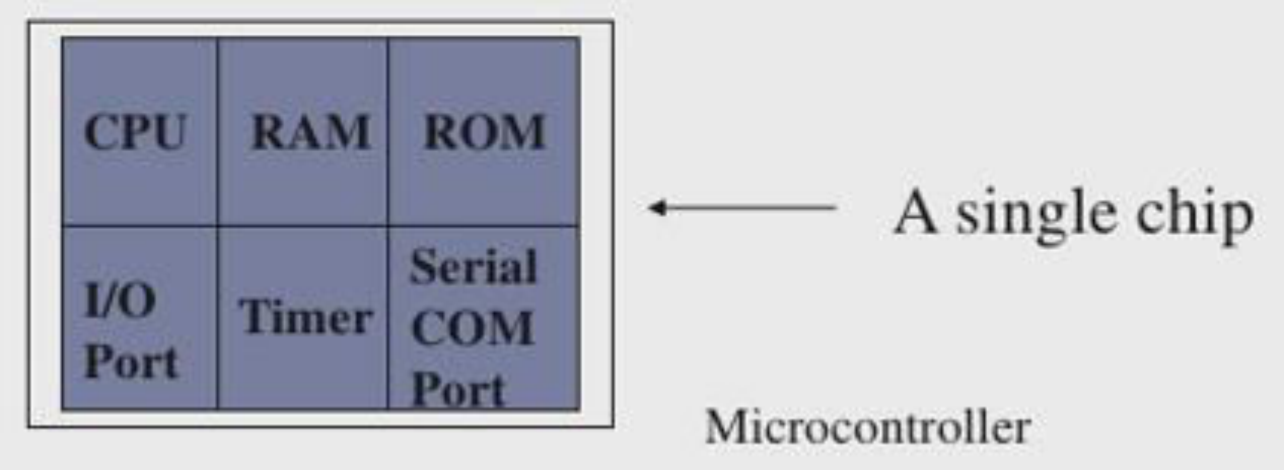


General-Purpose Microprocessor System

Embedded Systems – Microcontrollers vs Microprocessors

Microcontroller:

- A smaller computer
- On-chip RAM, ROM, I/O ports…
- Example: Motorola’s 6811, Intel’s 8051, Zilog’s Z8 and PIC 16X

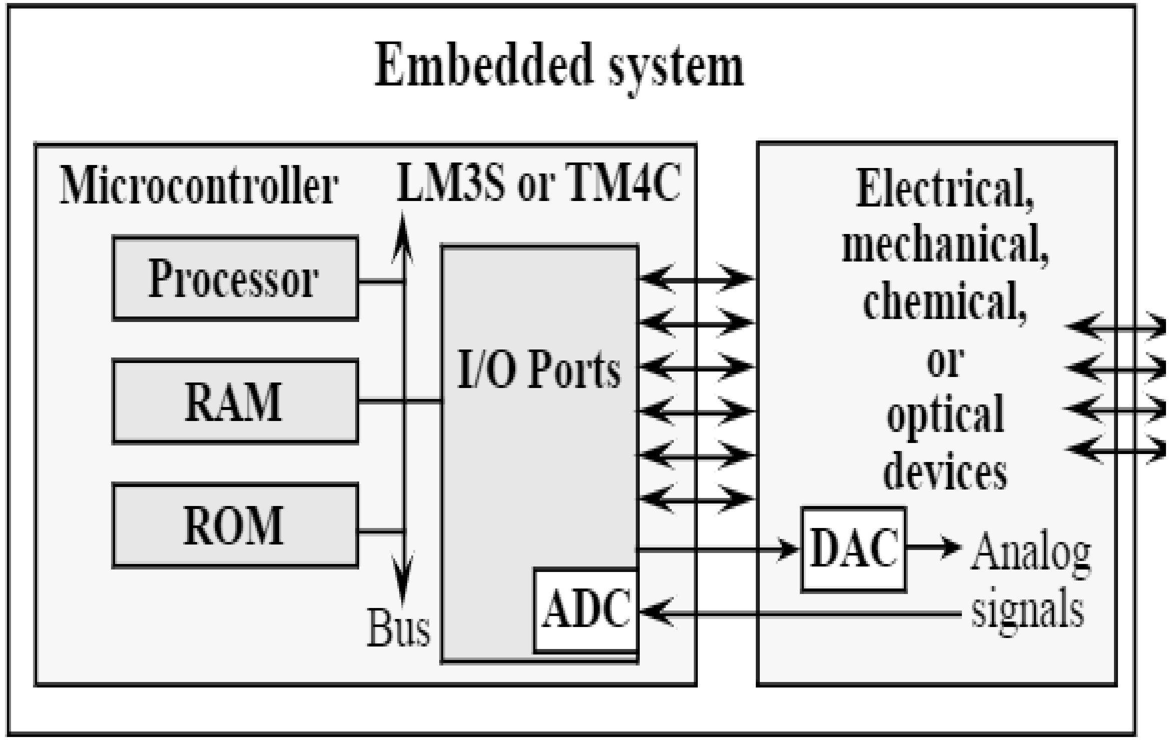


Embedded Systems – Microcontrollers vs Microprocessors

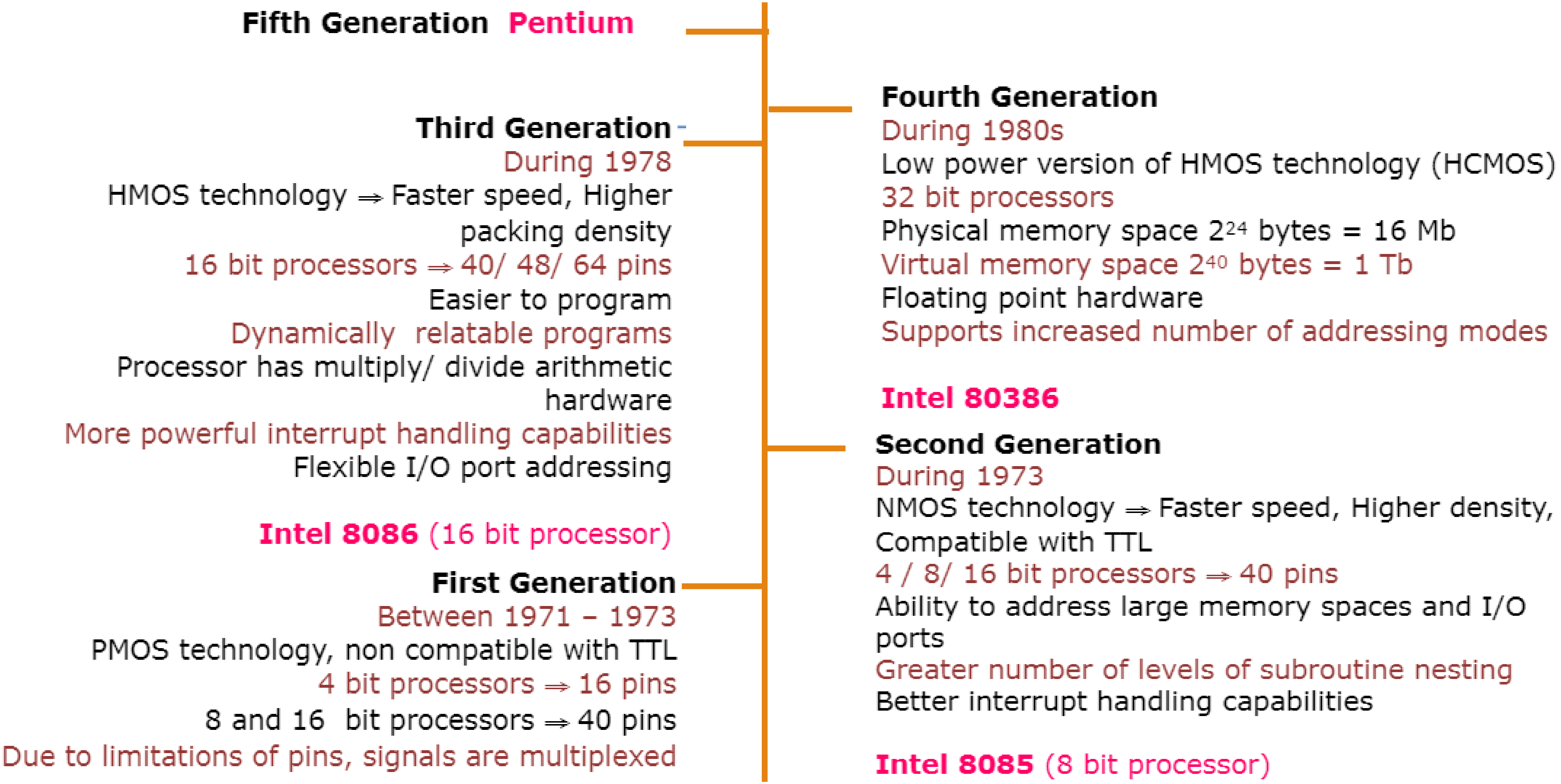
Microprocessors	Microcontrollers
Microprocessors are multitasking in nature. Can perform multiple tasks at a time. For example, on computer we can play music while writing text in text editor.	Single task oriented. For example, a washing machine is designed for washing clothes only.
RAM, ROM, I/O Ports, and Timers can be added externally and can vary in numbers.	RAM, ROM, I/O Ports, and Timers cannot be added externally. These components are to be embedded together on a chip and are fixed in
Designers can decide the number of memory or I/O ports needed.	Fixed number for memory or I/O makes a microcontroller ideal for a limited but specific task.
External support of external memory and I/O ports makes a microprocessor-based system heavier and costlier.	Microcontrollers are lightweight and cheaper than a microprocessor.
External devices require more space and their power consumption is higher.	A microcontroller-based system consumes less power and takes less space.

Embedded Systems

- Microprocessor
  - Intel: 4004, ..8080,.. x86
  - Freescale: 6800, .. 9S12,.. PowerPC
  - ARM, DEC, SPARC, MIPS, PowerPC, Natl. Semi.,...
  - Applications: Desktop PC's, Laptops, notepads etc.
- Microcontroller
  - Processor+Memory+I/O Ports (Interfaces)
  - Applications: Keyboards, mouse, pendrive, mobiles etc



Evolution of Microprocessors



First Generation

Between 1971 – 1973

PMOS technology, non compatible with TTL

4 bit processors ⇒ 16 pins

8 and 16 bit processors ⇒ 40 pins

Due to limitations of pins, signals are multiplexed

Second Generation

During 1973

NMOS technology ⇒ Faster speed, Higher density, Compatible with TTL

4 / 8/ 16 bit processors ⇒ 40 pins

Ability to address large memory spaces and I/O ports

Greater number of levels of subroutine nesting

Better interrupt handling capabilities

Intel 8085 (8 bit processor)

Third Generation

During 1978

HMOS technology ⇒ Faster speed, Higher packing density

16 bit processors ⇒ 40/ 48/ 64 pins

Easier to program

Dynamically relatable programs

Processor has multiply/ divide arithmetic hardware

More powerful interrupt handling capabilities

Flexible I/O port addressing

Intel 8086 (16 bit processor)

Fourth Generation

During 1980s

Low power version of HMOS technology (HCMOS)

32 bit processors

Physical memory space 2^{24} bytes = 16 Mb

Virtual memory space 2^{40} bytes = 1 Tb

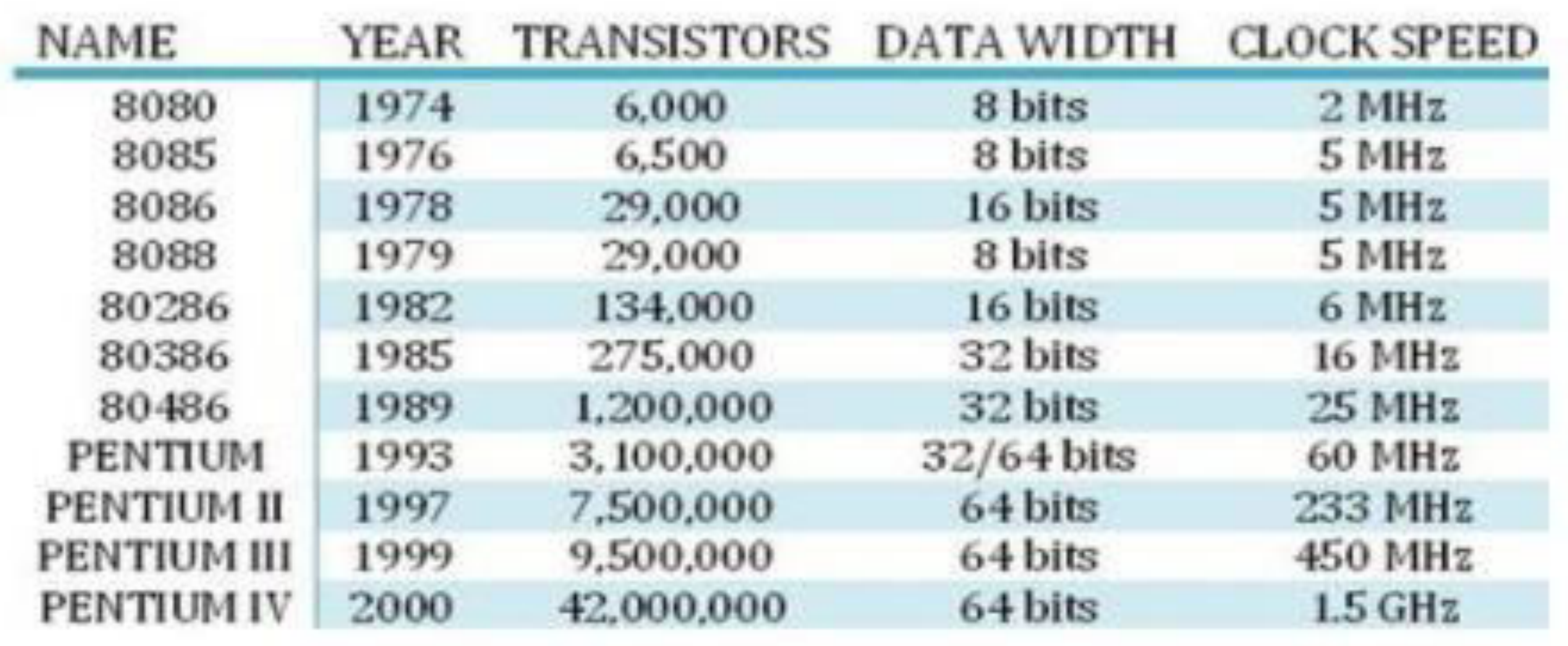
Floating point hardware

Supports increased number of addressing modes

Intel 80386

Fifth Generation **Pentium**

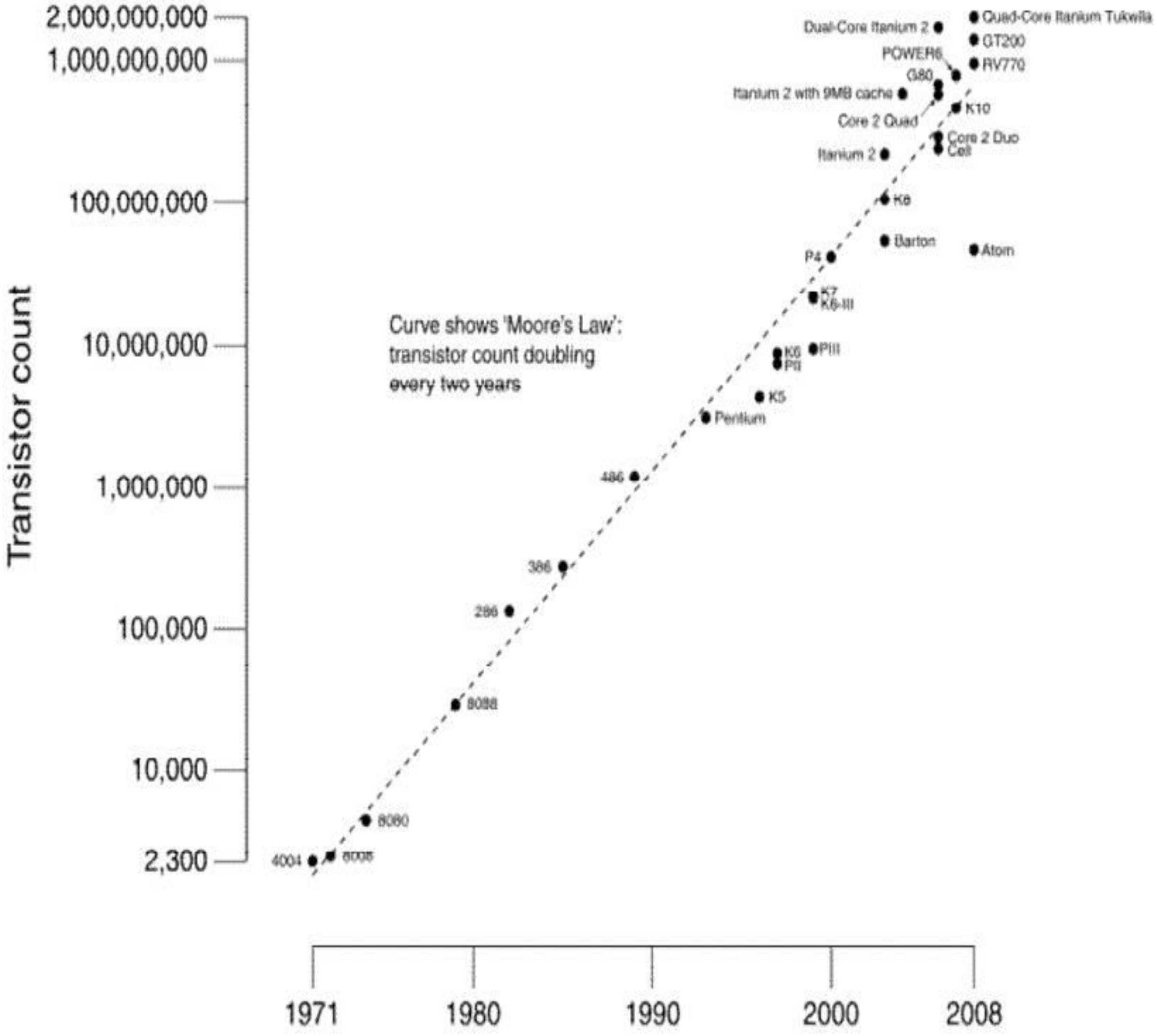
Evolution of Microprocessors



NAME	YEAR	TRANSISTORS	DATA WIDTH	CLOCK SPEED
8080	1974	6,000	8 bits	2 MHz
8085	1976	6,500	8 bits	5 MHz
8086	1978	29,000	16 bits	5 MHz
8088	1979	29,000	8 bits	5 MHz
80286	1982	134,000	16 bits	6 MHz
80386	1985	275,000	32 bits	16 MHz
80486	1989	1,200,000	32 bits	25 MHz
PENTIUM	1993	3,100,000	32/64 bits	60 MHz
PENTIUM II	1997	7,500,000	64 bits	233 MHz
PENTIUM III	1999	9,500,000	64 bits	450 MHz
PENTIUM IV	2000	42,000,000	64 bits	1.5 GHz

Moore’s Law:

- ➢ Moore's Law refers to Moore's perception that the number of transistors on a microchip doubles every two years, though the cost of computers is halved.
- ➢ Moore's Law states that we can expect the speed and capability of our computers to increase every couple of years, and we will pay less for them.



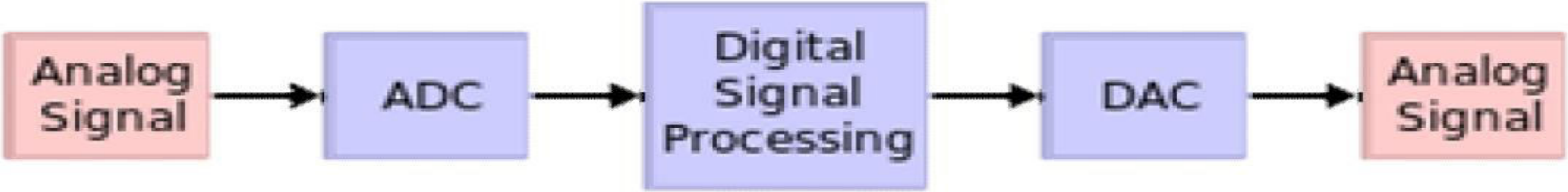
Digital Signal Processing

Definition:

Digital signal processing (DSP) is the process of analyzing and modifying a signal to optimize or improve its efficiency or performance.

It involves applying various mathematical and computational algorithms to analog and digital signals to produce a signal that's of higher quality than the original signal.

Digital Signal Processing



- Architecture optimized for signal processing applications
  - Large number of mathematical operations on a series of data samples
- Hardware implementation of Multiply/Accumulate function
  - Critical for FFT (Fast Fourier Transform) type applications
- The Texas Instruments TMS 5100

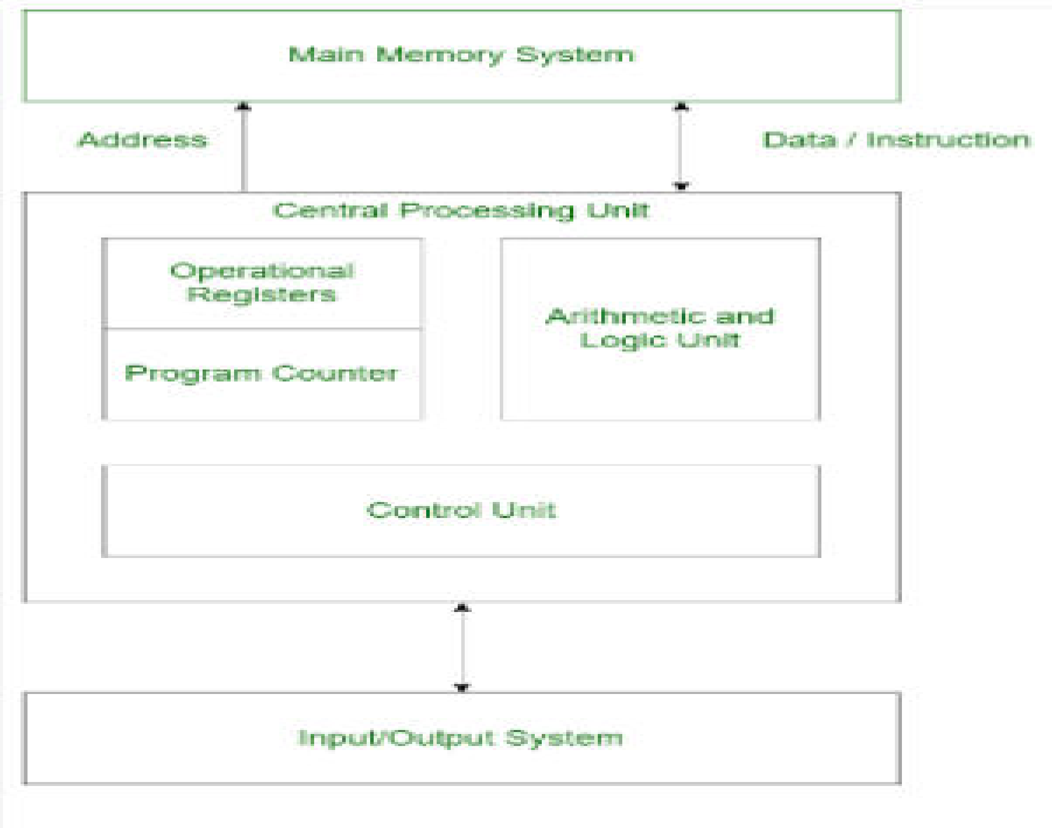
Architecture vs. Organization

Computer Architecture	Computer Organization
Computer Architecture is concerned with the way hardware components are connected together to form a computer system.	Computer Organization is concerned with the structure and behaviour of a computer system as seen by the user.
It acts as the interface between hardware and software.	It deals with the components of a connection in a system.
Computer Architecture helps us to understand the functionalities of a system.	Computer Organization tells us how exactly all the units in the system are arranged and interconnected.
A programmer can view architecture in terms of instructions, addressing modes and registers.	Organization expresses the realization of architecture.
While designing a computer system architecture is considered first.	An organization is done on the basis of architecture.
Computer Architecture deals with high-level design issues.	Computer Organization deals with low-level design issues.
Architecture involves Logic (Instruction sets, Addressing modes, Data types, Cache optimization)	Organization involves Physical Components (Circuit design, Adders, Signals, Peripherals)

Von-Neumann Architecture

- Von Neumann Architecture is a digital computer architecture whose design is based on the concept of stored program computers where program data and instruction data are stored in the same memory.

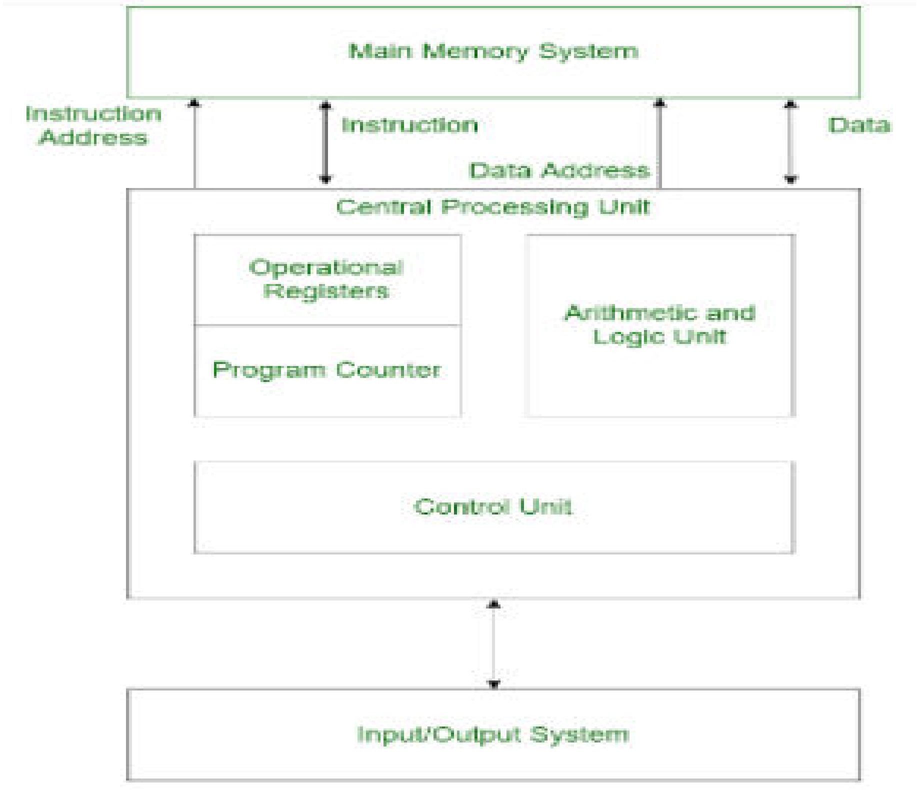
- This architecture was designed by the famous mathematician and physicist John Von Neumann in 1945.



Von Neumann Architecture

Harvard Architecture

- Harvard Architecture is the digital computer architecture whose design is based on the concept where there are separate storage and separate buses (signal path) for instruction and data.
- It was basically developed to overcome the bottleneck of Von Neumann Architecture



Harvard Architecture

Von-Neumann vs Harvard Architecture

Von-Neumann Architecture	Harvard Architecture
Single memory to be shared by both code and data.	Separate memories for code and data.
Processor needs to fetch code in a separate clock cycle and data in another	Single clock cycle is sufficient, as separate buses
Higher speed, thus less time consuming.	Slower in speed, thus more time-consuming.
Simple in design	Complex in design.

RISC vs CISC

RISC:

- Reduced Instruction Set Architecture
- The main idea behind is to make hardware simpler by using an instruction set composed
- Just like a load command will load data, store command will store the data.
- Mostly all have the same format.
- Reduce the number of memory accesses required by increasing the number of registers
- Reduce the number of addressing modes
- Allow pipelining of instructions
- To increase the CPU performance: Reduce the cycles per instruction at the cost of the number of instructions per program.

RISC vs CISC

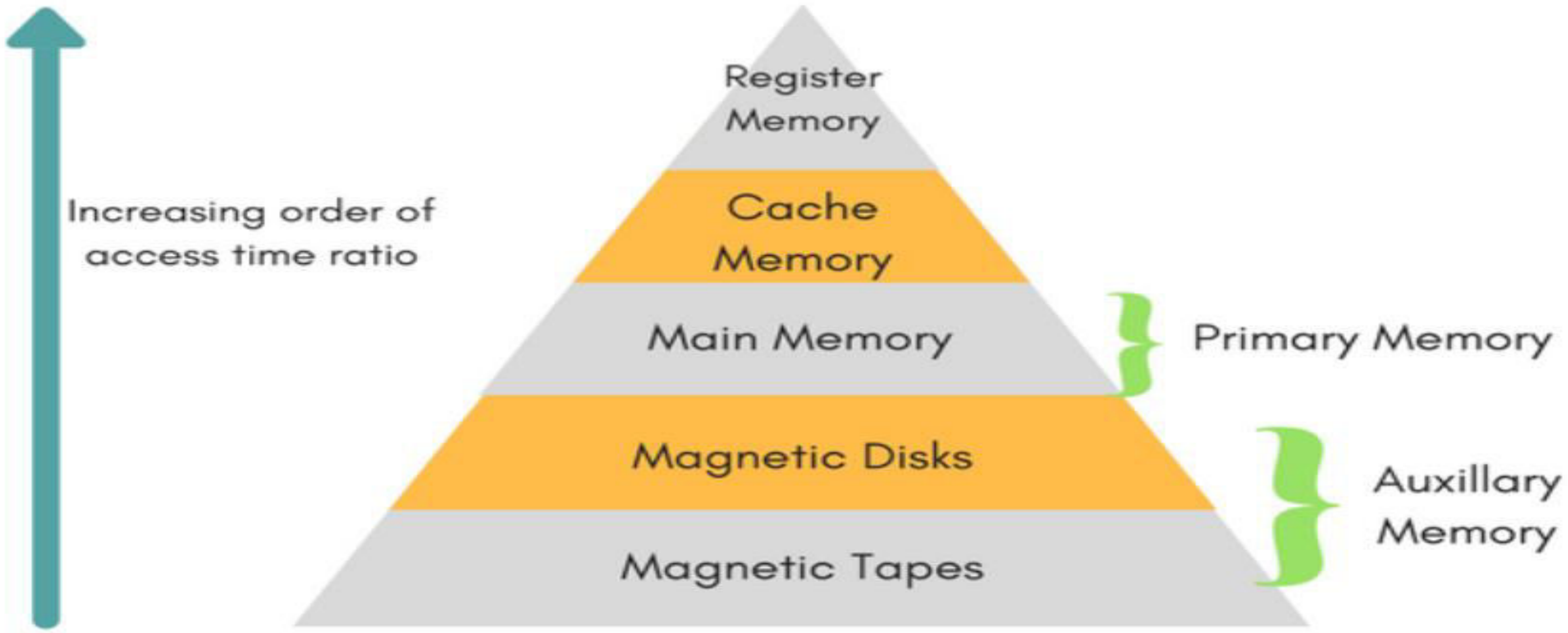
CISC:

- Complex Instruction Set Architecture
- The main idea is that a single instruction will do more operations
- Just like a multiplication command which does; loading data, evaluating, and storing it, hence it's complex.
- Number of instructions are reduced by having *multiple operations* within a single instruction
- In turn making instruction length variable and fetch-decode-execute time unpredictable – making it more complex
- Provide more addressing modes
- Less number of general-purpose registers as operation get performed in memory itself.
- To increase the CPU performance: Minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.

RISC vs CISC

CISC	RISC
Larger set of instructions. Easy to Program.	Smaller set of Instructions. Difficult to program.
Simpler design of compiler, considering larger set of instructions.	Complex design of compiler.
Many addressing modes causing complex instruction formats.	Few addressing modes, fix instruction format.
Instruction length is variable	Instruction length varies.
Higher clock cycles per second.	Low clock cycle per second.
Emphasis is on hardware.	Emphasis is on software.
Control unit implements large instruction set using micro-program unit.	Each instruction is to be executed by hardware.
Slower execution, as instructions are to be read from memory and decoded by the decoder unit.	Faster execution, as each instruction is to be executed by hardware.
Pipelining is not possible.	Pipelining of instructions is possible,

Memory Organization



Memory Organization

- Byte organized memory:

  Each address on the address bus points to a memory location where a byte (8 bit) is stored.

- Word organized memory:

  Each address on the address bus points to a memory location where a word (multiple of 8 bit) is stored.

Memory Organization

How data is stored in memory?

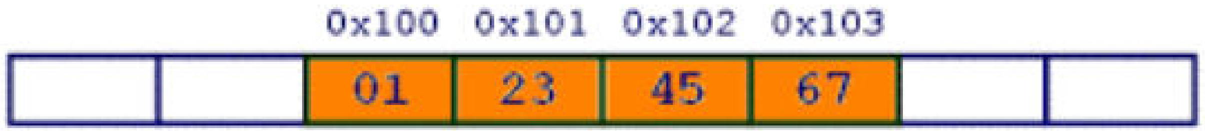
- Little Endian –

  LSB to Lower memory address

- Big Endian –

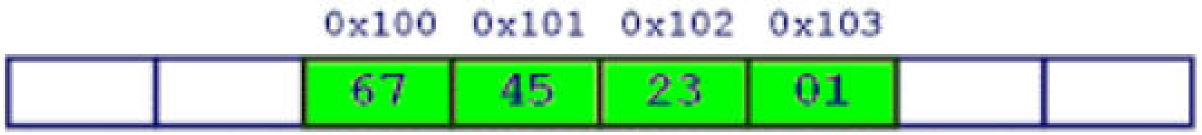
  LSB to Higher memory address

- 0x01234567 will be stored as



		0x100	0x101	0x102	0x103		
		01	23	45	67		

Big Endian



		0x100	0x101	0x102	0x103		
		67	45	23	01		

Little Endian

Thank You

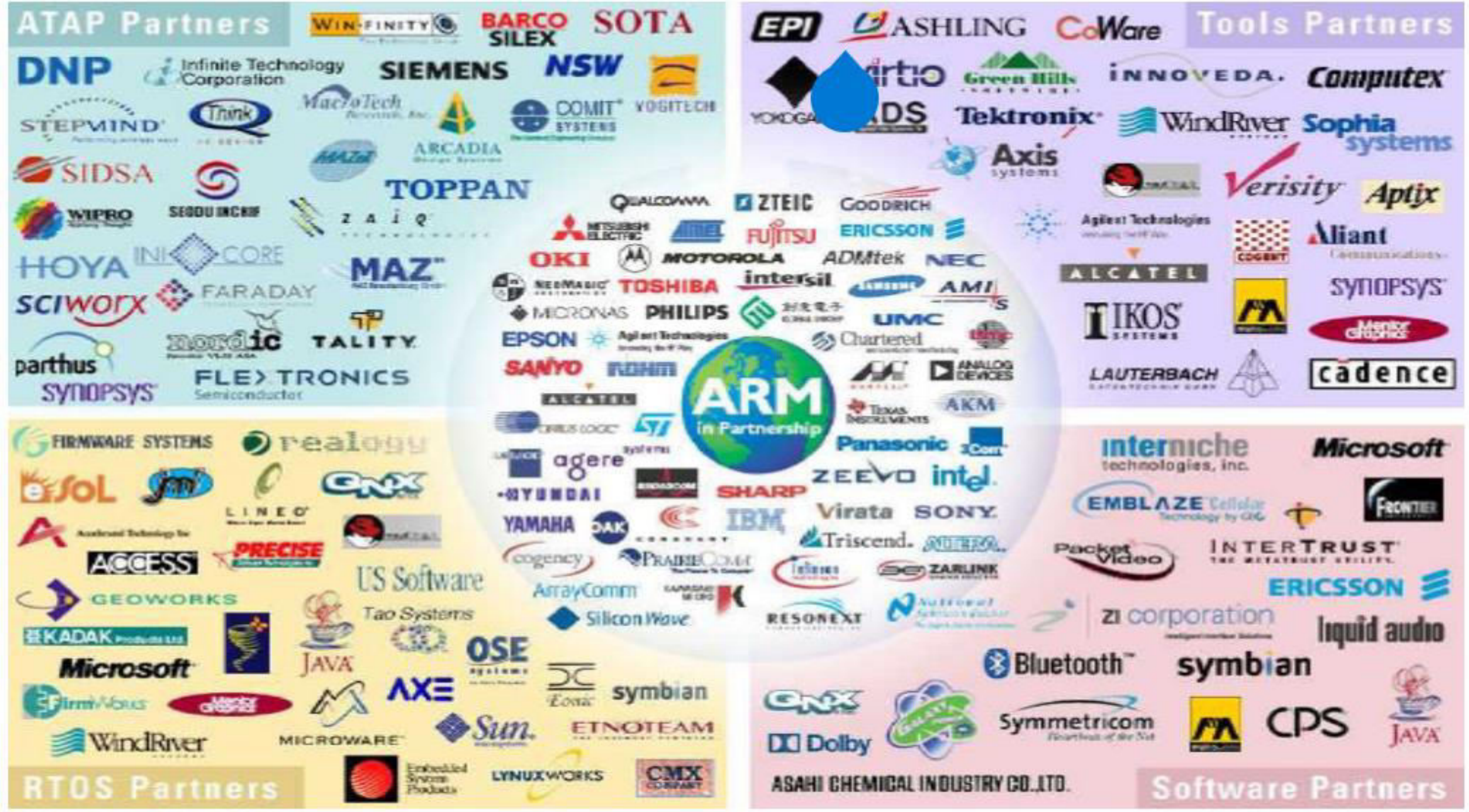
L	T	P	C
3	0	3	4

19CSE303 – Embedded Systems

ARM (History)

- ARM (Acorn RISC Machine) started as a new, powerful, CPU design for the replacement of the 8-bit 6502 in Acorn Computers (Cambridge, UK, 1985)
- First models had only a 26-bit program counter, limiting the memory space to 64 MB (not too much by today standards, but a lot at that time).
- 1990 spin-off: ARM renamed Advanced RISC Machines
- ARM now focuses on Embedded CPU cores
  - IP(Intellectual Property) licensing: Almost every silicon manufacturer sells some microcontroller with an ARM core. Some even compete with their own designs.
  - Processing power with low current consumption
    - Good MIPS/Watt figure
    - Ideal for portable devices
- Compact memories: 16-bit opcodes (Thumb)

ARM (Partnership Model)



ARM (Powered Products)

ARM Architecture

- Based on RISC architecture with enhancements to meet requirements of Embedded applications
- A large uniform register file
- Load Store Architecture
- Uniform and Fixed length instruction
- 32-bit processor
- Instructions are of 32-bit long
- Good speed / Power consumption ratio
- High Code density

Enhancement to Basic Architecture

- Control over ALU and shifter for every data processing operations to maximize their usage.
- Auto-increment and Auto-decrement addressing modes to optimize program loops
- Load –Store multiply instructions to maximize data throughput
- Conditional execution of instruction to maximize execution throughput

ARM Architecture Versions

- Version 1
  - The first ARM processor, developed at Acorn Computers Limited
  - 1983-1985
- Version 2
  - Sold in volume in the Acorn Archimedes and A3000 products
  - 26-bit addressing, including 32-bit result multiply and coprocessor
- Version 2a
  - Coprocessor 15 as the system control coprocessor to manage cache and the atomic load store (SWP) instruction

ARM Architecture Versions

- Version 3
  - First ARM processor designed by ARM Limited (1990)
  - ARM6 (macro cell)
  - ARM60 (stand-alone processor)
  - ARM600 (an integrated CPU with on-chip cache, MMU, write buffer)
  - ARM610 (used in Apple Newton)
  - 32-bit addressing, separate CPSR and SPSRs
  - And the undefined and abort modes to allow coprocessor emulation and virtual memory support in supervisor mode
- Version 3M
  - Introduce the signed and unsigned multiply and multiply accumulate
  - instructions that generate the full 64-bit result

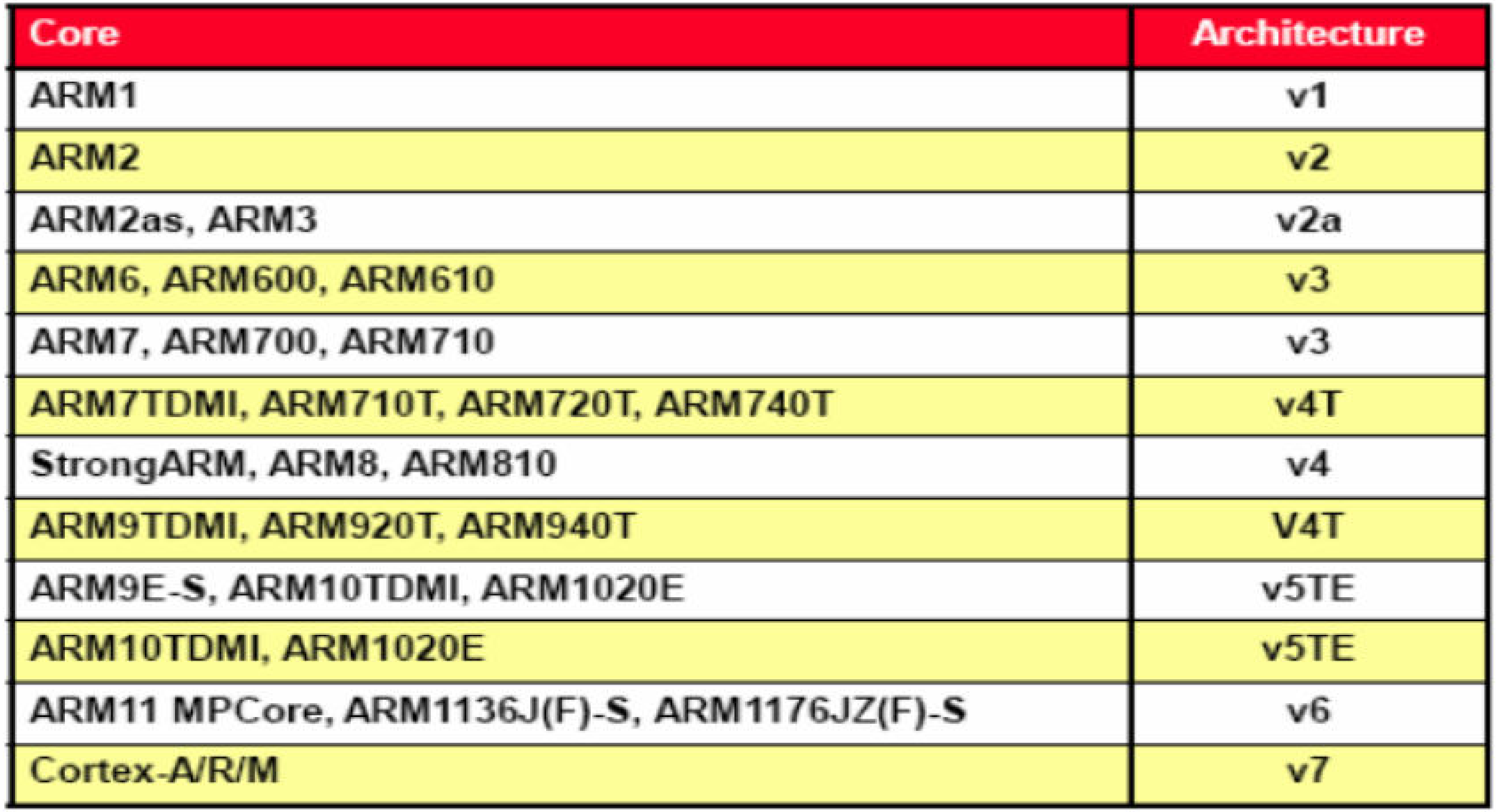
ARM Architecture Versions

- Version 4
  - Add the signed, unsigned half-word and signed byte load and store instructions
  - Reserve some of SWI space for architecturally defined operation
  - System mode is introduced
- Version 4T
  - 16-bit Thumb compressed form of the instruction set is introduced
- Version 5T
  - Introduced recently, a superset of version 4T adding the BLX, CLZ and
  - BRK instructions
- Version 5TE
  - Add the signal processing instruction set extension

ARM Architecture Versions

- Version 6
  - Media processing extensions (SIMD)
    - 2x faster MPEG4 encode/decode
    - 2x faster audio DSP
  - Improved cache architecture
  - Physically addressed caches
  - Reduction in cache flush/refill
  - Reduced overhead in context switches
  - Improved exception and interrupt handling
  - Important for improving performance in real-time tasks
  - Unaligned and mixed-endian data support
  - Simpler data sharing, application porting and saves memory

ARM Cores



Core	Architecture
ARM1	v1
ARM2	v2
ARM2as, ARM3	v2a
ARM6, ARM600, ARM610	v3
ARM7, ARM700, ARM710	v3
ARM7TDMI, ARM710T, ARM720T, ARM740T	v4T
StrongARM, ARM8, ARM810	v4
ARM9TDMI, ARM920T, ARM940T	V4T
ARM9E-S, ARM10TDMI, ARM1020E	v5TE
ARM10TDMI, ARM1020E	v5TE
ARM11 MPCore, ARM1136J(F)-S, ARM1176JZ(F)-S	v6
Cortex-A/R/M	v7

ARM

- ➢ Advanced RISC Machine / Acorn RISC Machine
- ➢ A 32-bit processor core
- ➢ On mid-1980’s replacement for 6502 used in BBC Micro.
- ➢ A Load-Store architecture
- ➢ Fixed-length 32-bit instructions (exceptions!)
- ➢ 3-address instruction formats
- ➢ Condition execution of ALL instructions

- ➢ Load-Store multiple registers in one instruction
- ➢ Most data processing instructions are single-cycle
- ➢ A single cycle n-bit shift with ALU operation
- ➢ But many other instructions take multiple clock cycles
- ➢ By default, Little-endian but can be configured for Big-endian
- ➢ In mid-90s used in Apple’s PDA Newton

Data Size and Instruction Sets

- The ARM is a 32-bit architecture.
- When used in relation to the ARM:
  - Byte means 8 bits
  - Half word means 16 bits (two bytes)
  - Word means 32 bits (four bytes)
- Most ARM's implement two instruction sets
  - 32-bit ARM Instruction Set
  - 16-bit Thumb Instruction Set
- Jazelle cores can also execute Java bytecode

ARM Processor Core

- Current low-end ARM core for applications like digital mobile phones
- TDMI

  T: Thumb, 16-bit instruction set

  D: on-chip Debug support, enabling the processor to halt in response to a debug request

  M: enhanced Multiplier, yield a full 64-bit result, high performance

  I: Embedded (in-circuit emulator) ICE hardware
- Von Neumann architecture
- 3-stage pipeline

Data Path - Overview

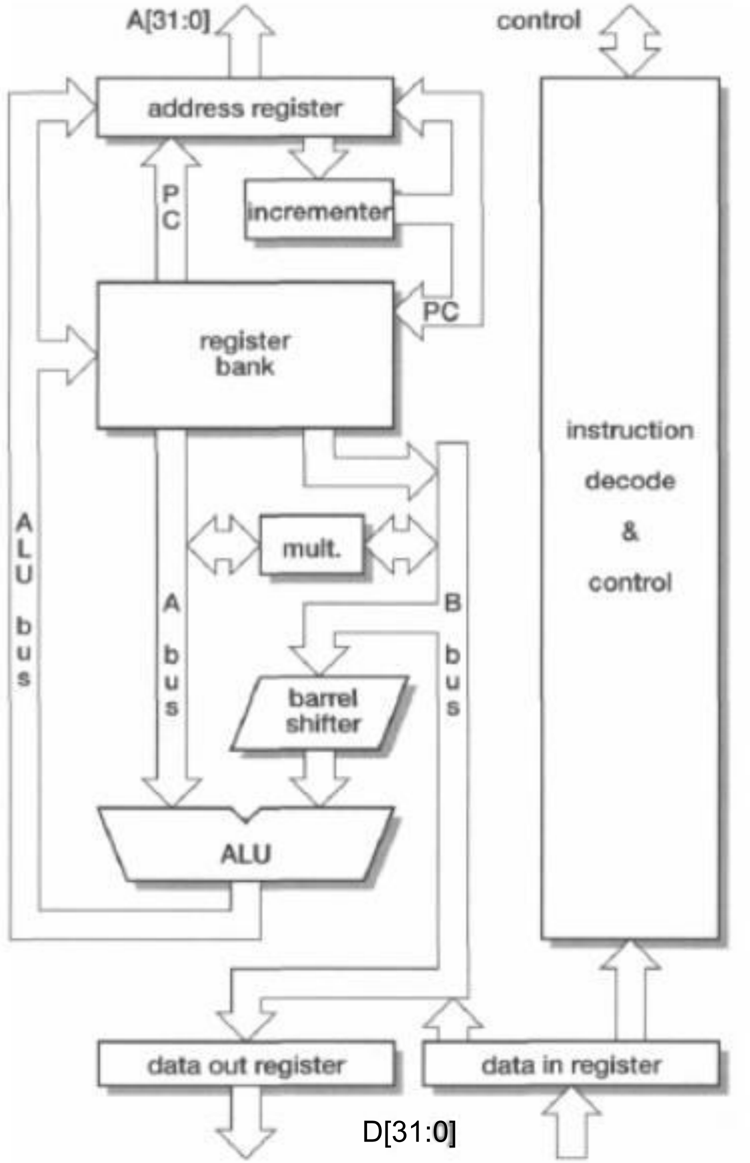
- A collection of functional units such as **arithmetic logic units or multipliers that perform data processing operations, registers, and buses**. Along with the control unit it composes the central processing unit (CPU).
- Principle: Memory will be the limiting factor
- Memory access will always take a clock cycle
- Each instruction takes exactly the number of clock cycles defined by the number of memory accesses it must take
- Datapath design with sufficient resource to allow these instructions to complete in two or one clock cycle

Data Path - Overview

- Data Items are placed in register file
- No data processing instruction directly manipulate data memory
- Instruction typically use two source registers and single result or destination register
- A Barrel shifter on data path can preprocess data before it enter ALU
- Increment/decrement logic can update register content for sequential access independent of ALU

Basic ARM Organization

- Register file –
  - 2 read ports, 1 write port +
    1 read, 1 write port reserved for r15 (pc)
- Barrel shifter – shift or rotate one operand for any number of bits
- ALU – performs the arithmetic and logic functions required
- Memory address register + incrementer
- Memory data registers
- Instruction decoder and associated control logic



ARM's Programmer Model

- ARM is a flexible programmer's designed architecture with different applications
- A processor's instruction set defines the operations that the programmer can use to change the state of the system incorporating the processor
- This state usually comprises the values of the data items in the processor's visible registers and the systems memory
- Each instruction can be viewed as performing a defined transformation from the state before the instruction is executed to the state after it has completed

ARM - Registers

- ARM has 37 registers, each 32 bit long
  - However, due to the applied conditions you can only use sixteen at any one time…
  - 1 dedicated program counter (PC)
  - 1 dedicated current program status register (CPSR)
  - 5 dedicated saved program status registers (SPSR)
  - 30 general purpose registers

ARM - Registers

- 16 register only visible registers
  - **Register 0 to register 7** are general purpose registers and can be used for **any** purpose.
  - **Register 8 to register 12** are general purpose registers, but they have shadow registers which come into use when you switch to FIQ mode
  - **Register 13** is typically the OS stack pointer, but can be used as a general purpose register. This is an operating system issue, not a processor issue, so if you don't use the stack you can corrupt this freely within your own code as long as you store it afterwards. Each of the processor modes shadow this register.

ARM - Registers

- **Register 14** is dedicated to holding the address of the return point to make writing subroutines easier. When you branch with link (BL) the return address is stored in R14

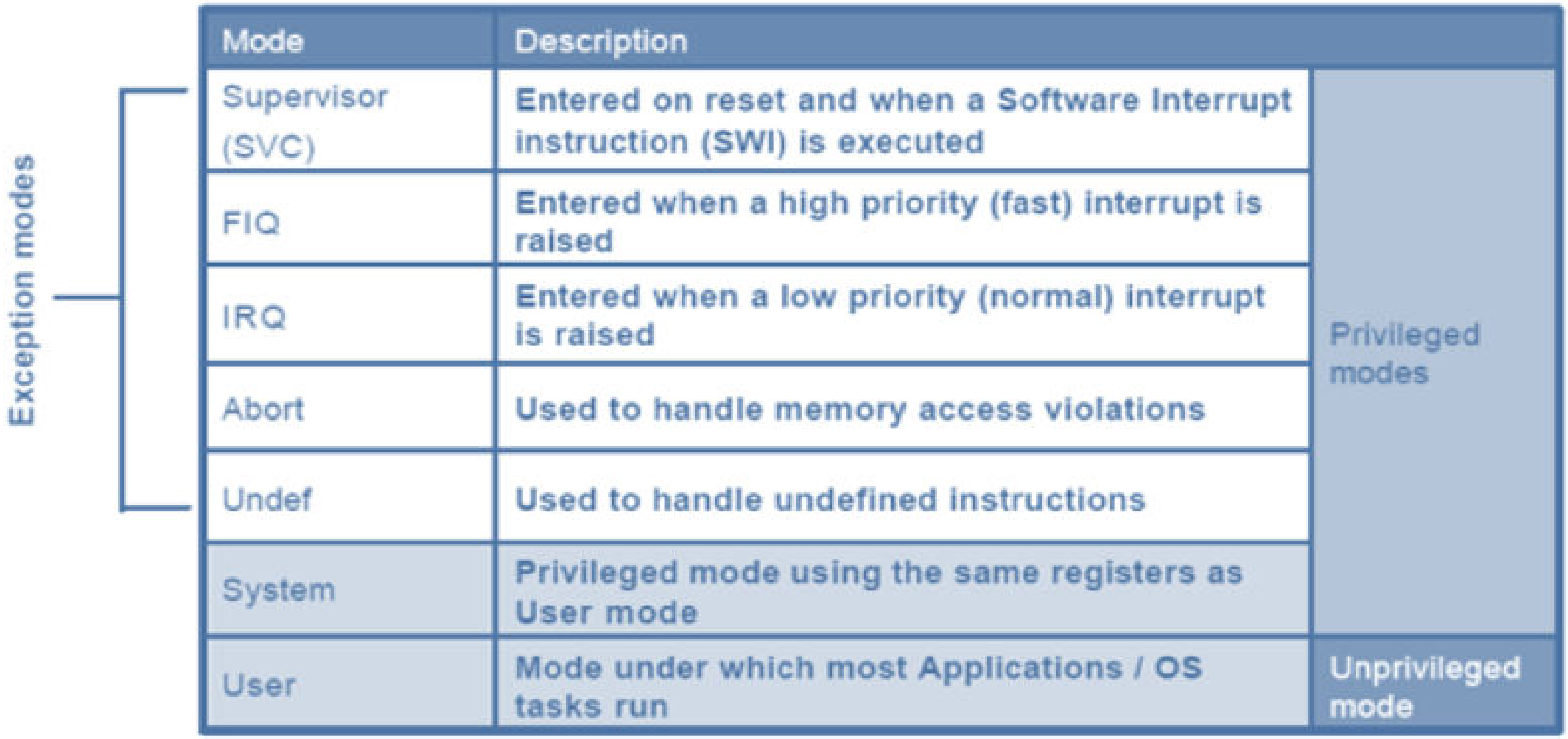
  - Likewise when the program is first run, the exit address is stored in R14. All instances of R14 must be preserved in other registers (not really efficient) or in a stack. This register is shadowed across all of the processor modes. This register can be used as a general purpose register once the link address has been preserved.

- **Register 15** is the program counter

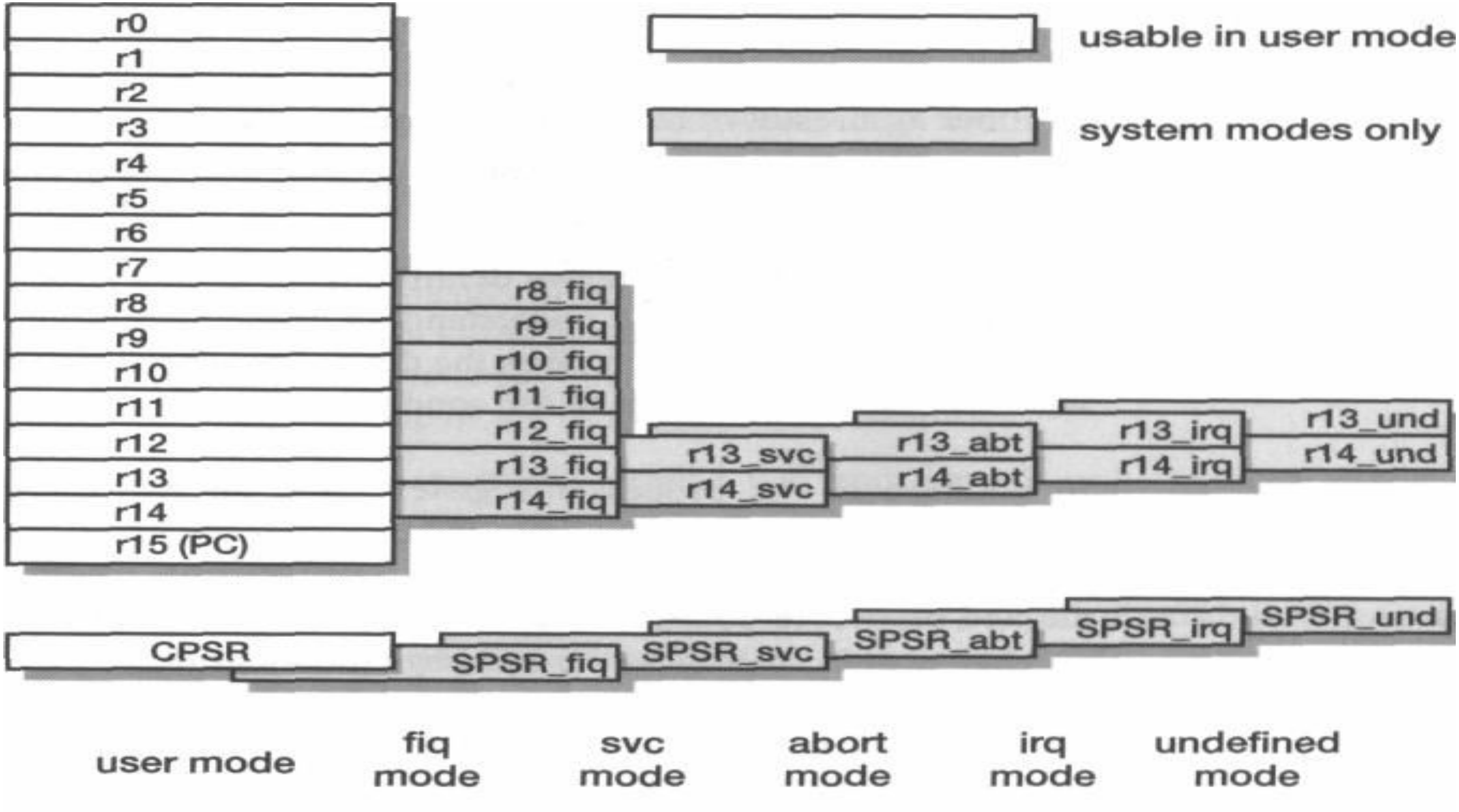
- 16 registers are user mode registers

Processor Modes

- Most ARM cores have 7 basic operating modes
  - Each mode has access to its own stack space and a different subset of registers
  - Some operations can only be carried out in a privileged mode



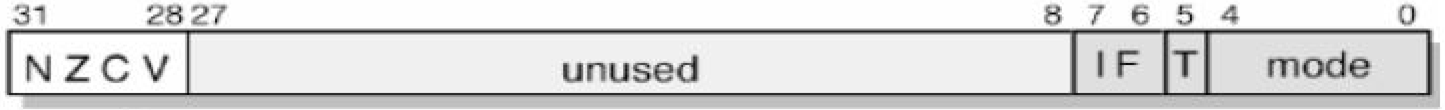
	Mode	Description	
Exception modes	Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a low priority (normal) interrupt is raised	
	Abort	Used to handle memory access violations	
	Undef	Used to handle undefined instructions	
	System	Privileged mode using the same registers as User mode	
	User	Mode under which most Applications / OS tasks run	Unprivileged mode



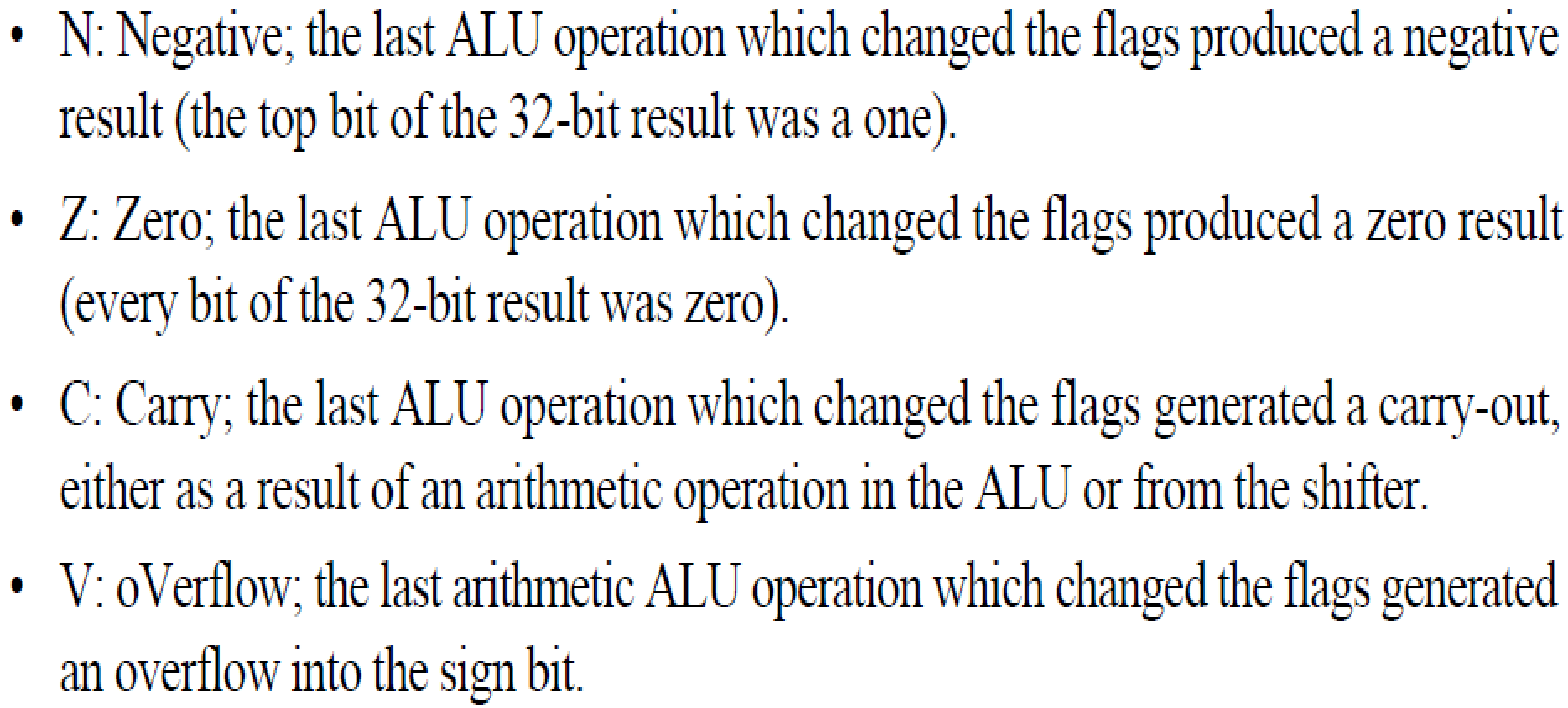
ARM’s Visible Registers

Program Status register

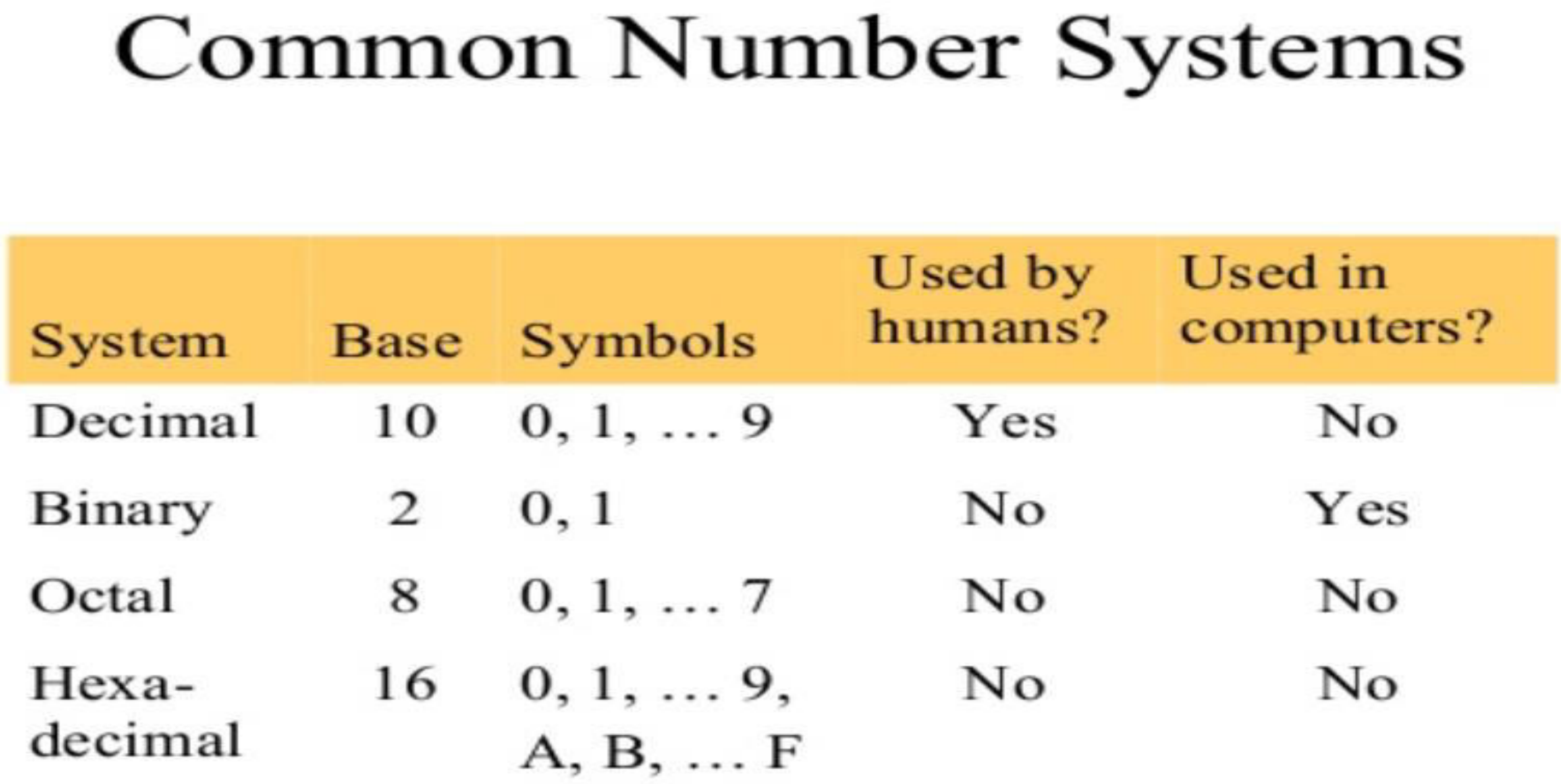
- **Current Program Status Register (CPSR):**
  - Monitors and controls internal operations
- **N** – Negative / sign (1 = negative & 0 = positive)
- **Z** – Zero (1 = zero & 0 = non zero)
- **C** – Carry (1 = carry occurred & 0 = no carry) – for unsigned operations
- **V** – Overflow (1 = overflow occurred & 0 = no overflow) – for signed operations
- **Mode**: Indicates which mode the processor is in
  - **I** - IRQ Enable (1 = disable & 0 = enable)
  - **F** – FIQ Enable (1 = disable & 0 = enable)
  - **T** – Thumb Mode (1 = thumb mode & 0 – ARM mode)



Program Status register

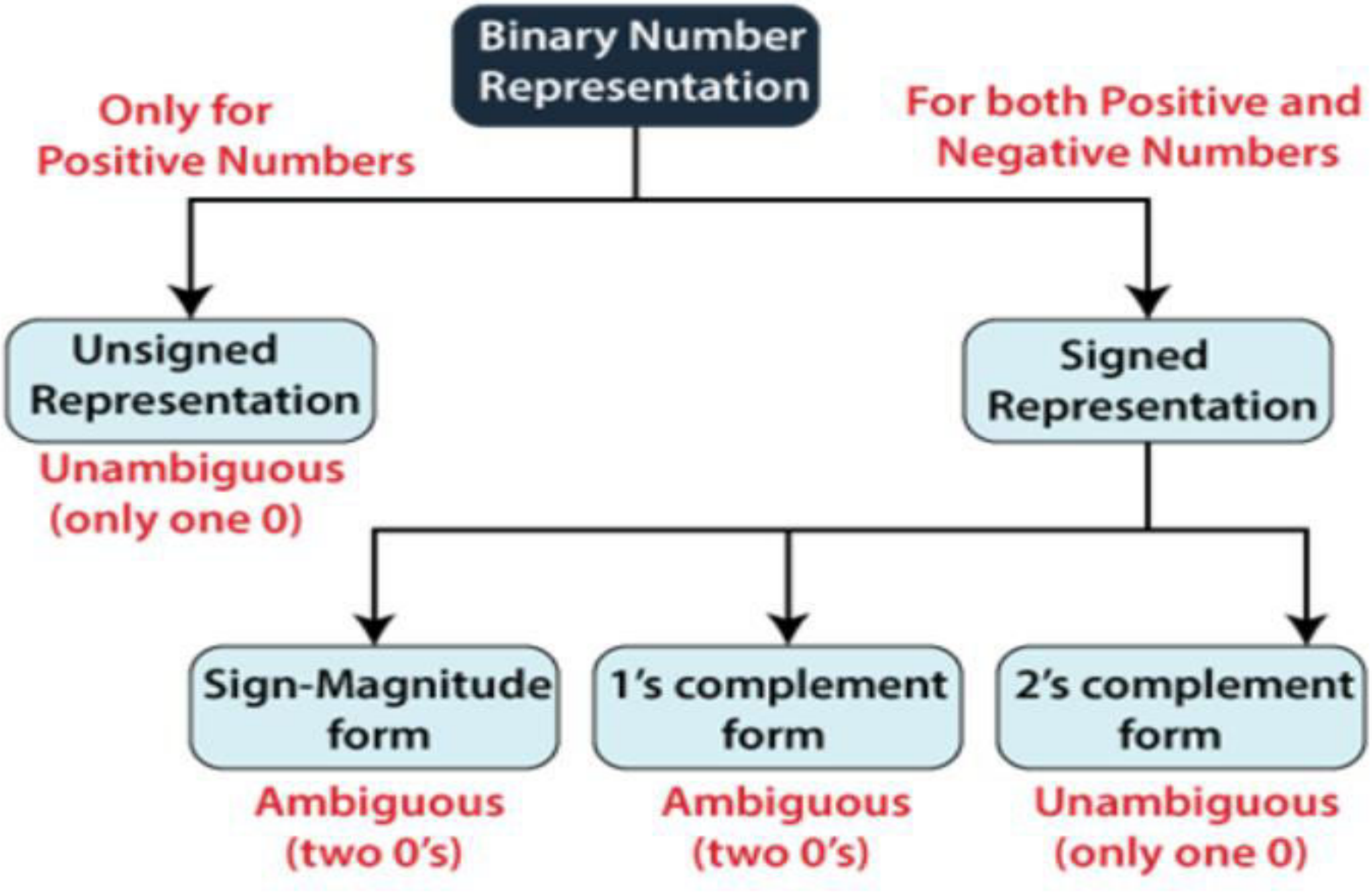


- N: Negative; the last ALU operation which changed the flags produced a negative result (the top bit of the 32-bit result was a one).
- Z: Zero; the last ALU operation which changed the flags produced a zero result (every bit of the 32-bit result was zero).
- C: Carry; the last ALU operation which changed the flags generated a carry-out, either as a result of an arithmetic operation in the ALU or from the shifter.
- V: oVerflow; the last arithmetic ALU operation which changed the flags generated an overflow into the sign bit.

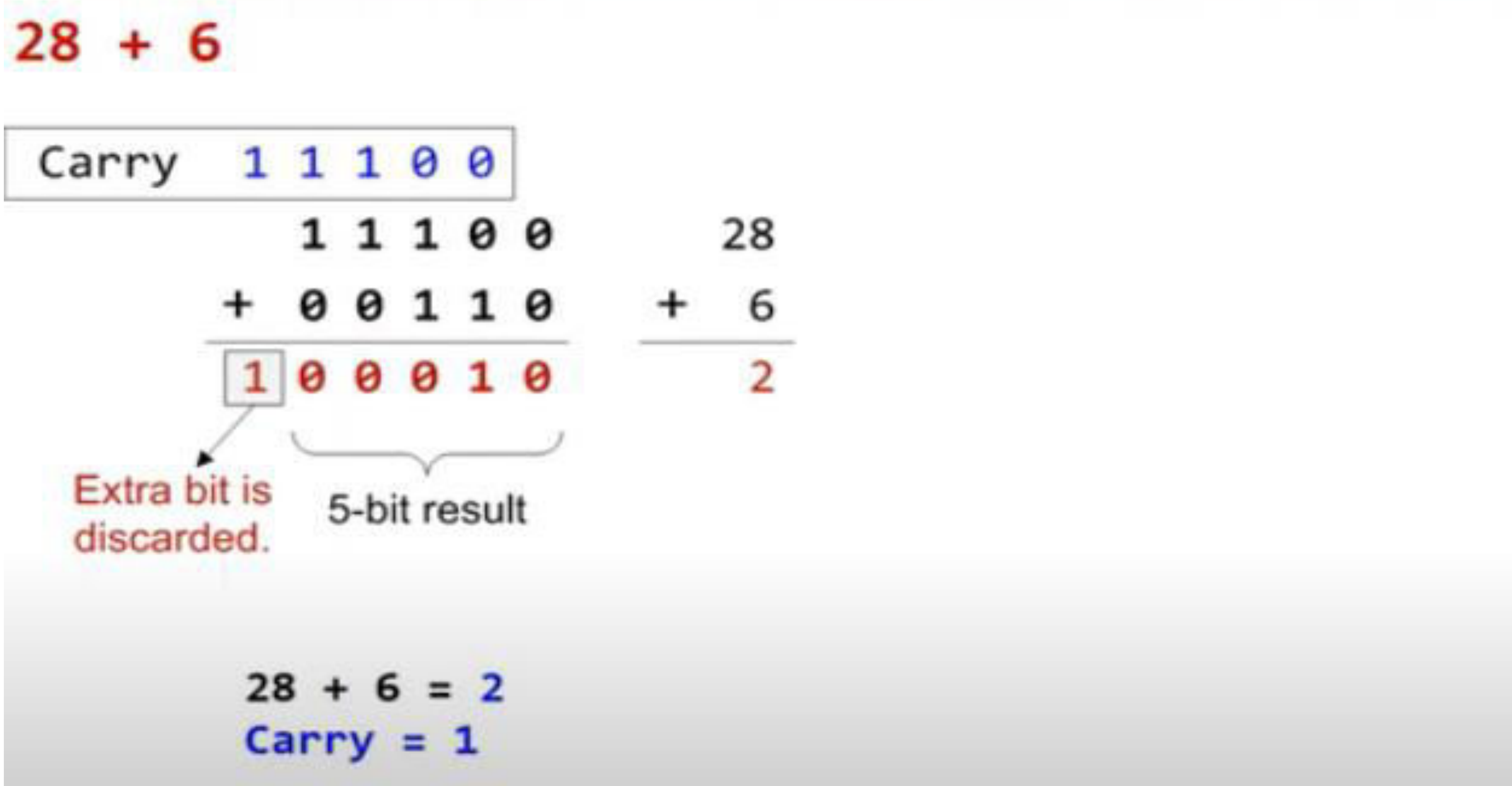


Common Number Systems

System	Base	Symbols	Used by humans?	Used in computers?
Decimal	10	0, 1, … 9	Yes	No
Binary	2	0, 1	No	Yes
Octal	8	0, 1, … 7	No	No
Hexa-decimal	16	0, 1, … 9, A, B, … F	No	No


Binary Number Representation
Only for Positive Numbers
For both Positive and Negative Numbers
Unsigned Representation
Unambiguous (only one 0)
Signed Representation
Sign-Magnitude form
Ambiguous (two 0's)
1's complement form
Ambiguous (two 0's)
2's complement form
Unambiguous (only one 0)

Carry Flag for Unsigned Addition



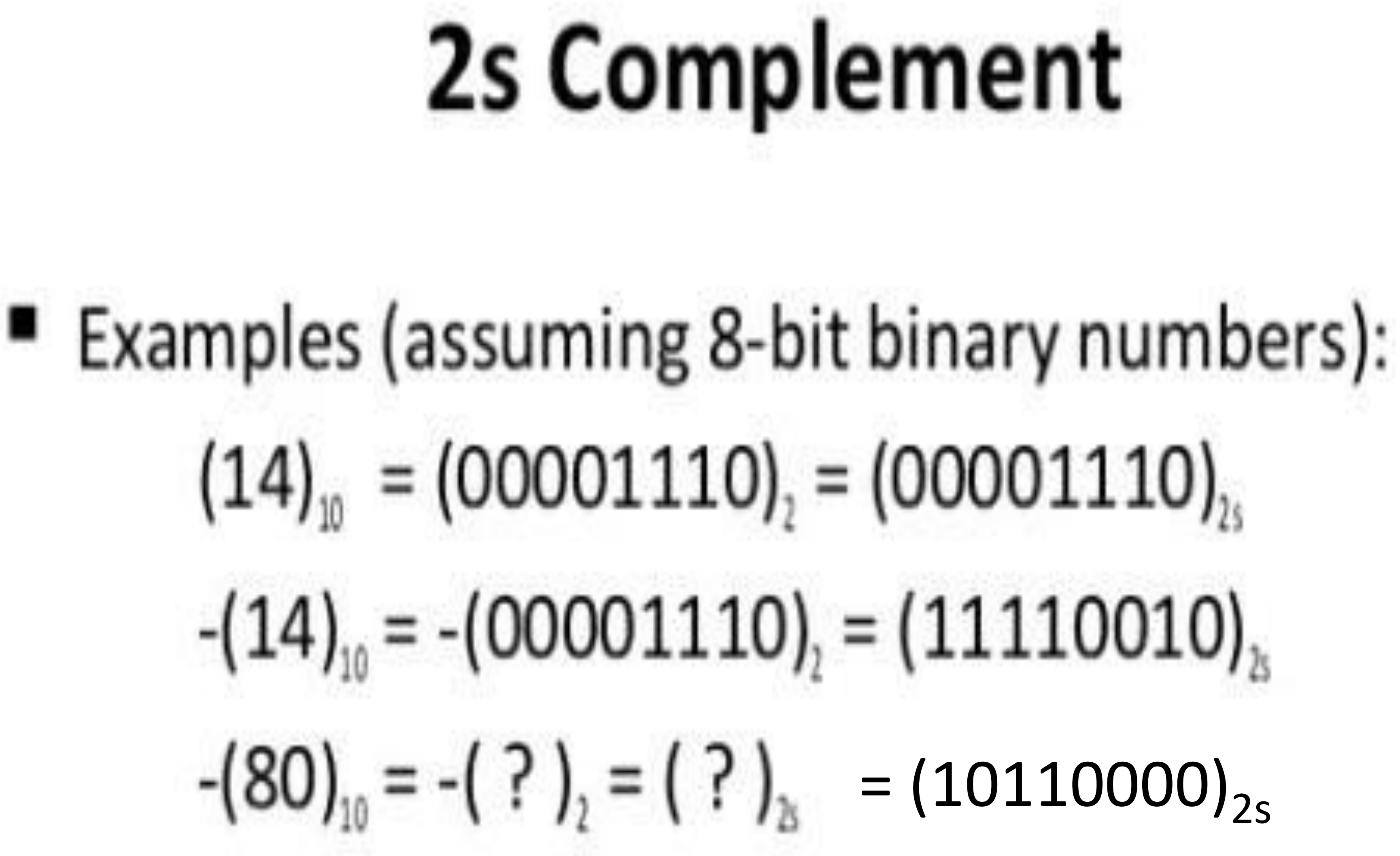
28 + 6

```
Carry   1 1 1 0 0
          1 1 1 0 0        28
      +   0 0 1 1 0     +   6
      ---------------   -------
        1 0 0 0 1 0         2
```

Extra bit is discarded.

5-bit result

28 + 6 = 2
Carry = 1



2s Complement

- Examples (assuming 8-bit binary numbers):

$(14)_{10} = (00001110)_2 = (00001110)_{2s}$

$-(14)_{10} = -(00001110)_2 = (11110010)_{2s}$

$-(80)_{10} = -(\ ?\ )_2 = (\ ?\ )_{2s}$ $= (10110000)_{2s}$

2’s Complement – Signed Numbers

5	0 101
+ 2	0 010
+ 7	0 111

Cin = 1

5	0 101
+ 4	0 100
+ 9	1 001

Cout = 0

Cin != Cout

Here Overflow bit is set

2	0 010
- 5	1 011
- 3	1 101
-3	-0 011

Cin = Cout , So no overflow

-5	1 011
- 4	1 100
-9	10 111

Cin != Cout , So overflow is set and carry flag is set

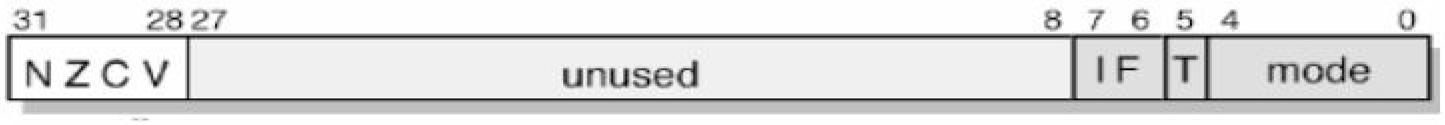
Overflow occurs when

- Two negative numbers are added and an answer comes positive or
- Two positive numbers are added and an answer comes as negative.

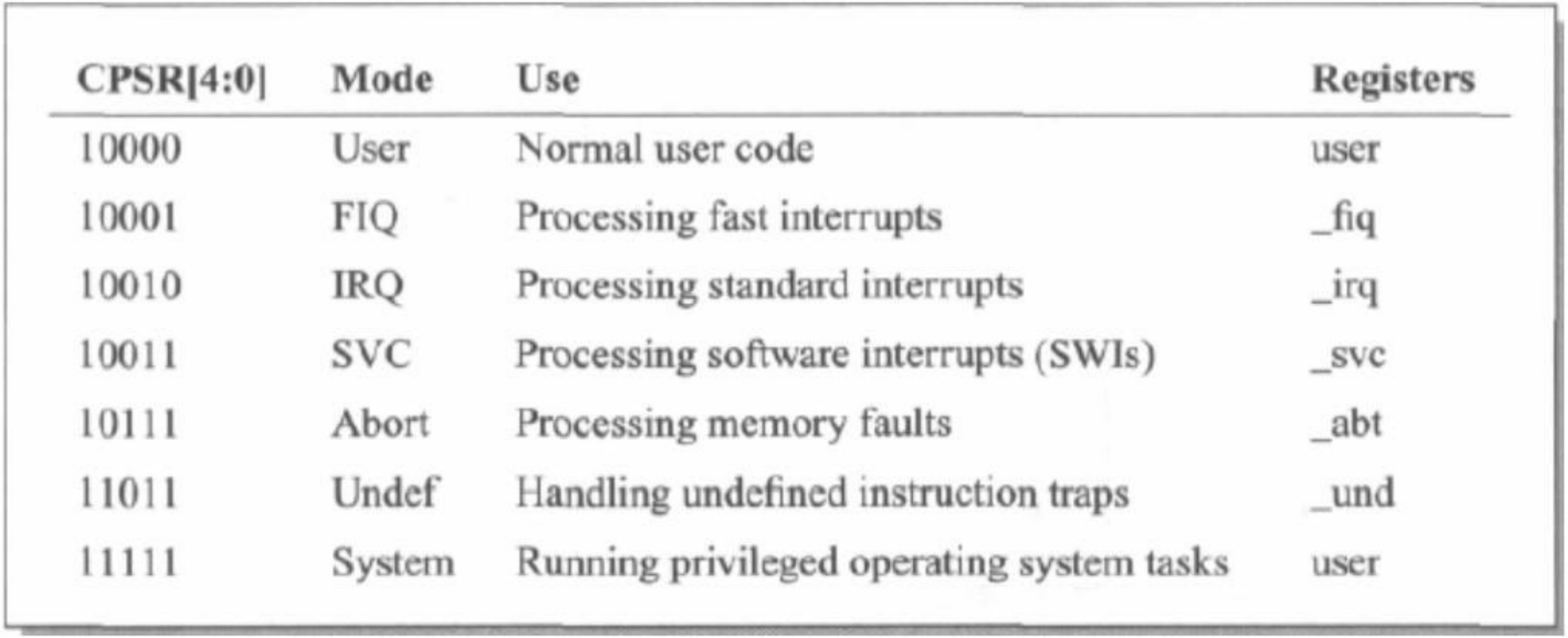
- So overflow can be detected by checking Most Significant Bit(MSB) of two operands
- Overflow can also be detected using 2 Bit Comparator just by checking Carry-in(C-in) and Carry-Out(C-out) from MSB's.

Program Status register

- **Current Program Status Register (CPSR):**
  - Monitors and controls internal operations
- **N** – Negative / sign (1 = negative & 0 = positive)
- **Z** – Zero (1 = zero & 0 = non zero)
- **C** – Carry (1 = carry occurred & 0 = no carry) – for unsigned operations
- **V** – Overflow (1 = overflow occurred & 0 = no overflow) – for signed operations
- **Mode**: Indicates which mode the processor is in
  - **I** - IRQ Enable (1 = disable & 0 = enable)
  - **F** – FIQ Enable (1 = disable & 0 = enable)
  - **T** – Thumb Mode (1 = thumb mode & 0 – ARM mode)



Processor Modes



CPSR[4:0]	Mode	Use	Registers
10000	User	Normal user code	user
10001	FIQ	Processing fast interrupts	_fiq
10010	IRQ	Processing standard interrupts	_irq
10011	SVC	Processing software interrupts (SWIs)	_svc
10111	Abort	Processing memory faults	_abt
11011	Undef	Handling undefined instruction traps	_und
11111	System	Running privileged operating system tasks	user

Program Counter (r15)

- A register in the control unit of the CPU that is used to keep track of the address of the current or next instruction.
- When the processor is executing in ARM state:
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned).
- When the processor is executing in Thumb state:
  - All instructions are 16 bits wide
  - All instructions must be halfword aligned
  - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned).
- When the processor is executing in Jazelle state:
  - All instructions are 8 bits wide
  - Processor performs a word access to read 4 instructions at once

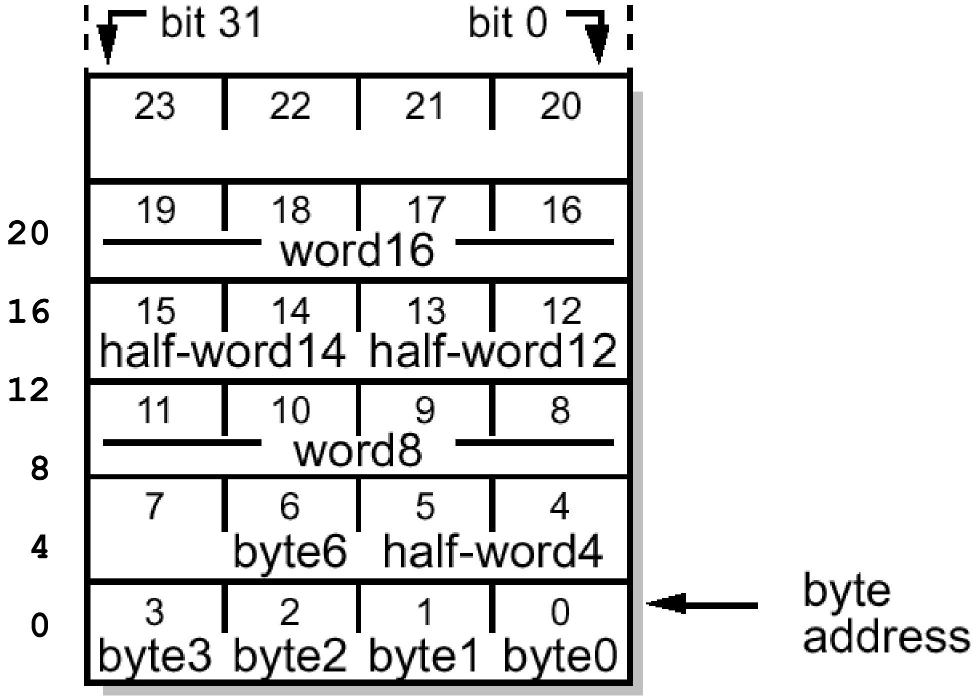
ARM’s Memory Organization

- Byte addressed memory
- Maximum 2^{32} bytes of memory
- A word = 32-bits, half-word = 16 bits
- Words aligned on 4-byte boundaries

NB - Lowest byte address = LSB of word

“Little-endian”

Word addresses follow LSB byte address



Thank You