Embedded System(EC381)

EC381	Embedded Systems	3-0-0-6

Introduction: Introduction to embedded systems with examples, Concept of real-time system, Challenges in embedded system design.

Embedded System Architecture: Basic Embedded processor/Microcontroller architecture, CISC (8051), RISC (ARM) Architecture, and Harvard Architecture (PIC).

Designing Embedded computing platform: The CPU Bus, memory devices, I/O devices, component interfacing, Design with microprocessor.

Embedded system design with FPGs: Introduction to FPGA and Verilog HDL, Hardware Design with Verilog HDL.

Processes and Operating Systems: Multiple Tasks and Multiple Processes; Preemptive Real-Time Operating Systems, Priority-Based Scheduling, Interprocess Communication Mechanisms, Evaluating Operating System Performance, Power Management and Optimization for Processes.

Networks: Distributed embedded architectures; Networks for embedded systems.

Case studies: Washing machine, Inkjet printer, telephone exchange, etc.

Texts:

1. W. Wolf, "Computers as components: Principles of embedded computing system design", 2/e, Elsevier, 2008. 2. Samir Palnitkar, “Verilog HDL: A Guide to Digital Design and Synthesis”, Prentice Hall, 2003

References:

1.D. Symes, and C. Wright, "ARM system developer's guide: Designing and optimizing system software",Elsevier,2008. 2. The 8051 Microcontroller and Embedded Systems by Muhammad Ali Mazidi, Janice G.Mazidi, RolinD.McKinlay 3. Jack Ganssle, "The art of designing embedded systems", 2/e, Elsevier, 2008. 4. M. D. Ciletti , “Advanced Digital Design with the Verilog HDL”, Prentice Hall, 2010

- Introduction to embedded systems
- Applications of embedded systems
- Typical subsystems in an embedded system

Prepared By Indranil Sengupta, IITKGP

- Introduction to embedded systems
- Applications of embedded systems
- Typical subsystems in an embedded system

Introduction

- We have been brought up in the age of computing.
  - Computers are everywhere (some we see, some we do not see).
- Types of computers we are familiar with:
  - Desktops and Laptops
  - Servers
  - Mobile phones
- But there's another type of computing system that is often hidden.
  - Far more common and pervasive...
  - Hidden in the environment.

What are Embedded Systems?







- Computers are embedded within other systems:
  - What is "*other systems*"? – Hard to define.
    - Any computing system other than desktop / laptop server.
  - Typical examples:
    - Washing machine, refrigerator, camera, vehicles, airplane, missile, printer.
    - Processors are often very simple and inexpensive (depending on application of course).
  - Billions of embedded system units produced yearly, versus millions of desktop units.

Common Features of Embedded Systems

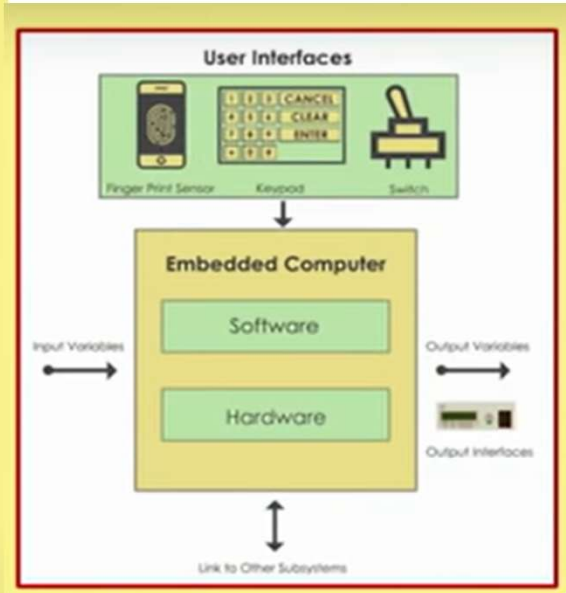
- They are special-purpose or single-functioned.
  - Executes a single program, possibly with inputs from the environment.
  - Imagine a microwave oven, a washing machine, an AC machine, etc.
- Tight constraints on cost, energy, form factor, etc.
  - Low cost, low power, small size, relatively fast.
- They must react to events in real-time.
  - Responds to inputs from the system's environment.
  - Must compute certain results in real-time without delay.
    - The delay that can be tolerated depends on the application.

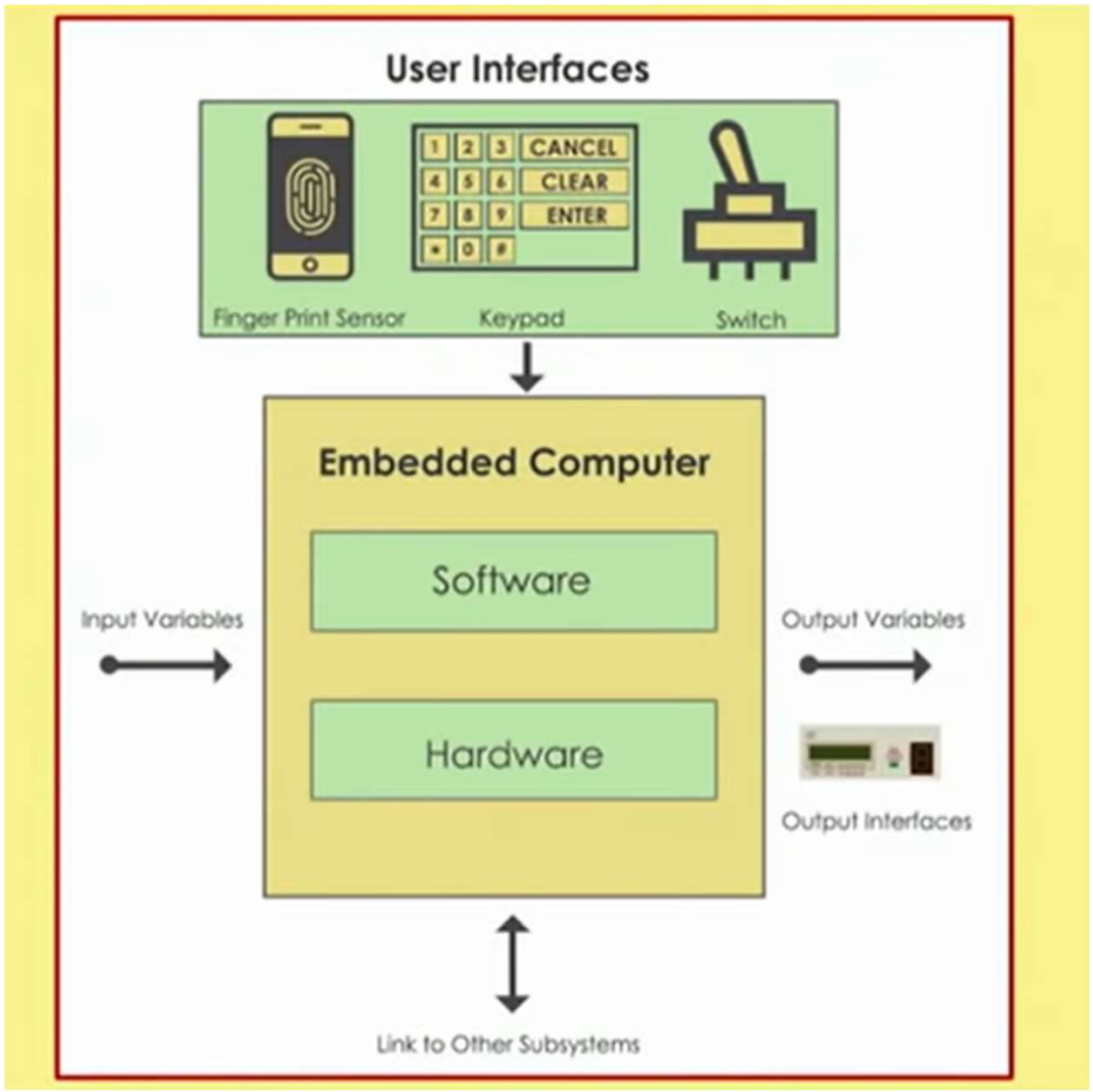
Typical Design Constraints

- Low Cost
  - A sophisticated processor can increase the cost of the embedded system.
- Low Energy Consumption
  - Many embedded systems operate on battery.
- Limited Memory
  - Typically constrained to a finite and small amount of memory.
- Real-Time Response
  - Most embedded systems are used for controlling some equipment.
  - Must generate response within a specified time.

How to define an Embedded System?

- It is a microcontroller-based system that is designed to control a function or range of functions, and is not meant to be programmed by the end user.
  - The user may make choices concerning the functionality but cannot change them.
  - The user cannot make modifications to the software.
  - Can you "*program*" your washing machine or refrigerator or car?
    - Not today … but not very sure of the near future.




User Interfaces
1 2 3 CANCEL
4 5 6 CLEAR
7 8 9 ENTER
* 0 #
Finger Print Sensor
Keypad
Switch
Embedded Computer
Software
Hardware
Input Variables
Output Variables
Output Interfaces
Link to Other Subsystems

Applications of Embedded Systems

- ***Limited by imagination.***

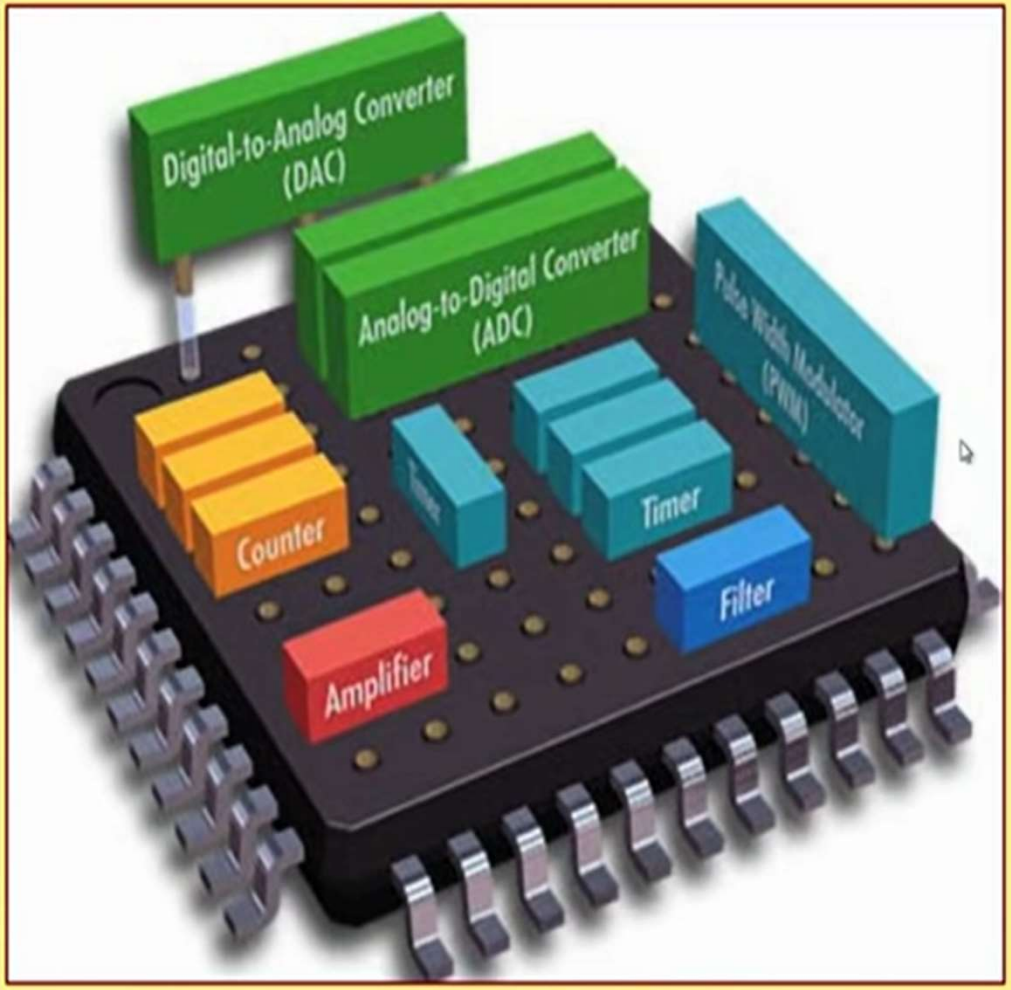
a) **Consumer Segment**: Refrigerator, washing machine, A/C machine, camera, microwave oven, TV, security system, etc.

b) **Office Automation**: Printers, Fax machines, photocopying machines, scanners, biometric scanner, surveillance camera, etc.

c) **Automobiles**: Air bags, anti-lock braking system (ABS), engine control, door lock, GPS system, vehicular ad-hoc network (VANET), etc.

d) **Communication**: Mobile phones, network switches, WiFi hotspots, telephones, MODEM, etc.

e) **Miscellaneous**: Automatic door locks, automatic baggage screening, surveillance systems, intelligent toilet, etc.

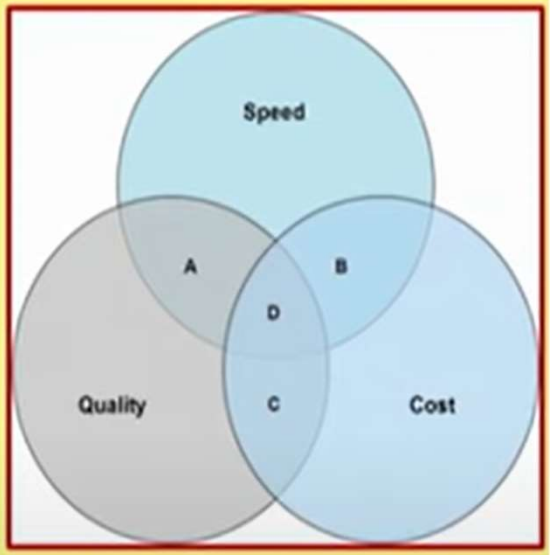


Notable subsystems:

a) Analog-to-digital (ADC) interfaces
b) Digital-to-analog (DAC) interfaces
c) Pulse-width-modulation (PWM) interfaces
d) Timers and counters
e) In addition to … processor, memory, digital I/O ports, etc.

Design Challenges

- Primary design goal:
  - An implementation that realizes the desired functionality.
- The main design challenge is …
  - To simultaneously optimize several design metrics.
  - Often mutually conflicting.
- What is a design metric?
  - Some feature of an implementation that can be *measured* and *evaluated*.



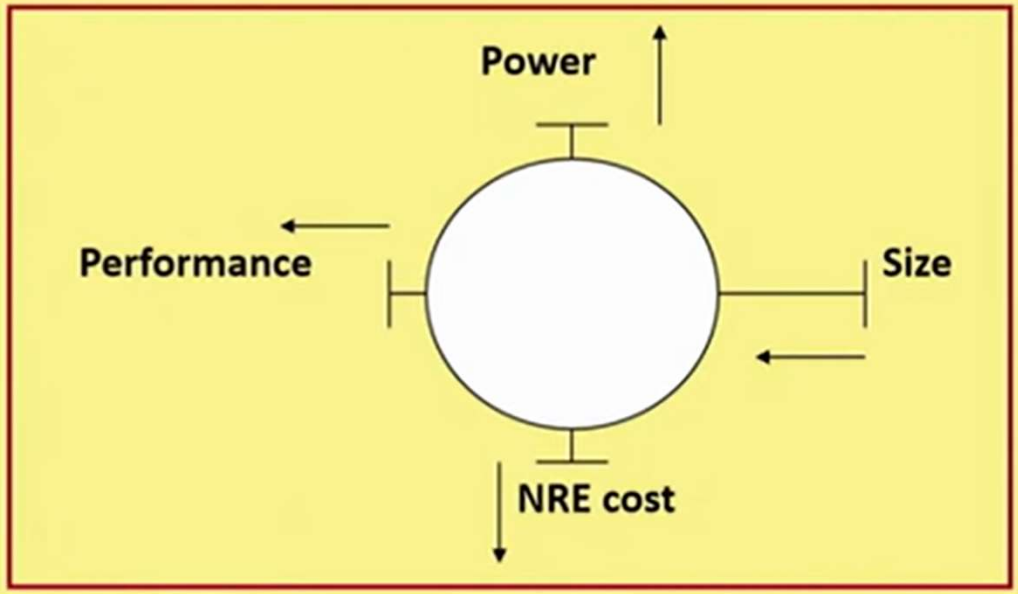
Common Design Metrics

- **Non Recurring Engineering (NRE) Cost**: One-time initial cost of designing a system.
- **Unit Cost**: The cost of manufacturing each copy of the system, without counting the NRE cost.
- **Size**: The actual physical space occupied by the system.
- **Performance**: This is measured in terms of the time taken or throughput.
- **Power**: The amount of (battery) power consumed by the system.
- **Flexibility**: The ability to change the functionality of the system.

- **Maintainability**: How easy or difficult it is to modify the design of the system?
- **Time-to-prototype**: How much time is required to build a working version of the system (i.e. a prototype)?
- **Time-to-market**: How much time is required to develop a system such that it can be released to the market commercially?
- **Safety**: Are there any adverse effects on the operating environment?
- Can be many more ...

Design Tradeoff

- Many of the design metrics can be mutually conflicting.
  - Improving one may degrade the other (e.g. power, performance, size and NRE cost, etc.).



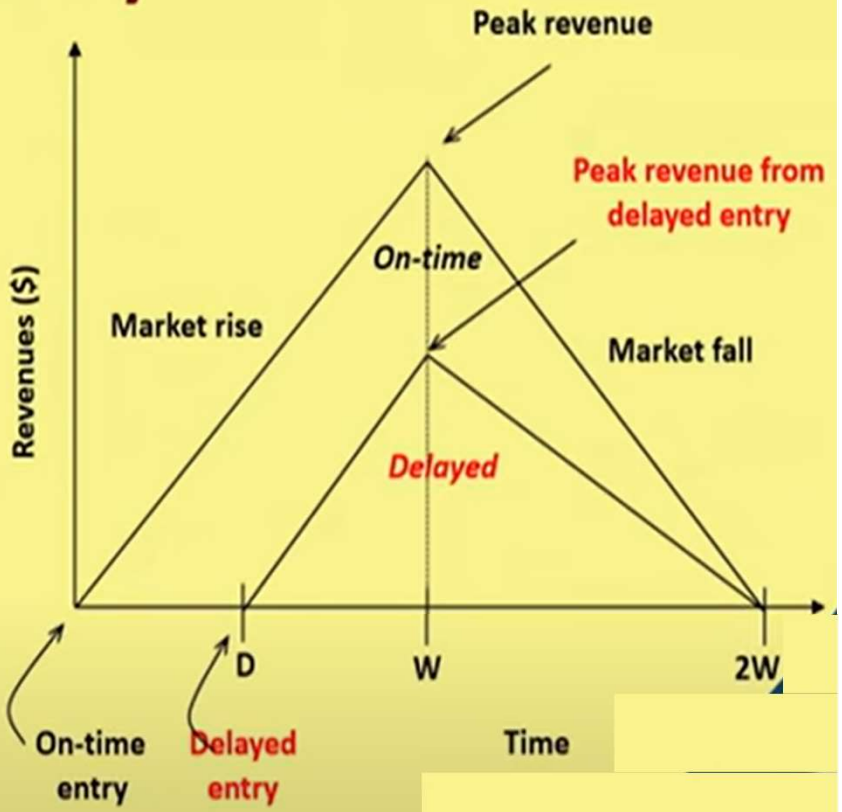
- Often requires expertise in both hardware and software to take a proper decision.
  - Expertise in hardware may indicate the types of co-processor or I/O interfaces to use for specific applications (e.g. analog ports, digital ports, PWM ports, etc.).
  - Expertise in software is required to identify parts of the implementation that need to be implemented in software and run on the microcontroller.
  - Hardware / Software Co-design becomes important.

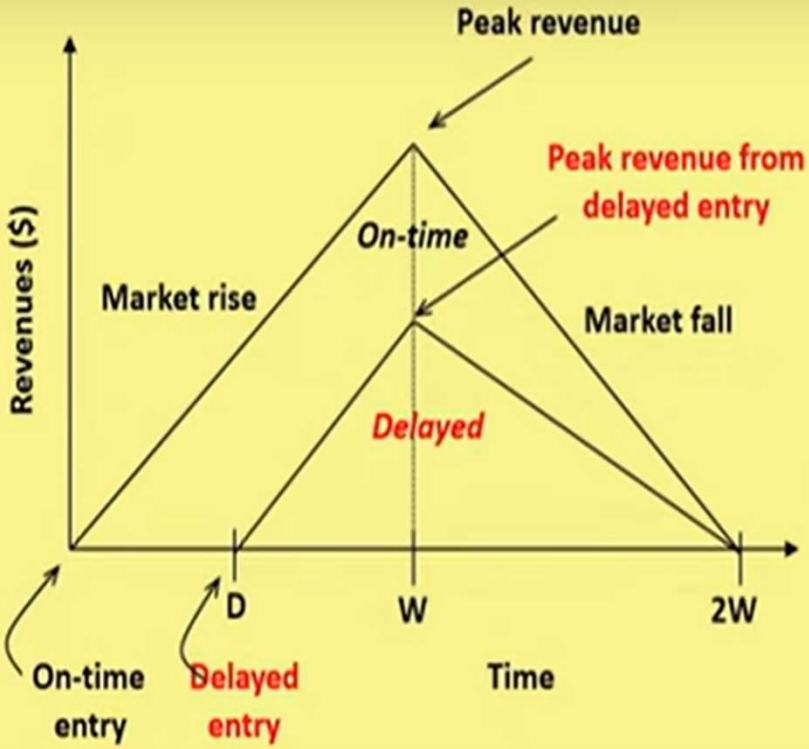
Time-to-market Design Metric

- This is a very crucial design metric.
  - Must be strictly followed to make a product commercially viable.
  - Requires exhaustive market study and analysis.
- Starting from the point a product design starts, we can define a *Market Window* within which it is expected to have the highest sales.
  - Any delay can result in drastic reductions in sales.

Loss due to Delayed Market Entry

- Consider a simplified revenue model:
  - Product life is *2W*
  - Maximum sale occurs at time *W*
  - Market rise and market fall defines a *triangle* in the revenue graph.
  - *Area of the triangle determines the total revenue.*
- For delayed entry, can estimate the loss.
  - Difference in areas of the expected and actual triangles.





- Area of a triangle = ½ * base * height
  - Area (on-time) = ½ * 2W * W
  - Area (delayed) = ½ * (2W – D) * (W – D)
  - Percentage revenue loss = $D(3W - D)/2W^2 * 100$
- Examples:
  - 2W = 52 weeks, D = 4 weeks → LOSS = 22%
  - 2W = 52 weeks, D = 10 weeks → LOSS = 50%

NRE and Unit Cost Metrics

- If C_{NRE} denotes the NRE cost and C_{unit} the unit cost of a product, then the total cost for manufacturing N units is given by:

  $Total\ Cost = C_{NRE} + N * C_{unit}$

- Therefore, per-unit cost is given by: $C_{NRE} / N + C_{unit}$
- Example:
  - C_{NRE}= Rs. 5,00,000 and C_{unit} = Rs. 5,000
  - Total cost for manufacturing 100 units = 5,00,000 + 5000 * 100 = 10,00,000
  - Per unit cost = 5,00,000 / 100 + 5000 = 10,000

$$\text{Per-unit cost} = C_{NRE} / N + C_{unit}$$

- We can compare technologies by cost:
  - Choice A: C_{NRE} = Rs. 20,000, C_{unit} = Rs. 8,000
  - Choice B: C_{NRE} = Rs. 4,00,000, C_{unit} = Rs. 3,000
  - Choice C: C_{NRE} = Rs. 10,00,000, C_{unit} = Rs. 8,000
- Of course, time-to-marker cost must also be considered.

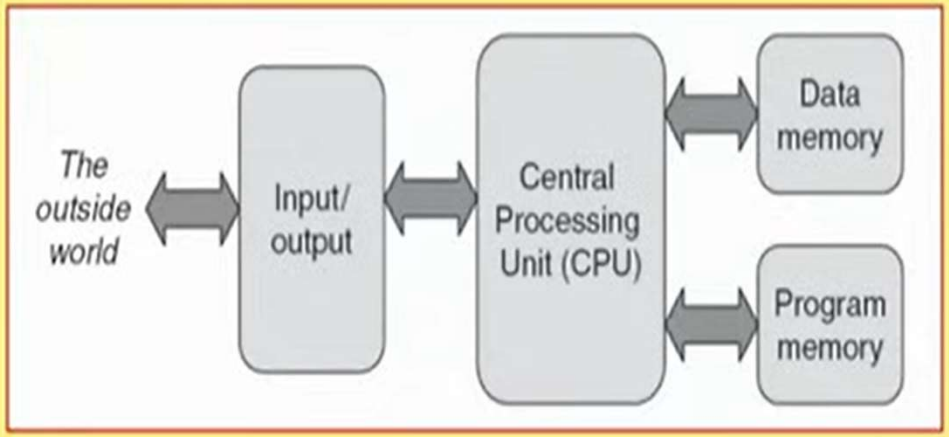
Performance Design Metric

- Most widely used, but can also be most misleading.
  - Must be careful in the evaluation.
- Some of the measures:
  - Clock frequency, MIPS --- not very good for comparison
  - Latency (response time)
  - Throughput
- Measure of speedup among design alternatives.

- Classification of computer architecture
- Characteristics of a microprocessor
- Characteristics of a microcontroller

Basic Operation of a Computing System

- The central processing unit (CPU) carries out all computations.
  - Fetches instructions from the program memory and executes it; may require access to data in data memory.
- The input/output block provides interface with the outside world.
  - Allows users to interact with the computing system, and also observe the output results.

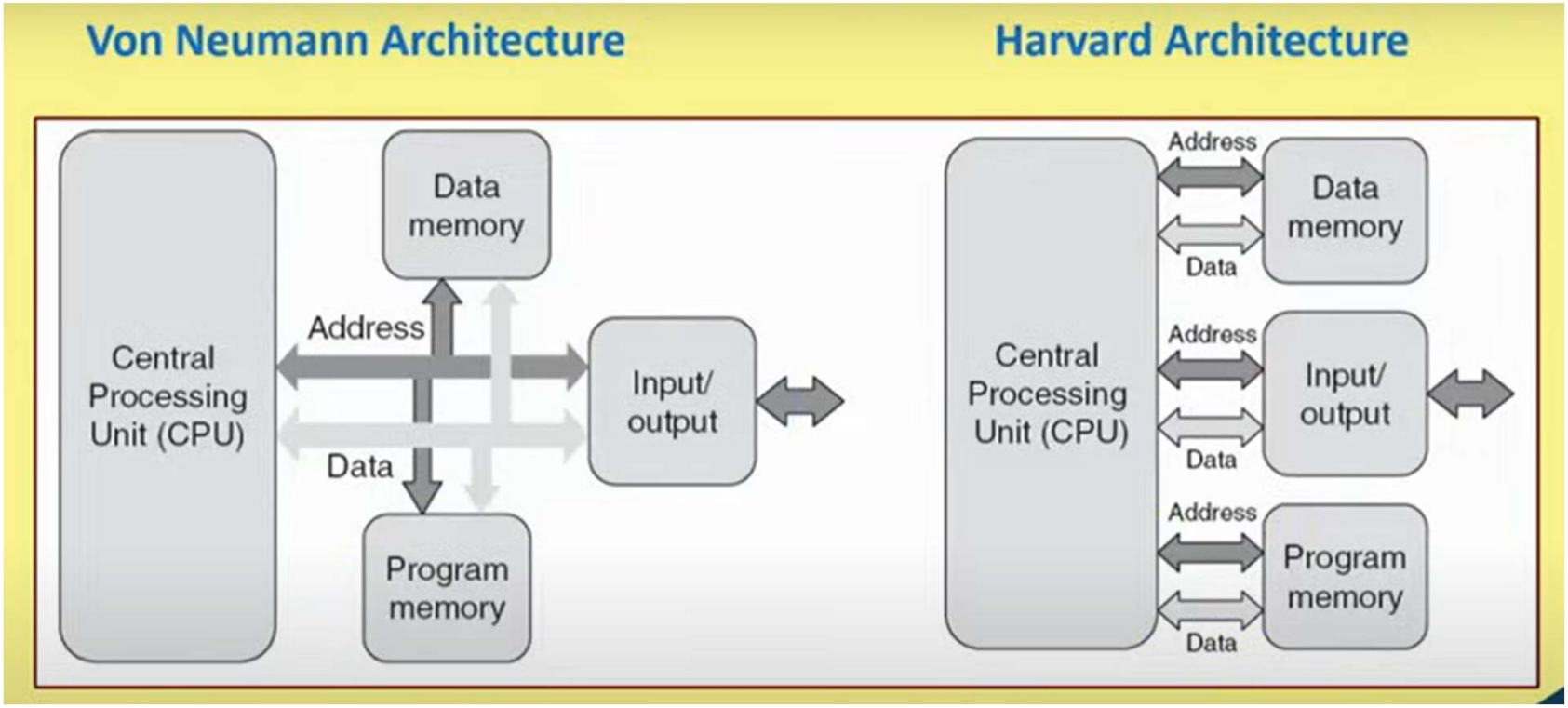


- About the instruction set architecture (ISA) of the CPU.
    a) Complex Instruction Set Computer (CISC)
        - Typically used in desktops, laptops and servers (courtesy Intel).
    b) Reduced Instruction Set Computer (RISC)
        - Typically used in microcontrollers, that are used to build embedded systems.

- Two different types of memory:
    a) Random Access Memory (RAM)
        - Volatile; used for data memory in microcontrollers.
    b) Read Only Memory (ROM)
        - Non-volatile; used for program memory in microcontrollers.

Classification of CPU Architecture

- Broadly two types of architectures:
  a) Von Neumann Architecture
     - Both instructions and data are stored in the same memory.
     - This model is followed in conventional computing systems.
  b) Harvard Architecture
     - Instructions and data are stored in separate memories.
     - Typically followed in microcontrollers, used for building embedded systems.
     - Instructions are stored in a ROM (permanent), while temporary data are in RAM.


Von Neumann Architecture
Harvard Architecture
Central Processing Unit (CPU)
Data memory
Address
Data
Input/ output
Program memory
Central Processing Unit (CPU)
Address
Data memory
Data
Address
Input/ output
Data
Address
Program memory
Data

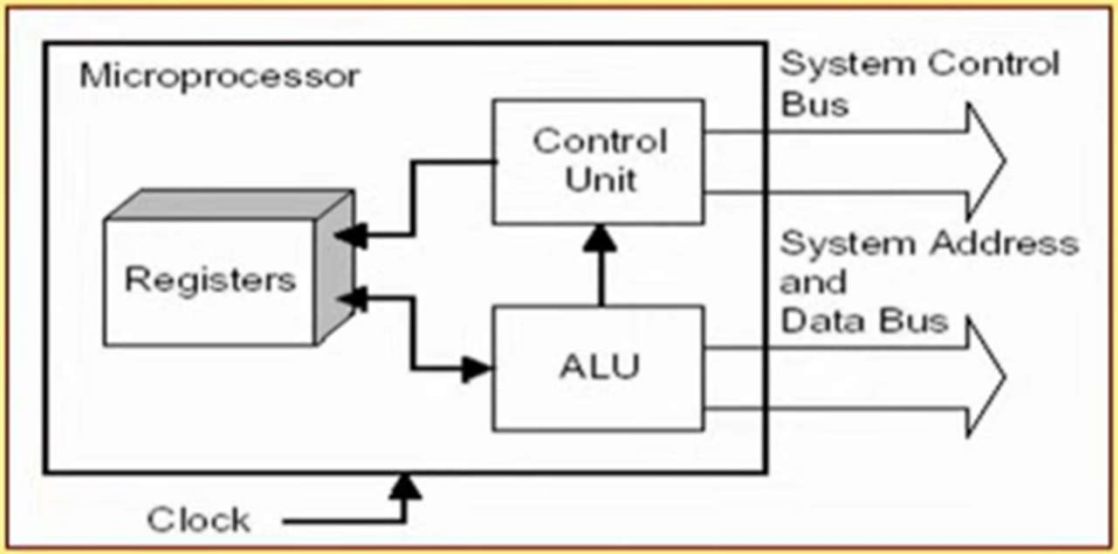
Von Neumann Vs Harvard

Parameters	Von Neumann	Harvard
Memory	❖ Data and Program {Code} are stored in same memory	❖ Data and Program {Code} are stored in different memory
Memory Type	❖ It has only RAM for Data & Code	❖ It has RAM for Data and ROM for Code
Buses	❖ Common bus for Address & Data/Code	❖ Separate Bus Address & Data/Code
Program Execution	❖ Code is executed serially and takes more cycles	❖ Code is executed in parallel with data so it takes less cycles.
Data/Code Transfer	❖ Data or Code in one cycle	❖ Data and Code in One cycle
Control Signals	❖ Less	❖ More
Space	❖ It needs less Space	❖ It needs more space
Cost	❖ Less	❖ Costly

What is a Microprocessor?

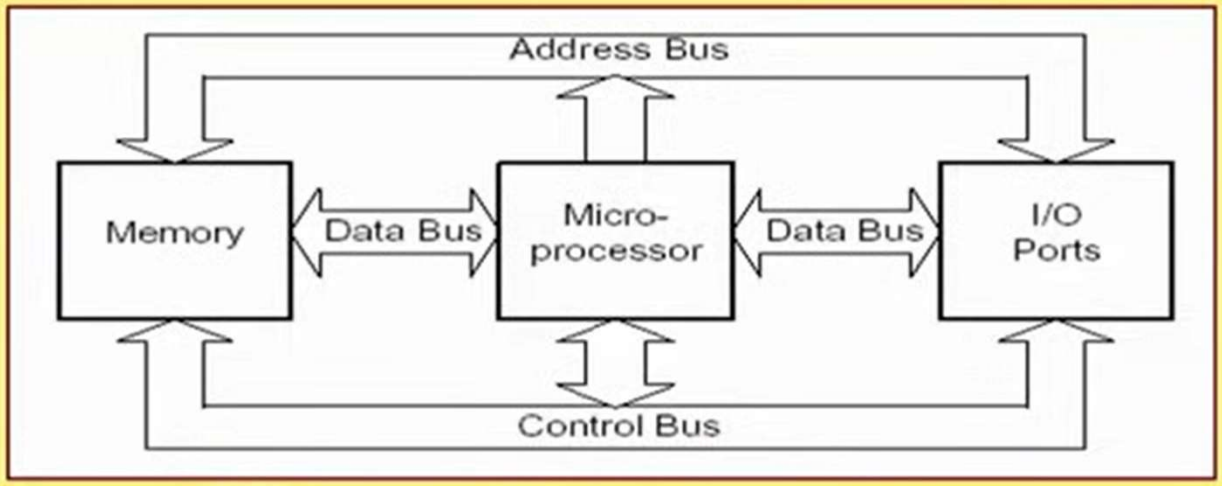
- It is basically the entire CPU fabricated on a single chip.
  - Consists of a set of registers to store temporary data.
  - Consists of an arithmetic logic unit (ALU), where all arithmetic and logical computations are carried out.
  - Consists of some mechanism to interface external devices (memory and I/O) through buses (address, data and control).
  - Consists of a control unit that synchronizes the operation.

Schematic Diagram of Microprocessor



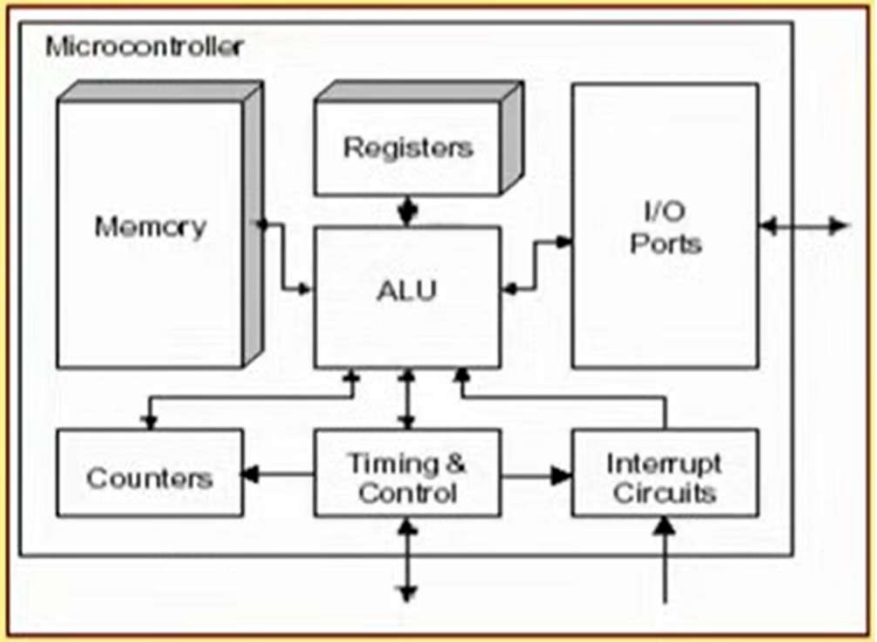
What is a Microcomputer?

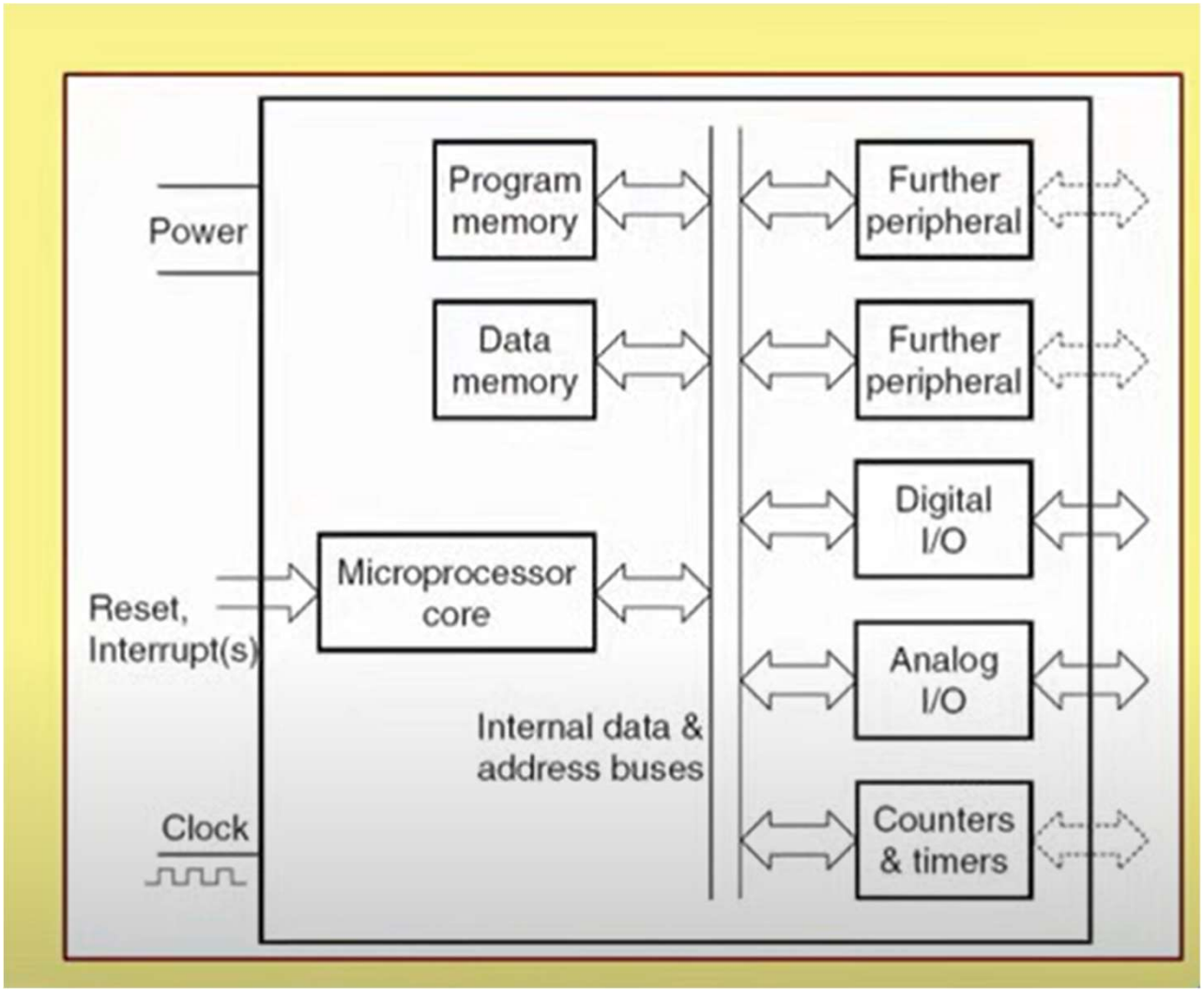
- It is a computer system built using a microprocessor.
- Since a microprocessor does not contain memory and I/O, we have to interface these to build a microcomputer.
  - Too complex and expensive for very small and low-cost embedded systems.



Microcontrollers: The Heart of Embedded Systems

- It is basically a computer on a single chip.
  - Very inexpensive, small, low power.
  - Convenient for use in embedded system design.
- It operates on data that are fed through its serial or parallel input ports, controlled by the software stored in on-chip memory.
  - Often has analog input pins, timers and other utility circuitry built-in.




Power
Program memory
Data memory
Further peripheral
Further peripheral
Digital I/O
Microprocessor core
Reset, Interrupt(s)
Analog I/O
Internal data & address buses
Clock
Counters & timers

Microcontroller Packaging and Appearance

From left to right:

PIC 12F508, PIC 16F84A, PIC 16C72, Motorola 68HC05B16, PIC 16F877, Motorola 68000

How Microcontrollers are different from PCs?

- When a PC executes a program, the program is first loaded from disk/SSD into an allocated section of memory.
  - Usually the program is loaded part by part to conserve memory space.
  - There is a complicated operating system that handles all low-level operations (includes low-level driver codes for interfacing with various devices).
- In a microcontroller there is no disk to read from.
  - On-chip ROM stores the program that is to be executed.
  - Size of the ROM limits the maximum size of the application.
  - There is no operating system, and the program is ROM is the only program that is running (must include low-level routines).

Where are Microcontrollers Used?

- Typically in applications where processing power is not critical.
  - Modern-day household can have 10 to 50 such devices embedded in various devices and equipments.
- One-third of the applications are in the office automation segment.
- Another one-third are in consumer electronics goods.
- Rest one-third are used in automotive and communication applications.

Evolution of Microcontrollers

- Microcontroller evolved from a microprocessor-based board-level design to a single chip in the mid-1970's.
  - As the process of miniaturization continued, all of the components needed for a controller were built into a single chip.
- In the mid-1980's, microcontrollers got embedded into a larger ASIC (Application Specific Integrated Circuit).
  - Microcontrollers are fabricated as a module inside a larger chip.

Advantages of using microcontrollers

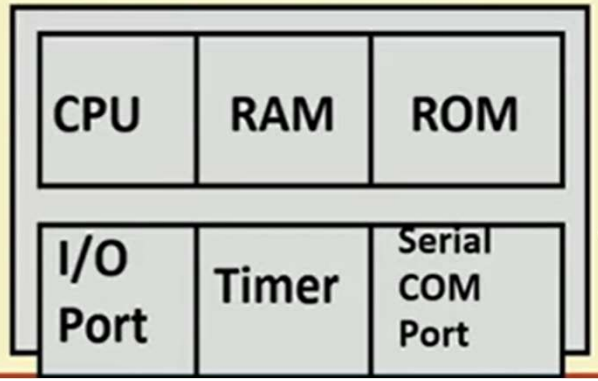
- Fast and effective
  - The architecture correlates closely with the problem being solved (control systems).
- Low cost / Low power
  - High level of system integration within one component.
  - Only a handful of components needed to create a working system.
- Compatibility
  - Opcodes and binaries are the SAME for all 80x51 / ARM / PIC variants.

8051 Microcontroller (CISC)

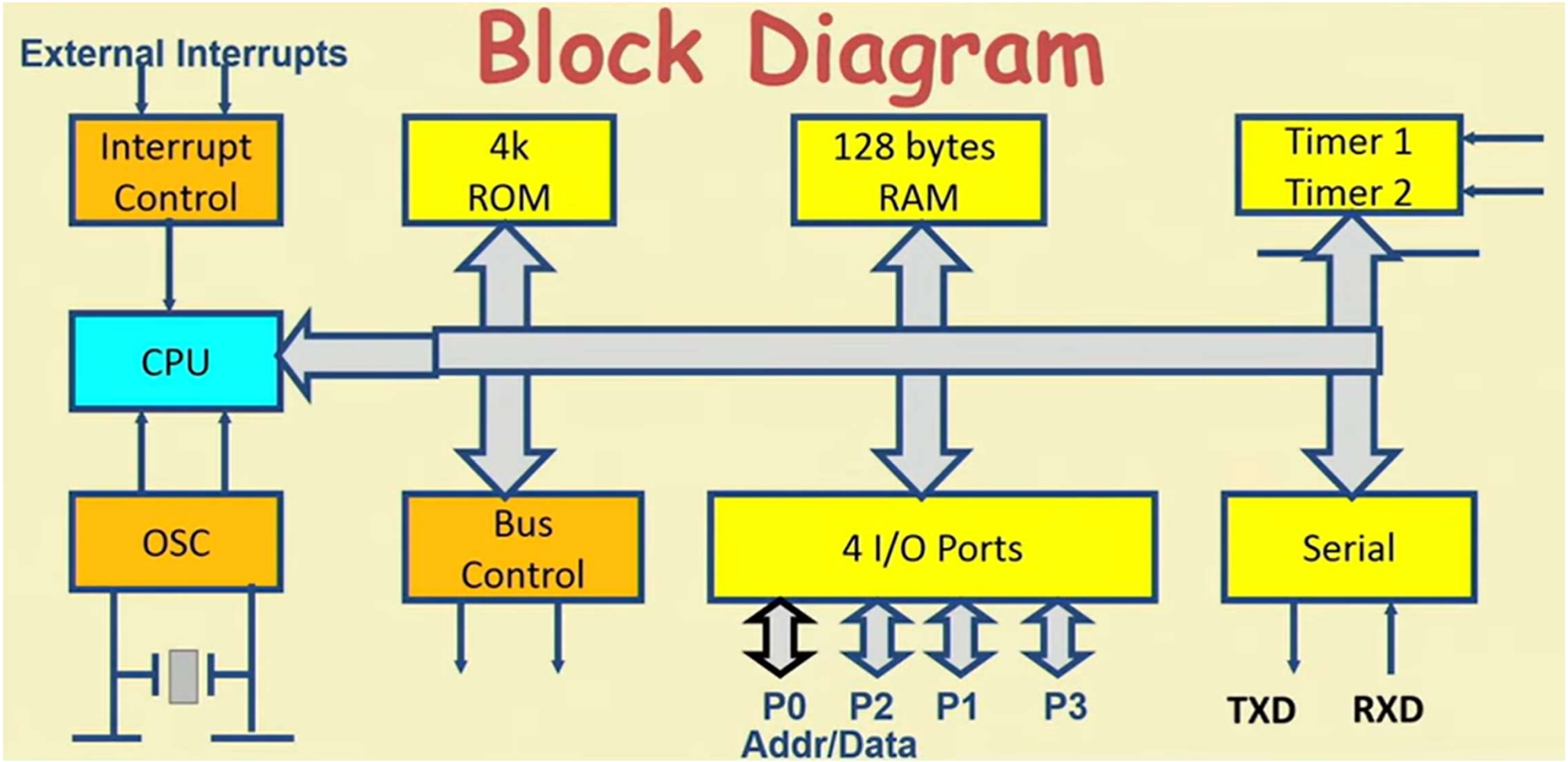
Prepared By Prof. Santanu Chattopadhyay, IITKGP

8051 Basic Component

- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface



A single chip Microcontroller


Block Diagram
External Interrupts
Interrupt Control
4k ROM
128 bytes RAM
Timer 1
Timer 2
CPU
OSC
Bus Control
4 I/O Ports
Serial
P0
Addr/Data
P2
P1
P3
TXD
RXD

Other 8051 featurs

- Only 1 On chip oscillator (external crystal)
- 6 interrupt sources (2 external , 3 internal, Reset)
- 64K external code (program) memory(only read)PSEN
- 64K external data memory(can be read and write) by RD,WR
- Code memory is selectable by EA (internal or external)
- We may have External memory as data and code

Three criteria in Choosing a Microcontroller

- meeting the computing needs of the task efficiently and cost effectively
  - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption
  - easy to upgrade
  - cost per unit
- availability of software development tools
  - assemblers, debuggers, C compilers, emulator, simulator, technical support
- wide availability and reliable sources of the microcontrollers

Comparison of the 8051 Family Members

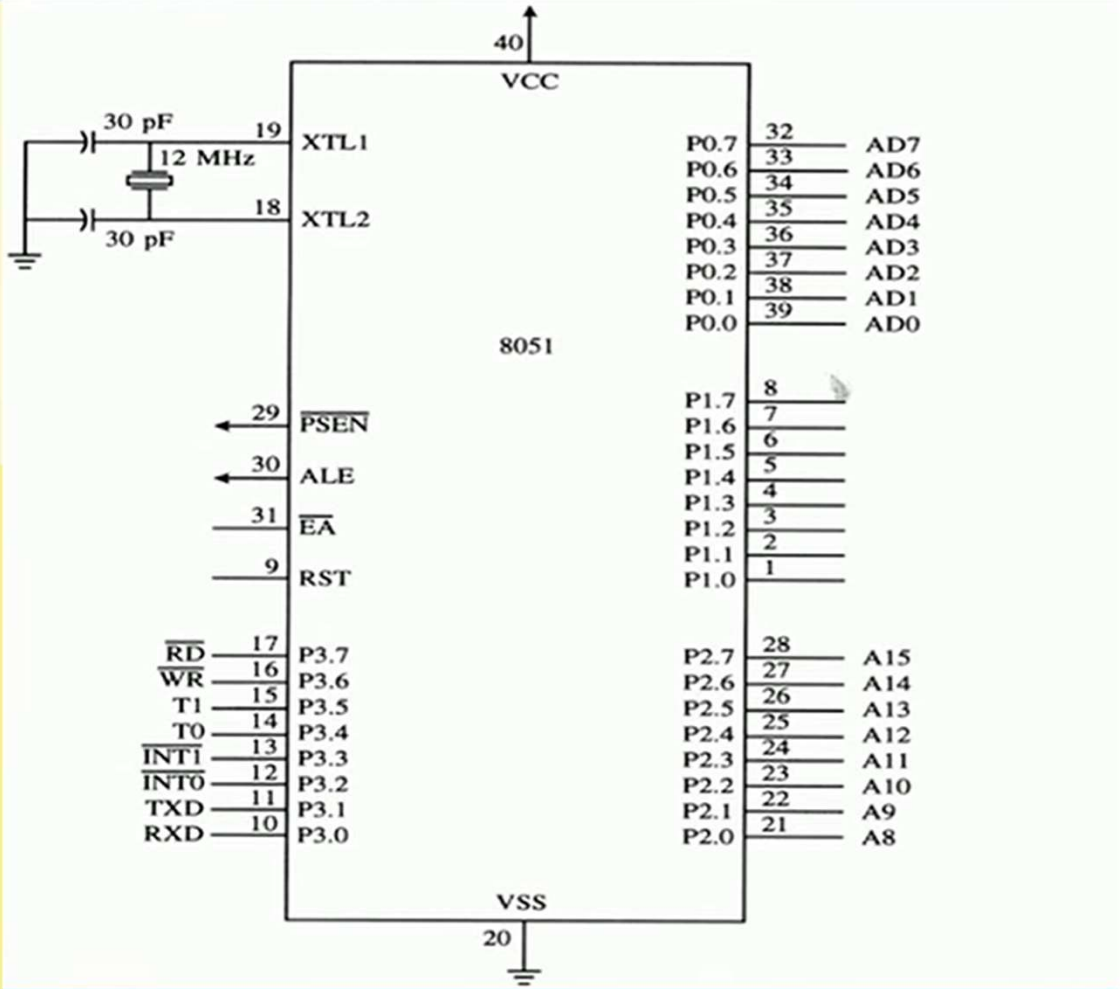
- ROM type
  - 8031 no ROM
  - 80xx mask ROM
  - 87xx EPROM
  - 89xx Flash EEPROM
- 89xx
  - 8951
  - 8952
  - 8953
  - 8955
  - 898252
  - 891051
  - 892051
- Example (AT89C51,AT89LV51,AT89S51)
  - AT= ATMEL(Manufacture)
  - C = CMOS technology
  - LV= Low Power(3.0v)

Comparison of the 8051 Family Members

89XX	ROM	RAM	Timer	Int Source	IO pin	Other
8951	4k	128	2	6	32	-
8952	8k	256	3	8	32	-
8953	12k	256	3	9	32	WD
8955	20k	256	3	8	32	WD
898252	8k	256	3	9	32	ISP
891051	1k	64	1	3	16	AC
892051	2k	128	2	6	16	AC

WD: Watch Dog Timer
AC: Analog Comparator
ISP: In System Programable

8051 Schematic Pin out



IMPORTANT PINS (IO Ports)

- **One of the most useful features of the 8051 is that it contains four I/O ports (P0 - P3)**

- Port 0 （pins 32-39）：P0（P0.0～P0.7）
  - 8-bit R/W - General Purpose I/O
  - Or acts as a multiplexed low byte address and data bus for external memory design

- Port 1 （pins 1-8）　：P1（P1.0～P1.7）
  - Only 8-bit R/W - General Purpose I/O

- Port 2 （pins 21-28）：P2（P2.0～P2.7）
  - 8-bit R/W - General Purpose I/O
  - Or high byte of the address bus for external memory design

- Port 3 （pins 10-17）：P3（P3.0～P3.7）
  - General Purpose I/O
  - if not using any of the internal peripherals (timers) or external interrupts.
- **Each port can be used as input or output (bi-direction)**

IMPORTANT PINS

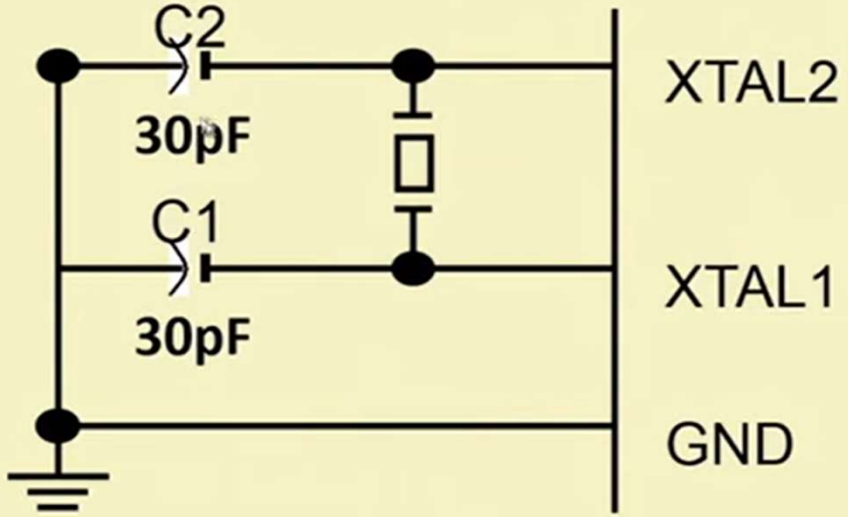
- **PSEN** (out): Program Store Enable, the read signal for external program memory (active low).
- **ALE** (out): Address Latch Enable, to latch address outputs at Port0 and Port2
- **EA** (in): External Access Enable, active low to access external program memory locations 0 to 4K
- **RXD**,**TXD**: UART pins for serial I/O on Port 3
- **XTAL1** & **XTAL2**: Crystal inputs for internal oscillator.

Pins of 8051

- Vcc（pin 40）：
  - Vcc provides supply voltage to the chip.
  - The voltage source is +5V.
- GND（pin 20）：ground
- XTAL1 and XTAL2（pins 19,18）：
  - These 2 pins provide external clock.
  - Way 1：using a quartz crystal oscillator
  - Way 2：using a TTL oscillator
  - Example 4-1 shows the relationship between XTAL and the machine cycle.

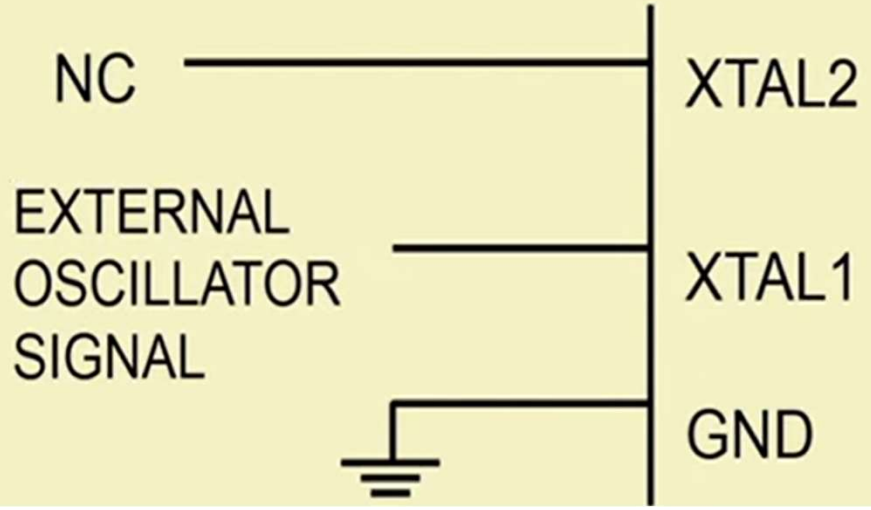
XTAL Connection to 8051

- Using a quartz crystal oscillator
- We can observe the frequency on the XTAL2 pin.



XTAL Connection to an External Clock Source

- Using a TTL oscillator
- XTAL2 is unconnected.



Machine cycle

Find the machine cycle for

(a) XTAL = 11.0592 MHz

(b) XTAL = 16 MHz.

Solution:

(a) 11.0592 MHz / 12 = 921.6 kHz;

machine cycle = 1 / 921.6 kHz = 1.085 μs

(b) 16 MHz / 12 = 1.333 MHz;

machine cycle = 1 / 1.333 MHz = 0.75 μs

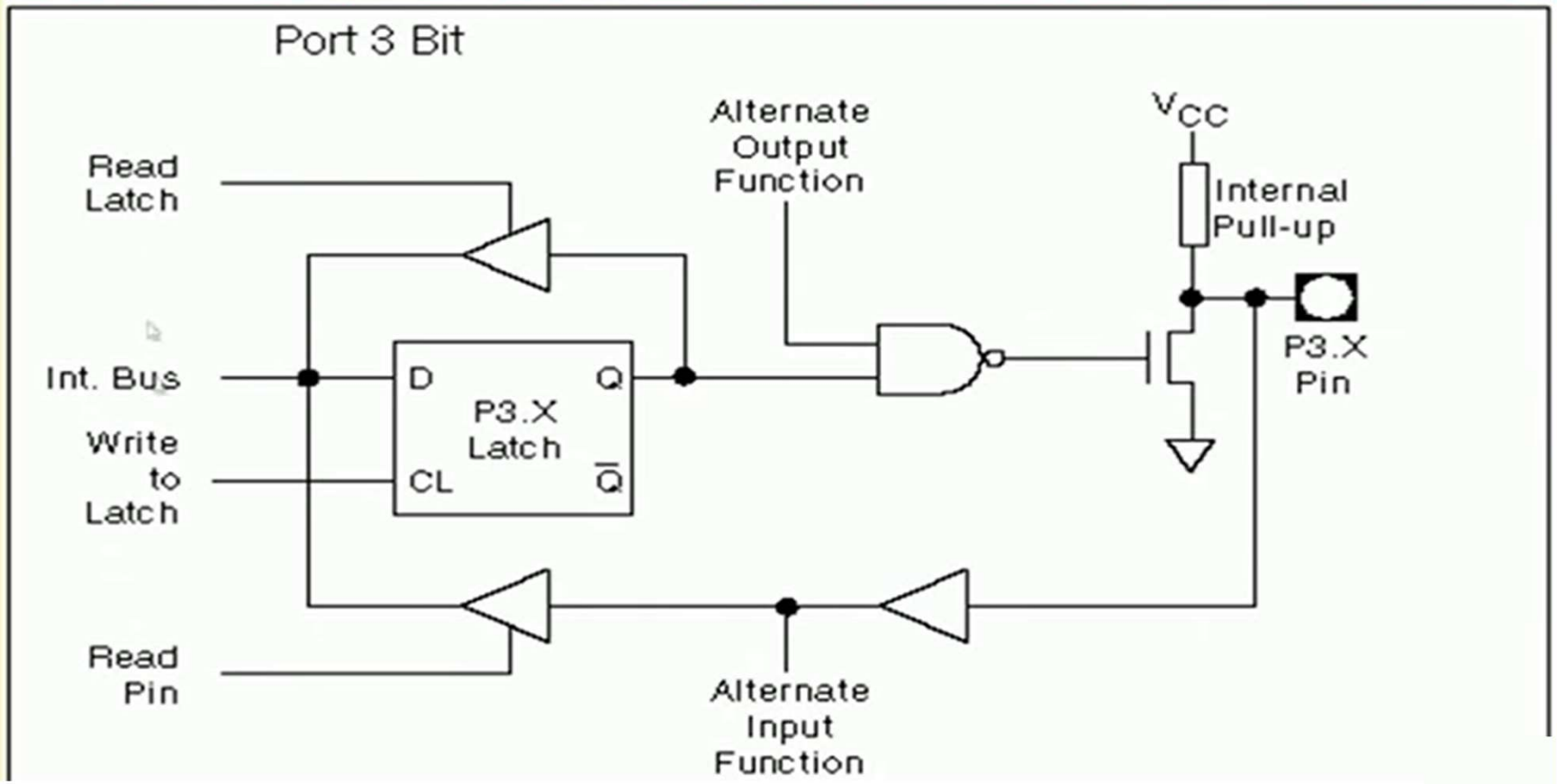
Machine cycle

- Find the machine cycle for
- (a) XTAL = 11.0592 MHz
- (b) XTAL = 16 MHz.

- **Solution:**

- (a) 11.0592 MHz / 12 = 921.6 kHz;
- machine cycle = 1 / 921.6 kHz = 1.085 μs
- (b) 16 MHz / 12 = 1.333 MHz;
- machine cycle = 1 / 1.333 MHz = 0.75 μs

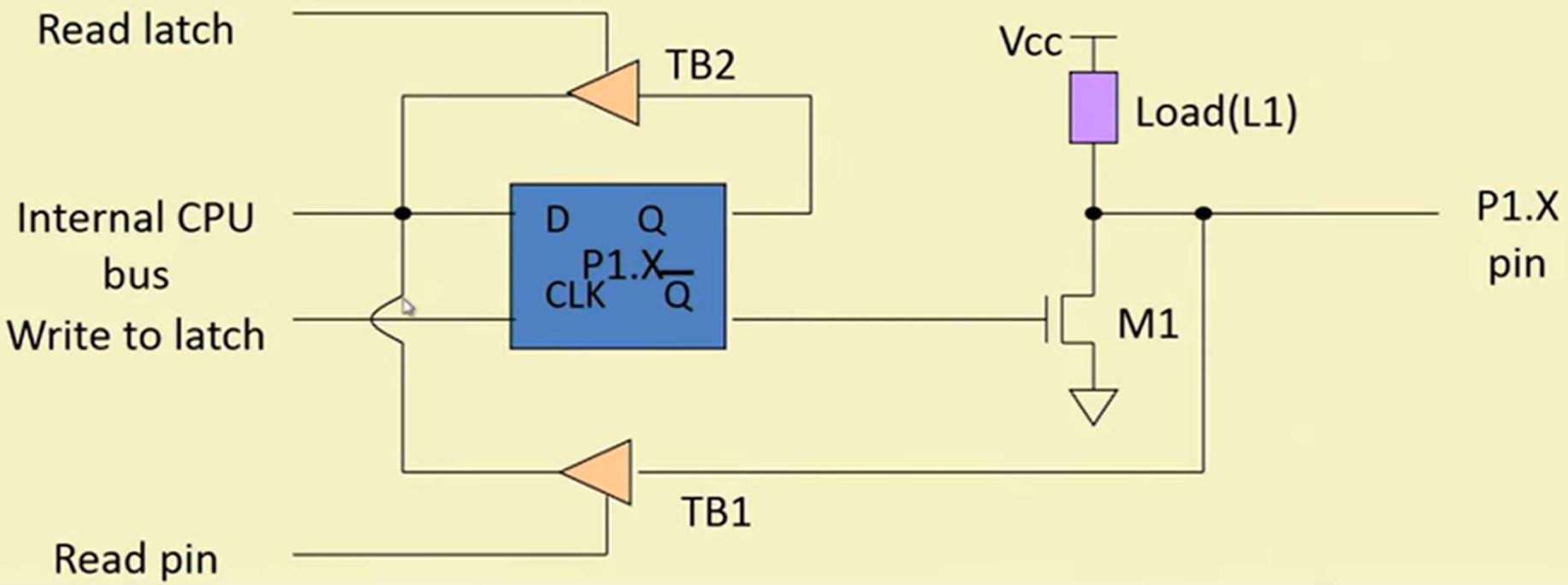
8051 Port 3 Bit Latches and I/O Buffers

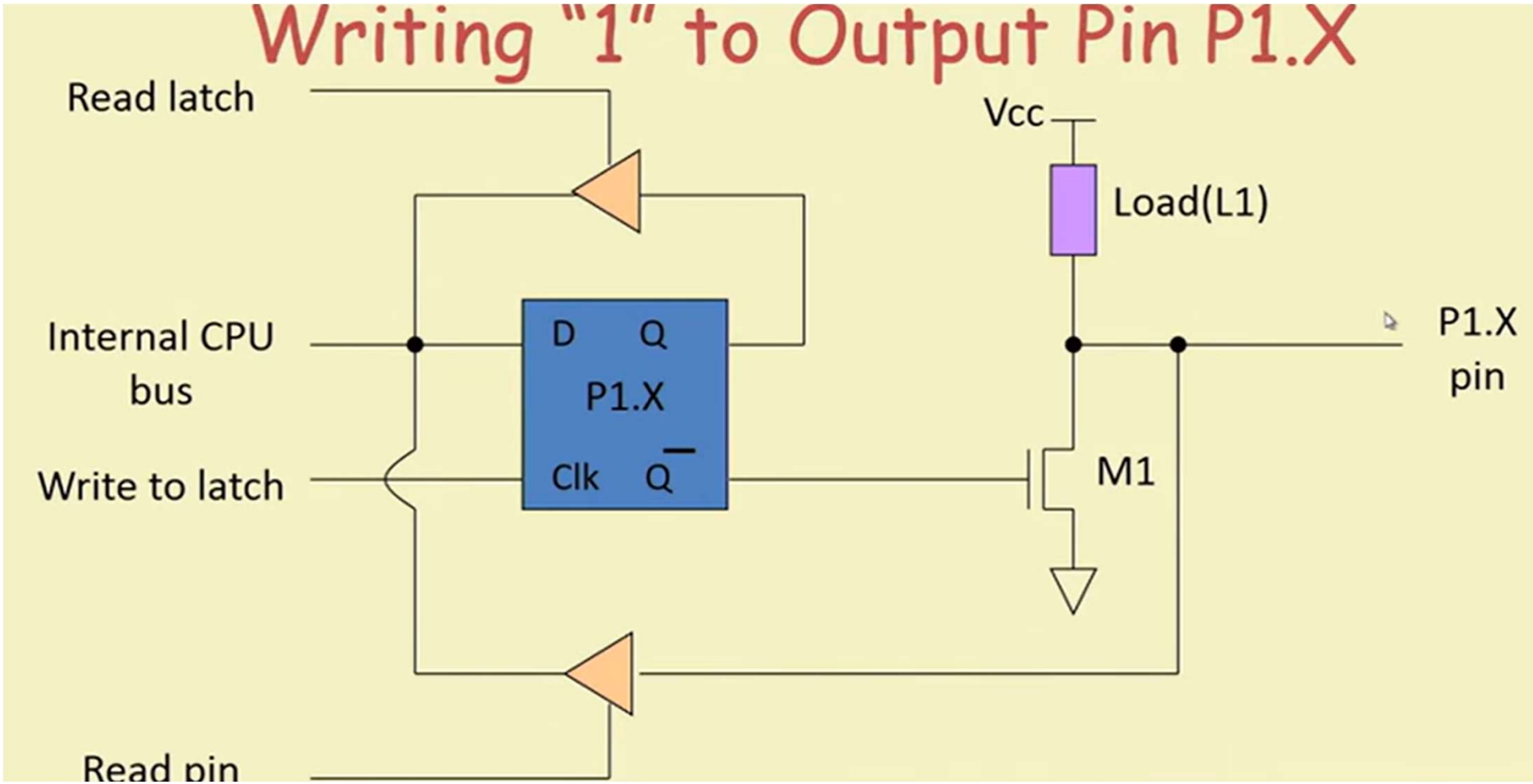


Hardware Structure of I/O Pin

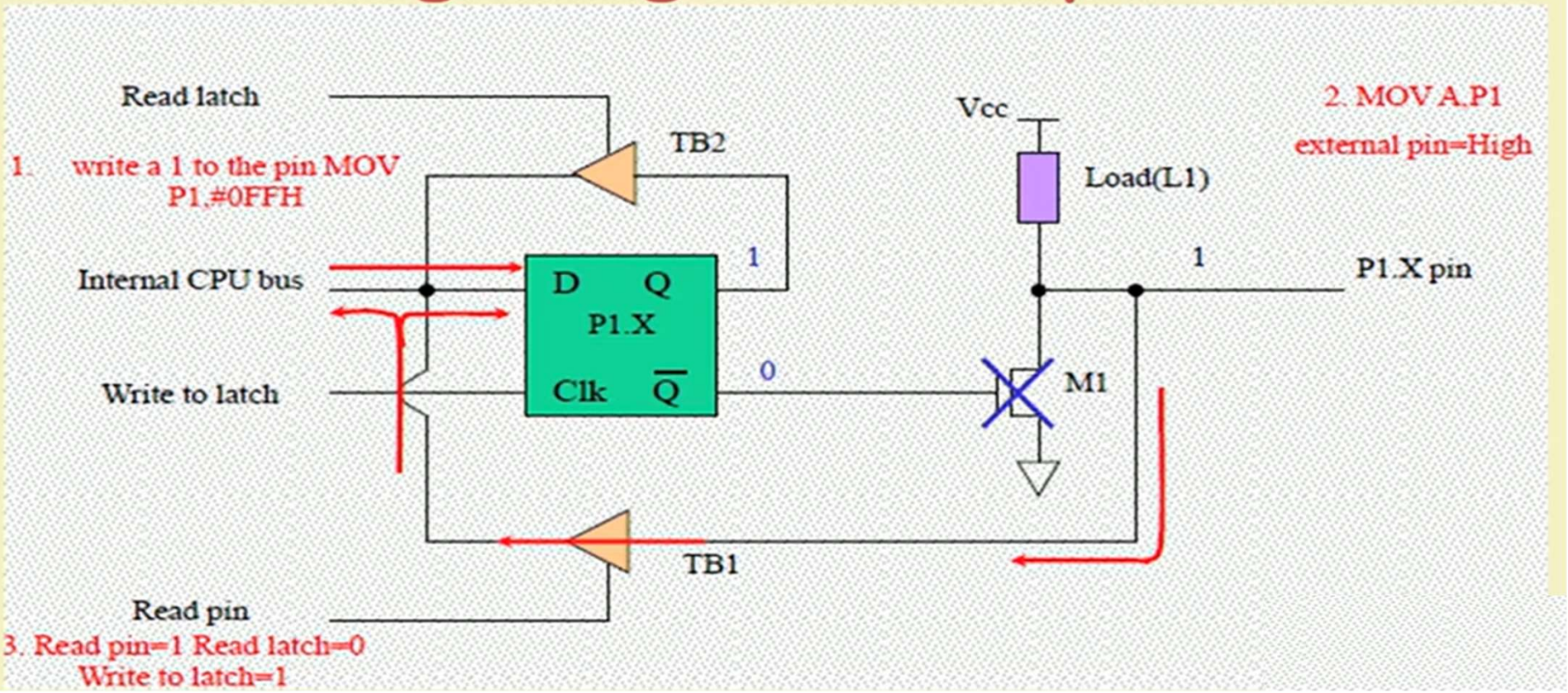
- Each pin of I/O ports
  - Internally connected to CPU bus
  - A D latch store the value of this pin
    - Write to latch＝1：write data into the D latch
  - 2 Tri-state buffer：
    - TB1: controlled by “Read pin”
      - Read pin＝1：really read the data present at the pin
    - TB2: controlled by “Read latch”
      - Read latch＝1：read value from internal latch
  - A transistor M1 gate
    - Gate=0: open
    - Gate=1: close

Hardware Structure of I/O Pin

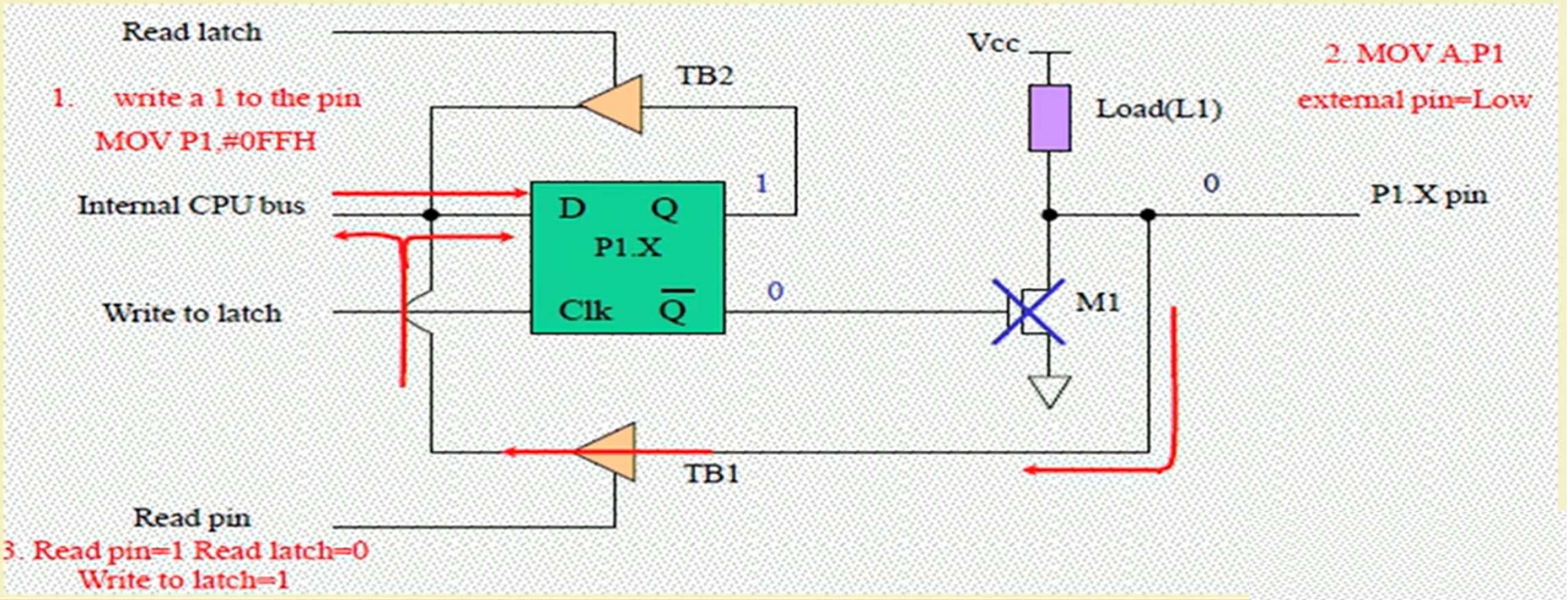



Writing "1" to Output Pin P1.X
Read latch
Vcc
Load(L1)
Internal CPU bus
D
Q
P1.X
P1.X pin
Write to latch
Clk
Q
M1
Read pin

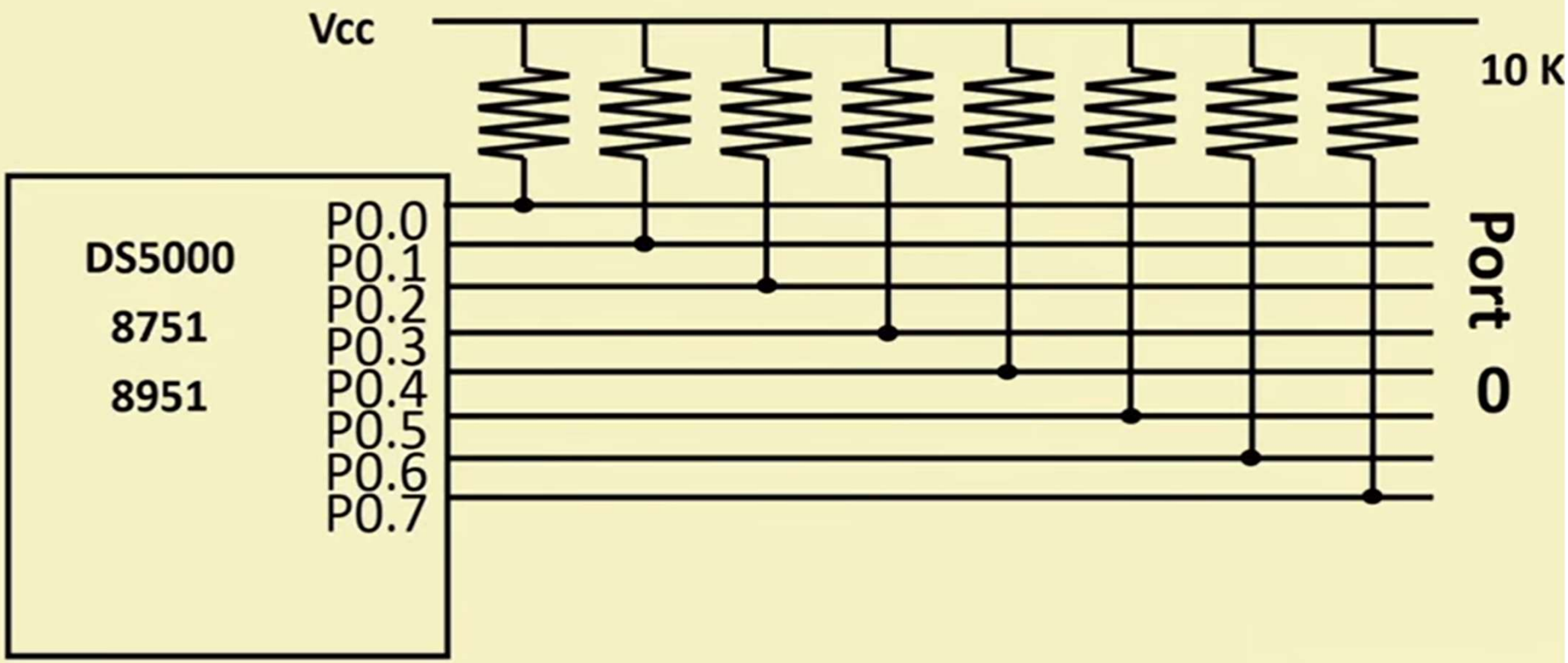
Reading "High" at Input Pin



Reading "Low" at Input Pin

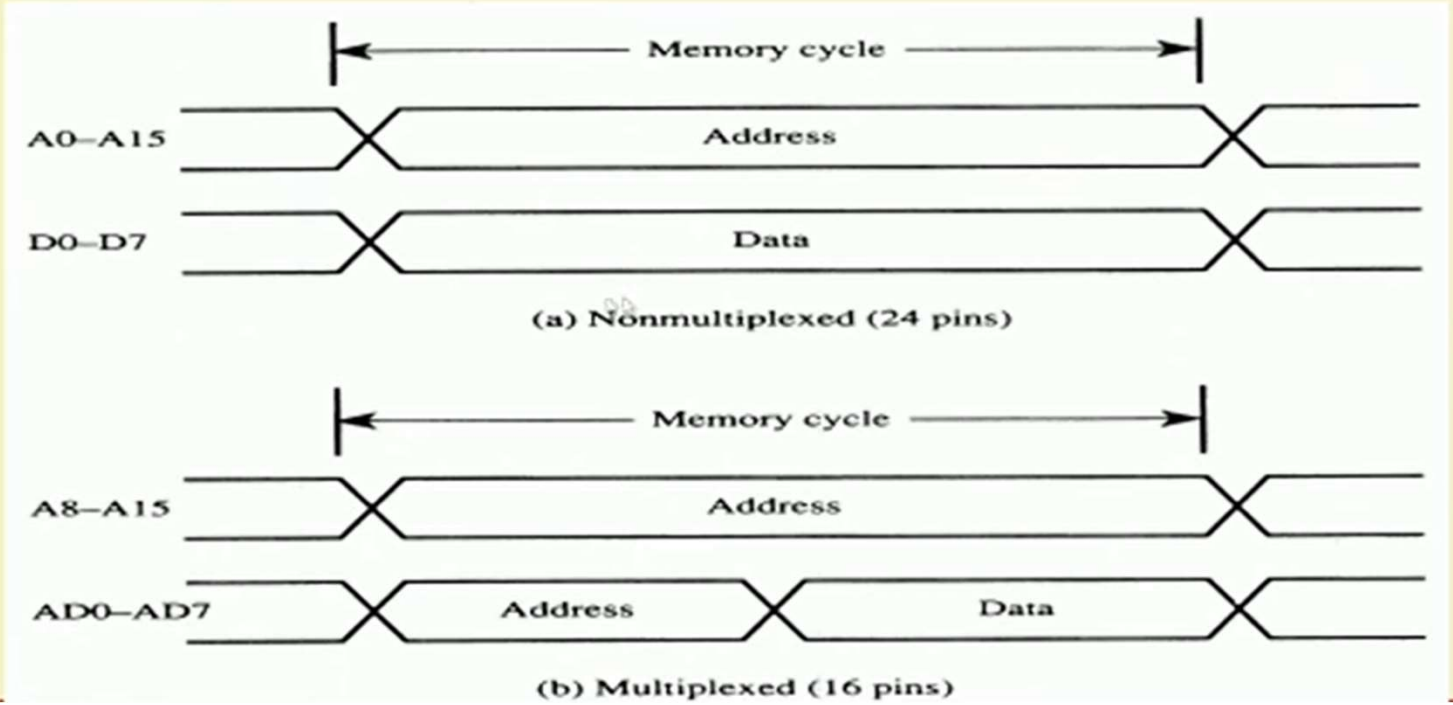


Port 0 with Pull-Up Resistors



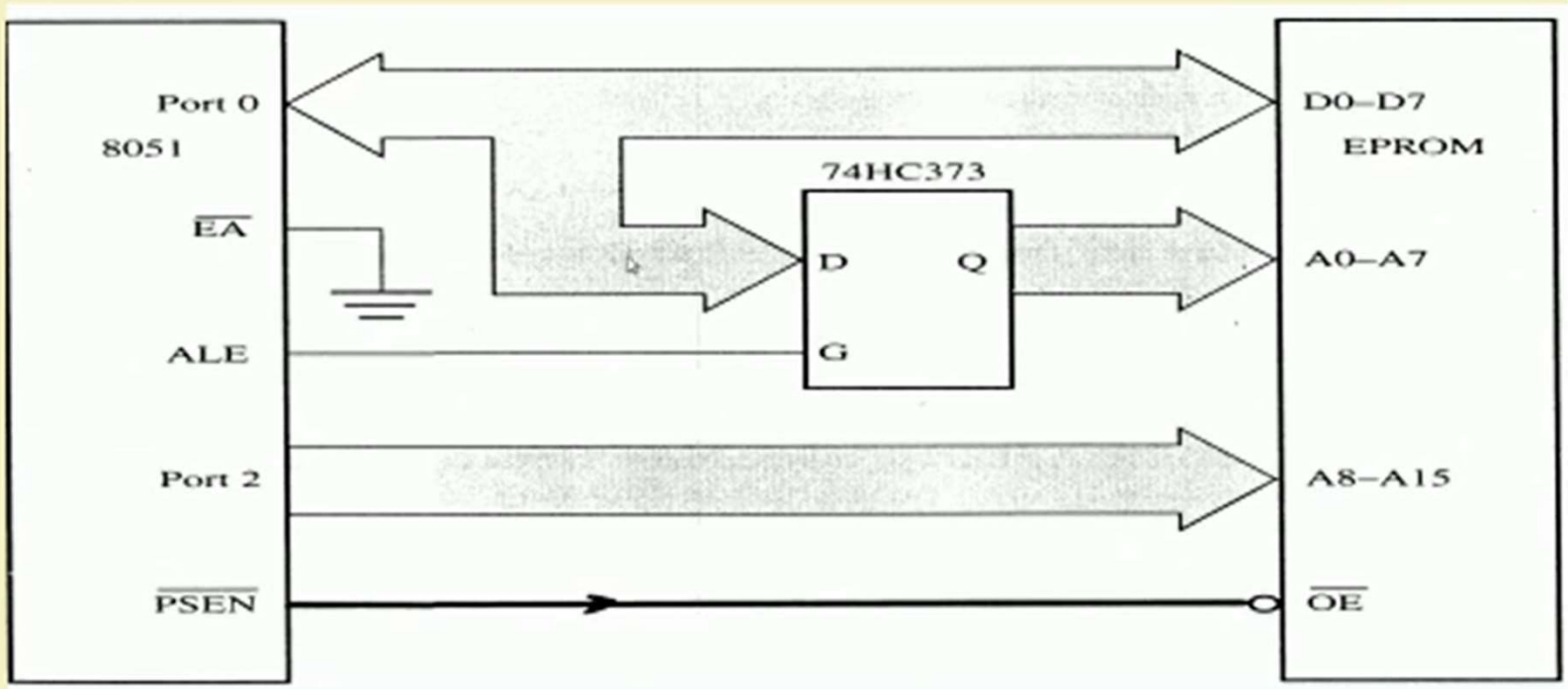
Address Multiplexing for External Memory

Multiplexing the address (low-byte) and data bus

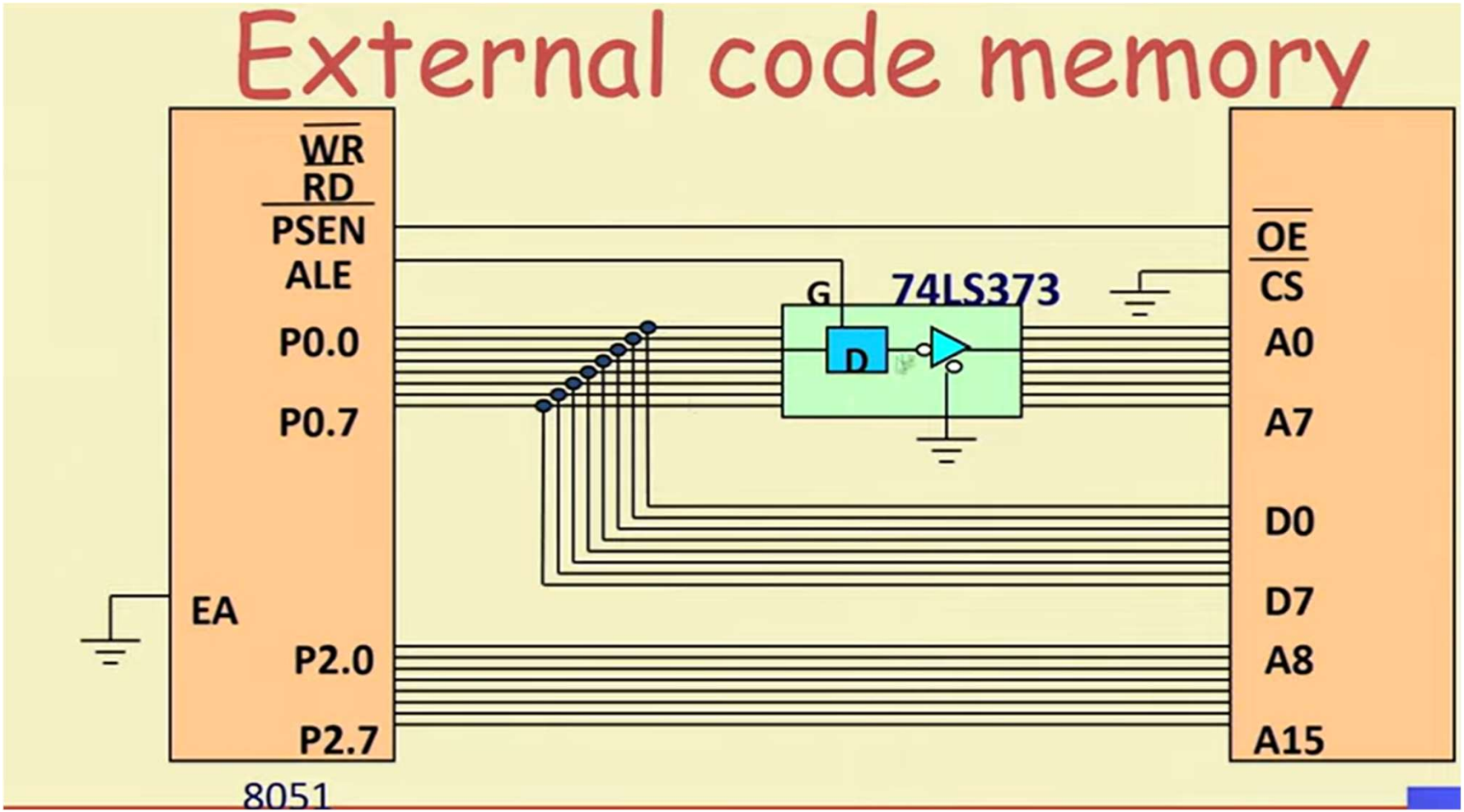


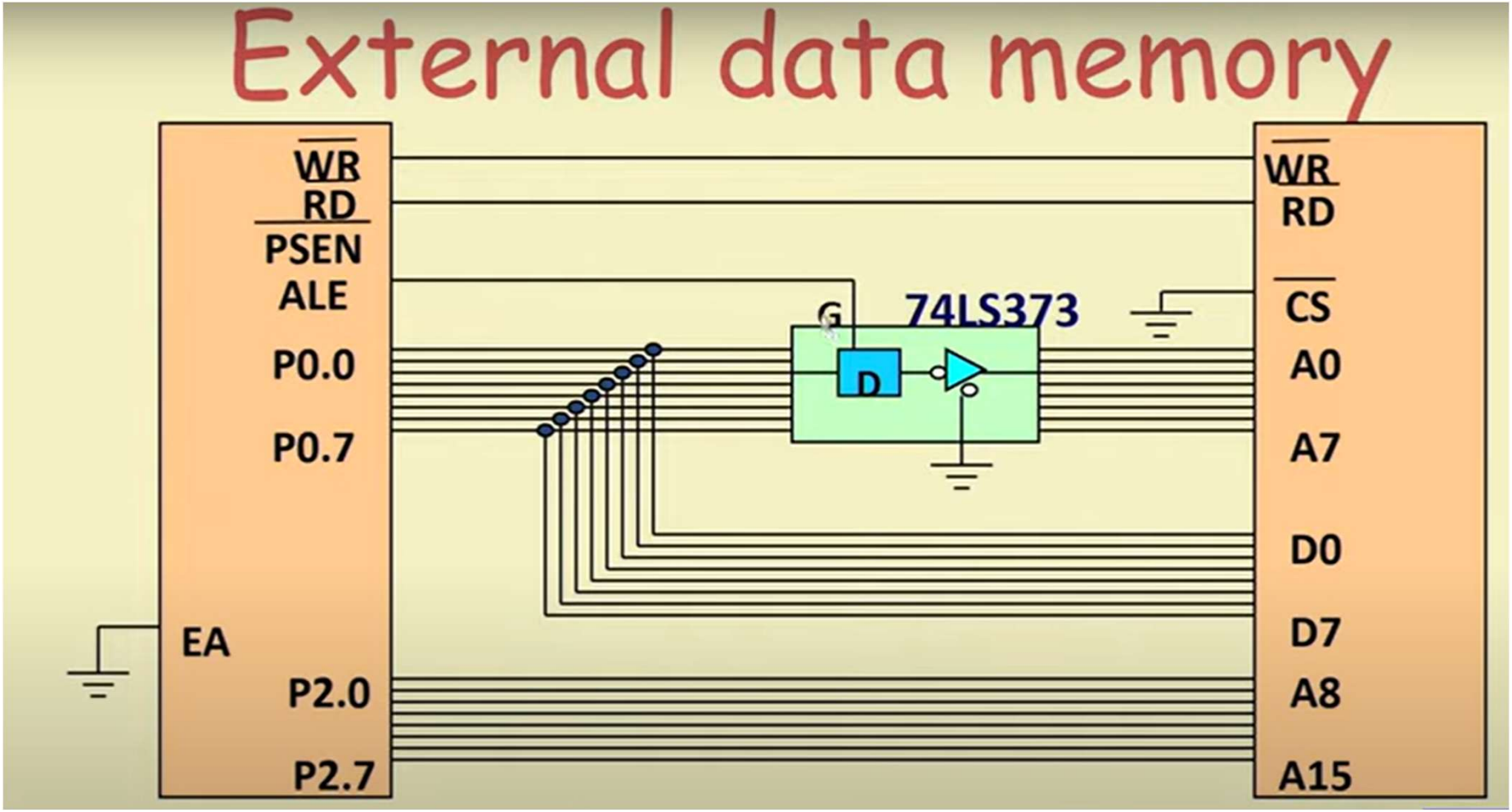
Address Multiplexing for External Memory

Accessing external code memory

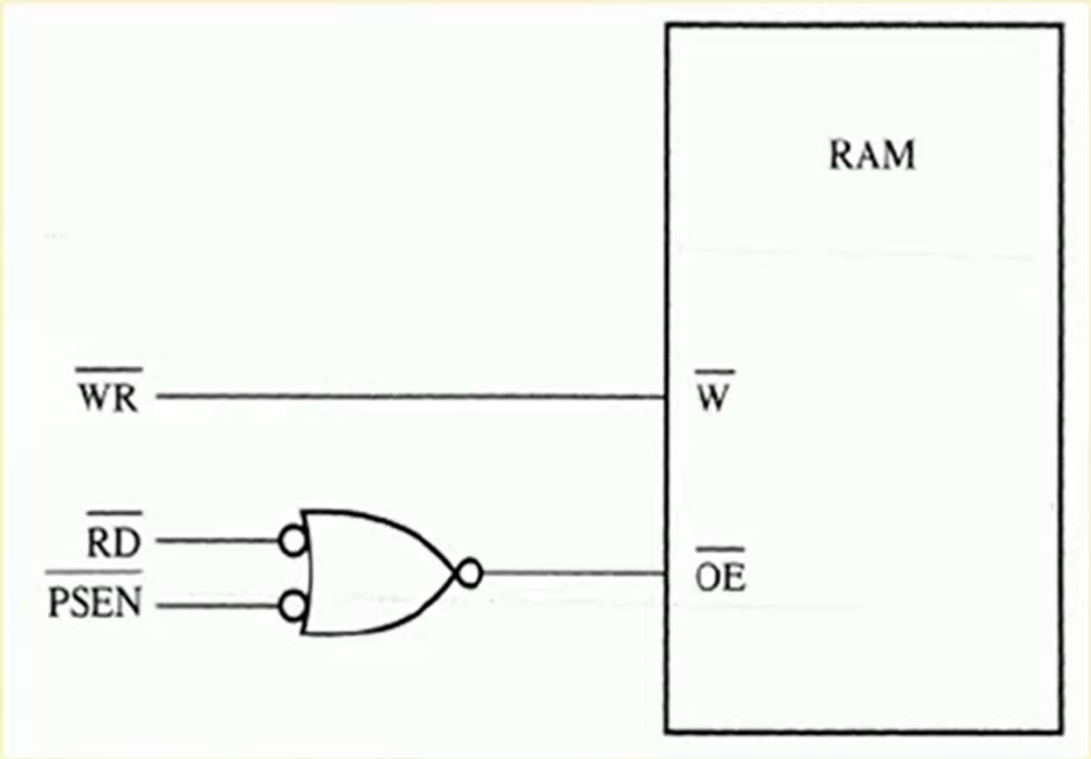


External code memory

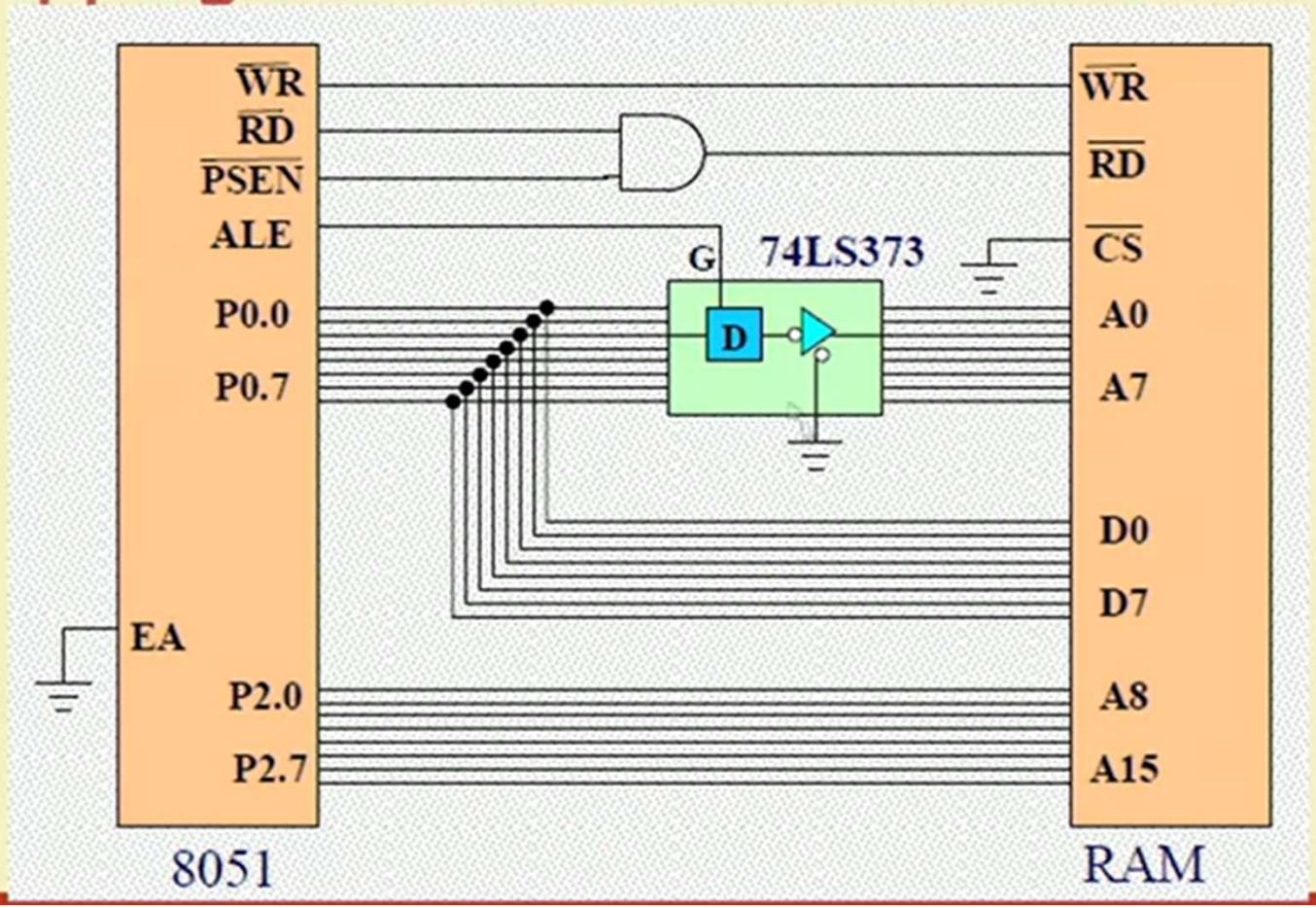



External data memory
WR
RD
PSEN
ALE
P0.0
P0.7
EA
P2.0
P2.7
G
74LS373
D
WR
RD
CS
A0
A7
D0
D7
A8
A15

Overlapping External Code and Data Spaces



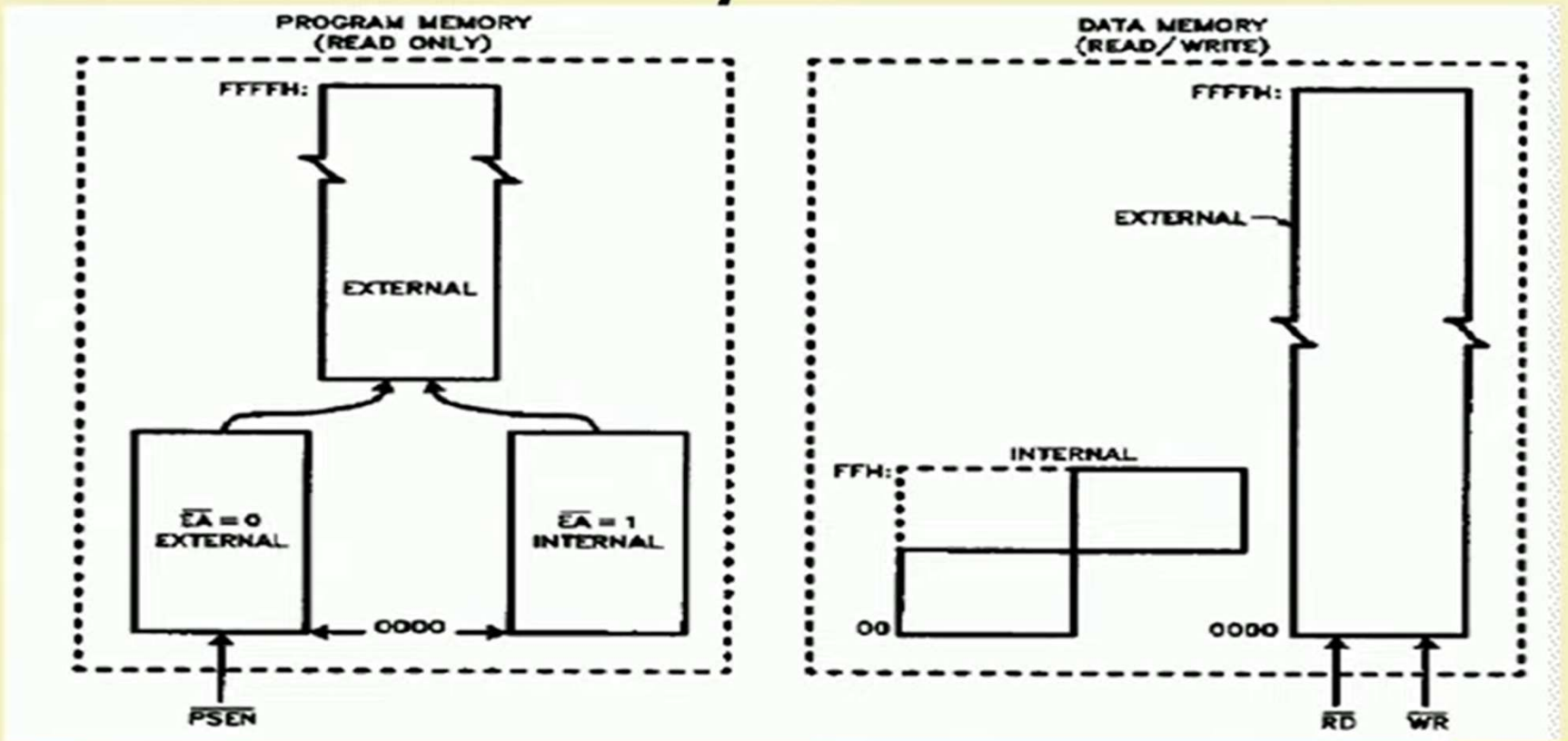
Overlapping External Code and Data Spaces



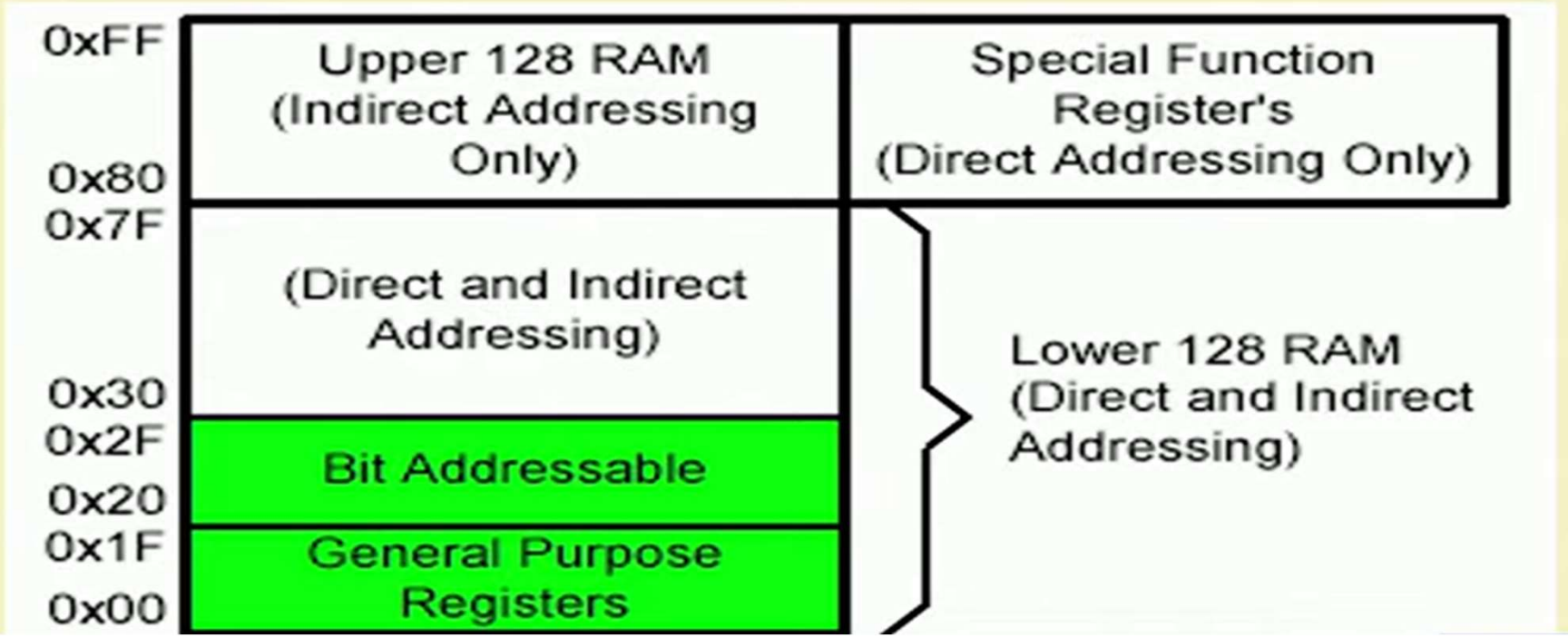
Overlapping External Code and Data Spaces

- Allows the RAM to be
  - written as data memory, and
  - read as data memory as well as **code memory**.
- This allows a program to be
  - downloaded from outside into the RAM as data, and
  - executed from RAM as code.

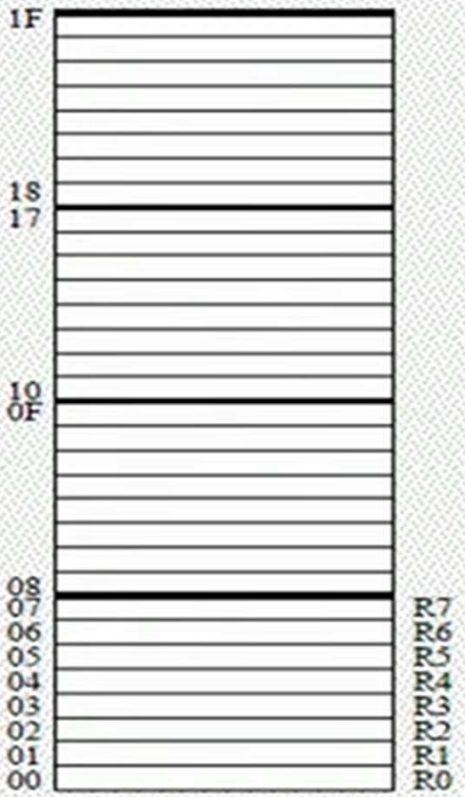
Memory Structure



On-Chip Memory
Internal RAM



Registers



Bank 3

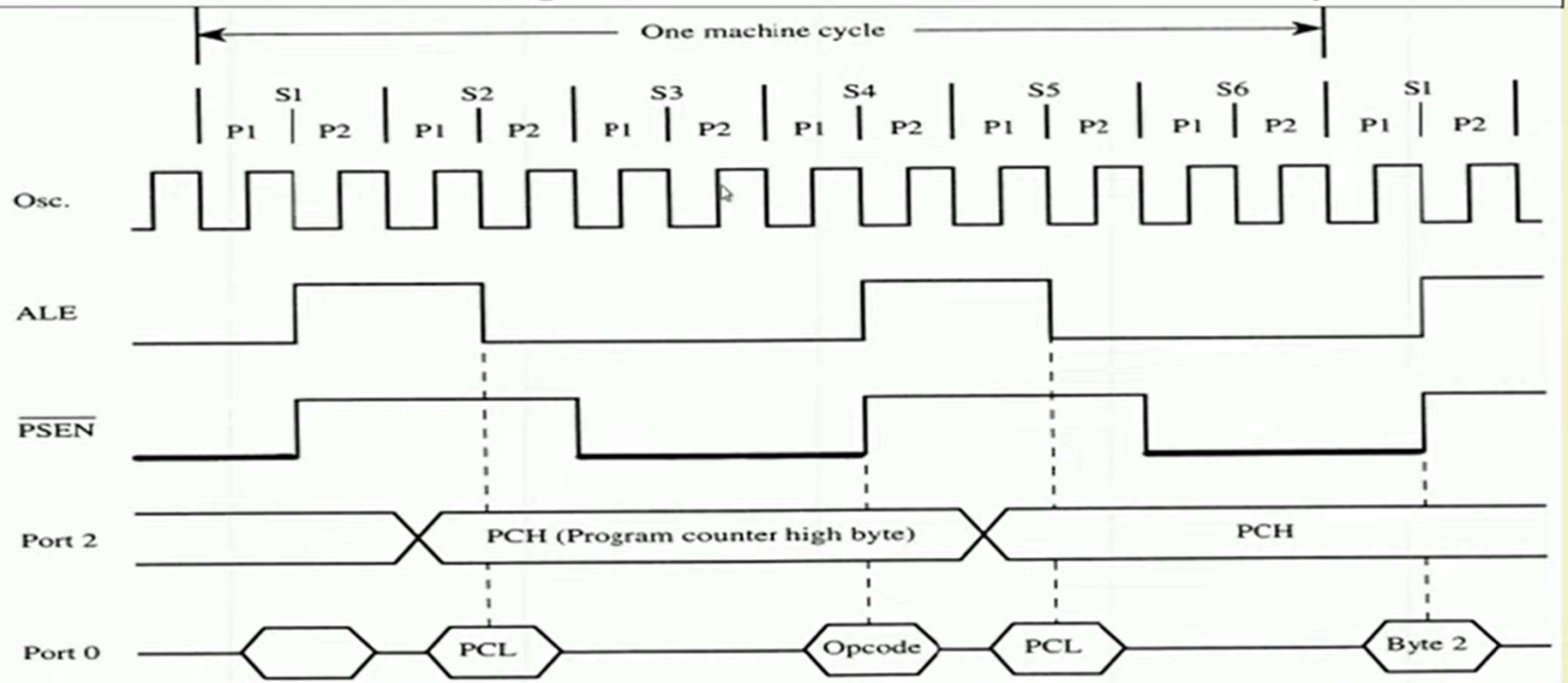
Bank 2

Bank 1

Bank 0

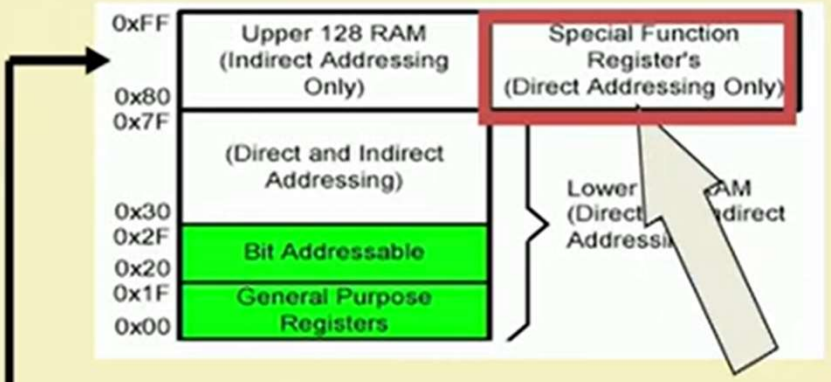
Four Register Banks
Each bank has R0-R7
Selectable by psw.2,3

Read Timing for External Code Memory



Special Function Registers

- DATA registers
- CONTROL registers
  - Timers
  - Serial ports
  - Interrupt system
  - Analog to Digital converter
  - Digital to Analog converter
  - Etc.



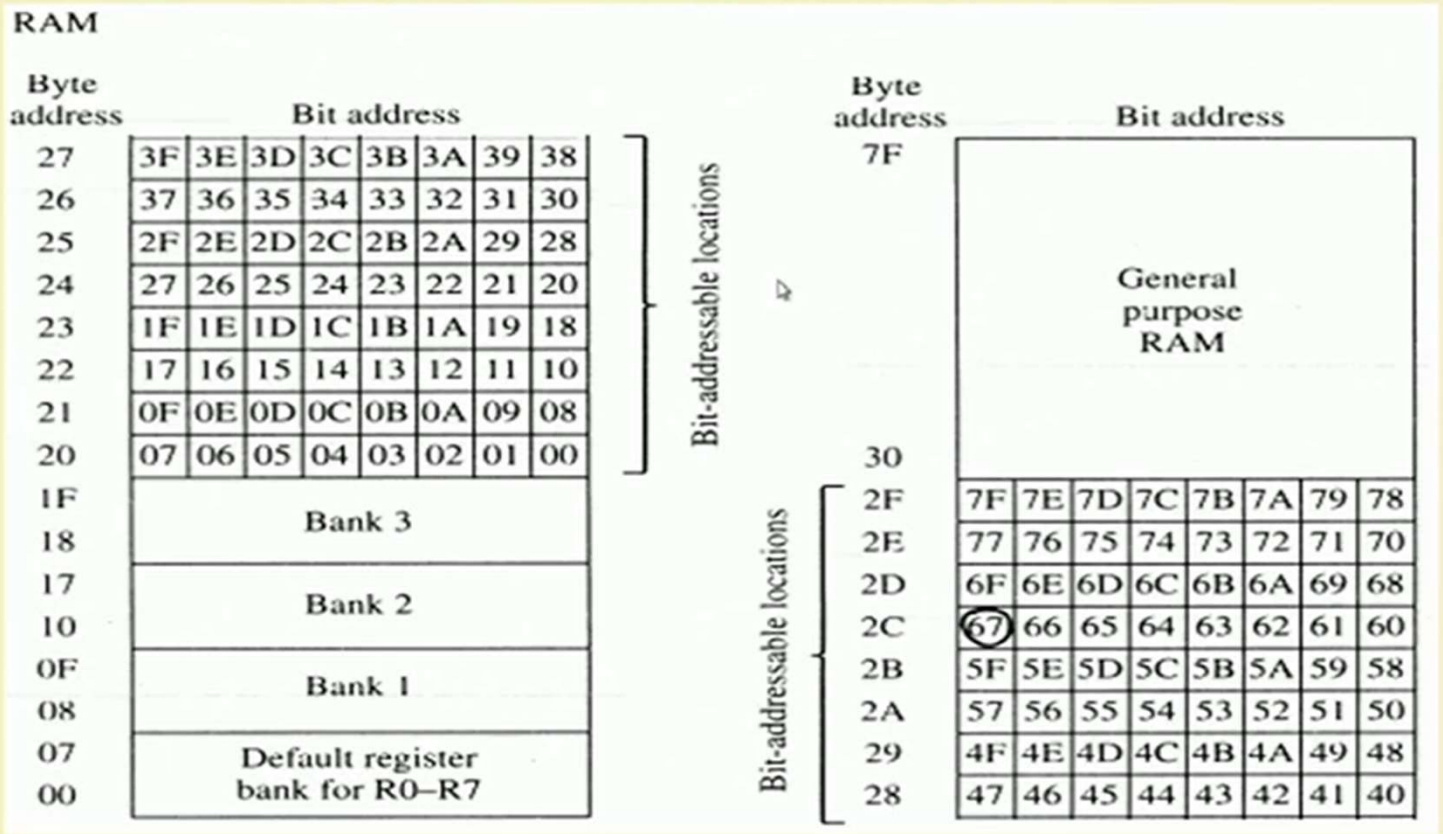
Addresses 80h – FFh

Direct Addressing used to access SPRs

Bit Addressable RAM

Summary of the 8051 on-chip data memory

(RAM)



Bit Addressable RAM

Summary of the 8051 on-chip data memory

(Special Function Registers)

Byte address	Bit address								
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	not bit addressable								TH1
8C	not bit addressable								TH0
8B	not bit addressable								TL1
8A	not bit addressable								TL0
89	not bit addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	not bit addressable								PCON
83	not bit addressable								DPH
82	not bit addressable								DPL
81	not bit addressable								SP
80	87	86	85	84	83	82	81	80	P0

Byte address	Bit address								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	–	D0	PSW
B8	–	–	–	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	–	–	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	not bit addressable								SBUF

Register Banks

- Active bank selected by PSW [**RS1**,**RS0**] bit
- Permits fast "context switching" in interrupt service routines (ISR).

PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

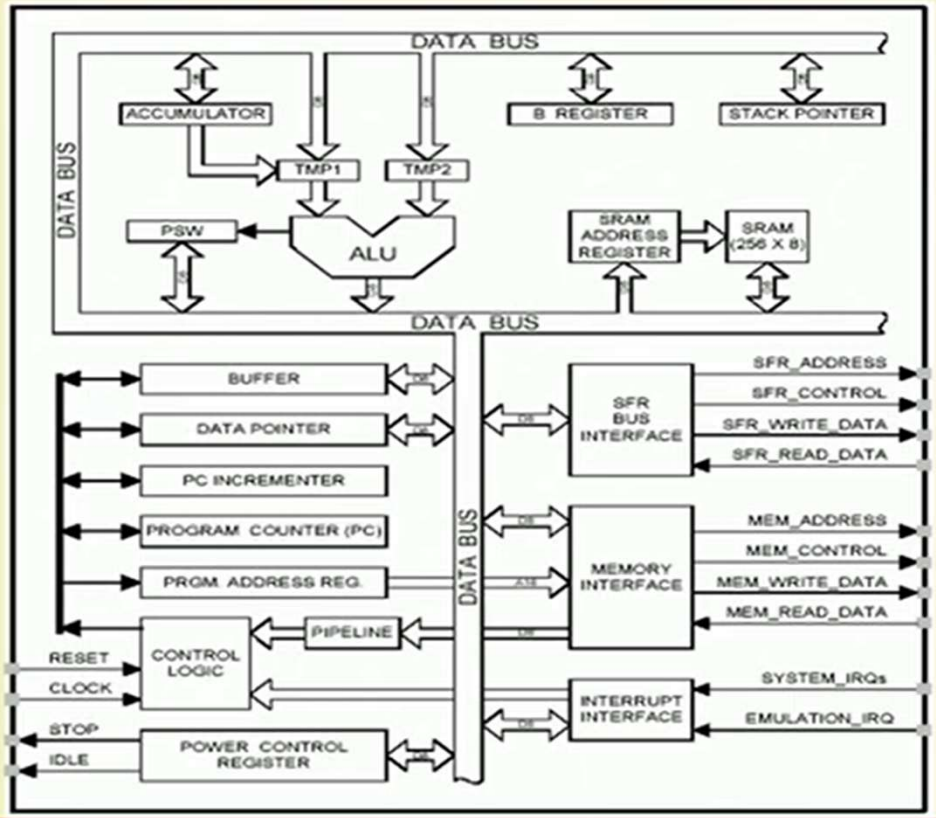
CY	AC	F0	RS1	RS0	OV	—	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

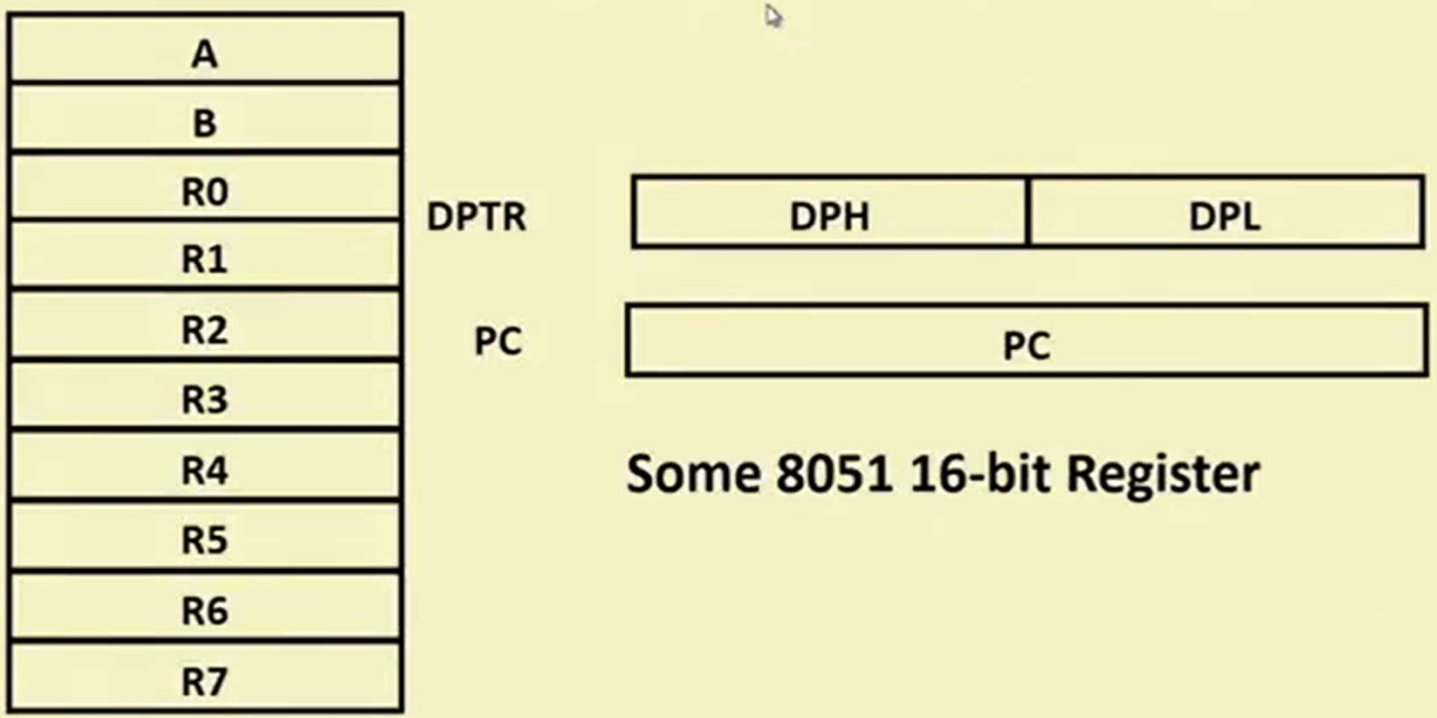
8051 CPU Registers



- A (Accumulator)
- B
- PSW (Program Status Word)
- SP (Stack Pointer)
- PC (Program Counter)
- DPTR (Data Pointer)

Used in assembler instructions

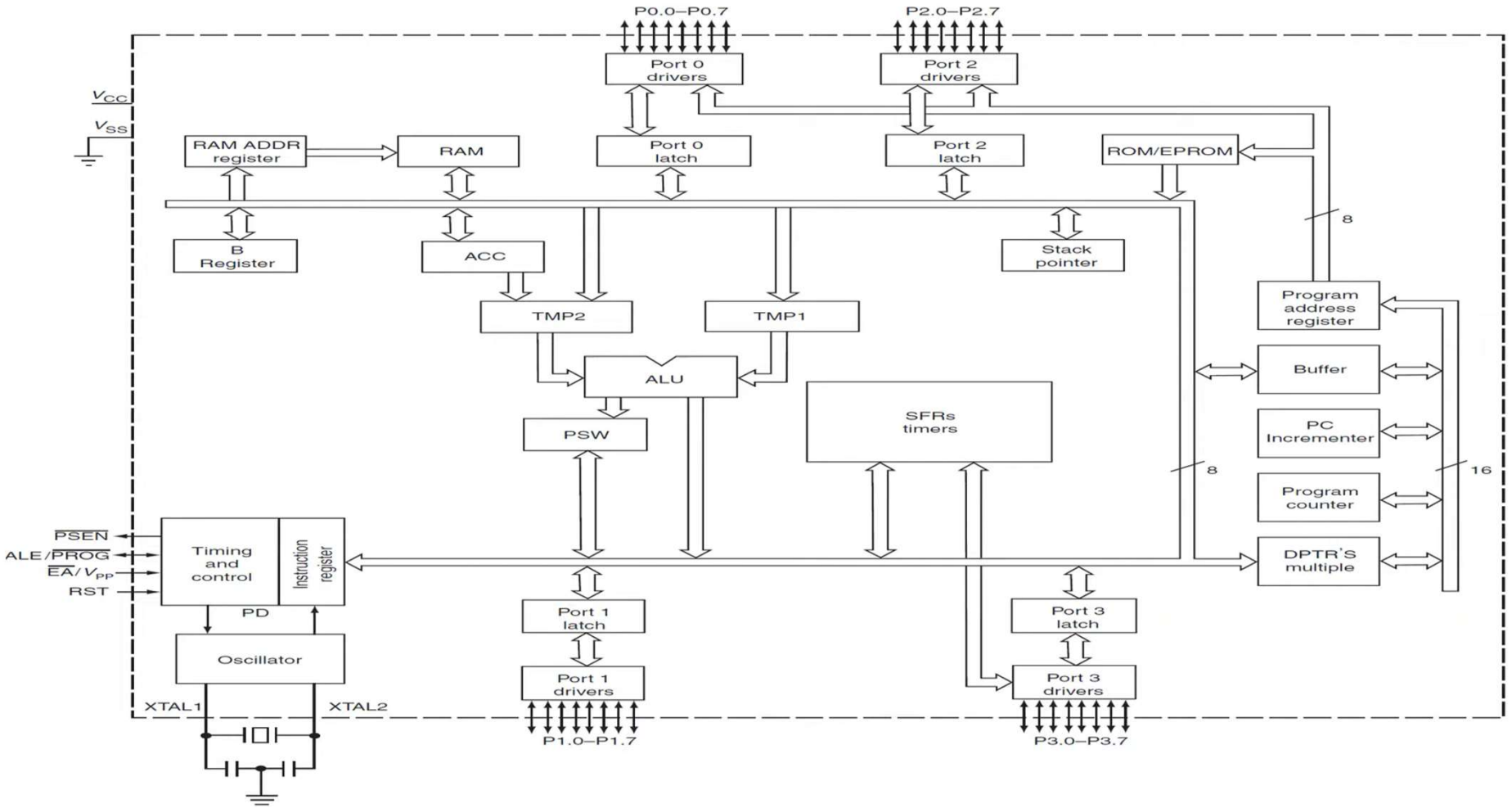
Registers



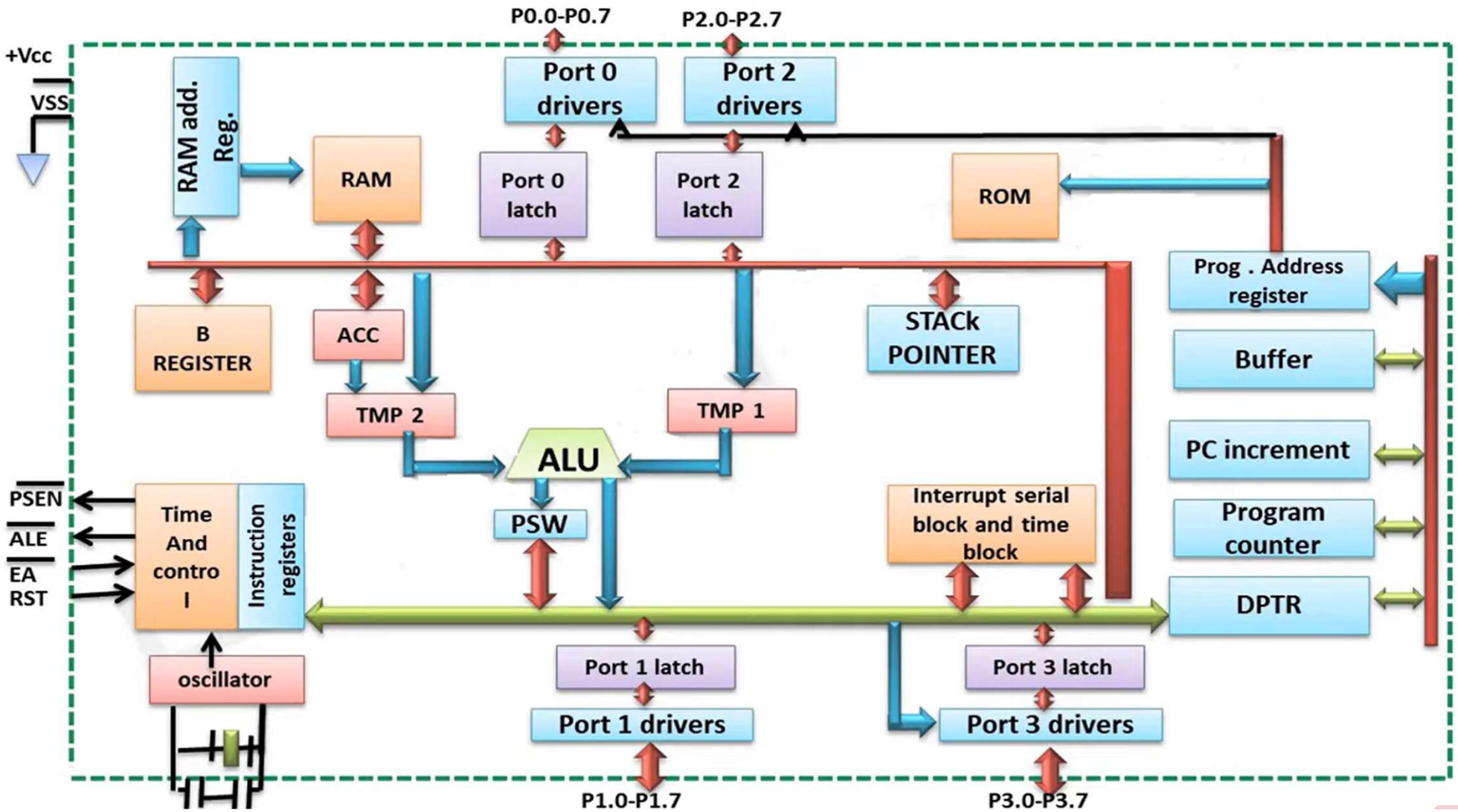
Some 8051 16-bit Register

Some 8-bit Registers of the 8051

Microcontroller 8051- Architecture



Microcontroller 8051- Architecture



FLAG BITS AND PSW REGISTER

Program Status Word (cont')

CY	AC	F0	RS1	RS0	OV	--	P

A carry from D3 to D4

Carry out from the d7 bit

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
--	PSW.5	Available to the user for general purpose
RS1	PSW.4	Register Bank selector bit 1.
RS0	PSW.3	Register Bank selector bit 0.
OV	PSW.2	Overflow flag.
--	PSW.1	User definable bit.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of 1 bits in the accumulator.

Reflect the number of 1s in register A

The result of signed number operation is too large, causing the high-order bit to overflow into the sign bit

RS1	RS0	Register Bank	Address
0	0	0	00H – 07H
0	1	1	08H – 0FH
1	0	2	10H – 17H
1	1	3	18H – 1FH

FLAG BITS AND PSW REGISTER

ADD Instruction And PSW

Instructions that affect flag bits

Instruction	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RPC	X		
PLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

FLAG BITS AND PSW REGISTER

ADD Instruction And PSW (cont')

- **The flag bits affected by the ADD instruction are CY, P, AC, and OV**

Example 2-2

Show the status of the CY, AC and P flag after the addition of 38H and 2FH in the following instructions.

```
MOV A, #38H
ADD A, #2FH ;after the addition A=67H, CY=0
```

Solution:

38	00111000
+2F	00101111
67	01100111

CY = 0 since there is no carry beyond the D7 bit

AC = 1 since there is a carry from the D3 to the D4 bi

P = 1 since the accumulator has an odd number of 1s (it has five 1s)

FLAG BITS AND PSW REGISTER

ADD Instruction And PSW (cont')

Example 2-3

Show the status of the CY, AC and P flag after the addition of 9CH and 64H in the following instructions.

```
MOV A, #9CH
ADD A, #64H  ;after the addition A=00H, CY=1
```

Solution:

9C	10011100
+ 64	01100100
100	00000000

CY = 1 since there is a carry beyond the D7 bit

AC = 1 since there is a carry from the D3 to the D4 bi

P = 0 since the accumulator has an even number of 1s (it has zero 1s)

FLAG BITS AND PSW REGISTER

ADD Instruction And PSW (cont')

Example 2-4

Show the status of the CY, AC and P flag after the addition of 88H and 93H in the following instructions.

```
MOV A, #88H
ADD A, #93H  ;after the addition A=1BH, CY=1
```

Solution:

88	10001000
+ 93	10010011
11B	00011011

CY = 1 since there is a carry beyond the D7 bit

AC = 0 since there is no carry from the D3 to the D4 bi

P = 0 since the accumulator has an even number of 1s (it has four 1s)

The 8051 Assembly Language

8051 ASSEMBLY PROGRAMMING

Structure of Assembly Language

- **Assembly language instruction includes**
  - **a mnemonic** (abbreviation easy to remember)
    - the commands to the CPU, telling it what those to do with those items
  - **optionally followed by one or two operands**
    - the data items being manipulated
- **A given Assembly language program is a series of statements, or lines**
  - **Assembly language instructions**
    - Tell the CPU what to do
  - **Directives (or pseudo-instructions)**
    - Give directions to the assembler

8051 ASSEMBLY PROGRAMMING

Structure of Assembly Language

- **An Assembly language instruction consists of four fields:**

`[label:] Mnemonic [operands] [;comment]`

```
   ORG   0H              ;start(origin) at location
   0
   MOV   R5,  #25H       ;load 25H into R5
   MOV   R7,  #34H       ;load 34H in
   MOV   A,  #0          ;load 0 into
   ADD   A,  R5          ;add content
                         ;now A = A +
   ADD   A,  R7          ;add contents of R7 to A
                         ;now A = A + R7
   ADD   A,  #12H        ;add to A value 12H
                         ;now A = A + 12H
HERE:  SJMP  HERE        ;stay in this loop
   END                   ;en
```

Mnemonics produce opcodes

Directives do not generate any machine code and are used only by the assembler

The label field allows the program to refer to a line of code by name

Comments may be at the end of a line or on a line by themselves
The assembler ignores comments

ASSEMBLING AND RUNNING AN 8051 PROGRAM

- **The step of Assembly language program are outlines as follows:**

1) First we use an editor to type a program, many excellent editors or word processors are available that can be used to create and/or edit the program
   - Notice that the editor must be able to produce an ASCII file
   - For many assemblers, the file names follow the usual DOS conventions, but the source file has the extension "asm" or "src", depending on which assembly you are using

ASSEMBLING AND RUNNING AN 8051 PROGRAM (cont')

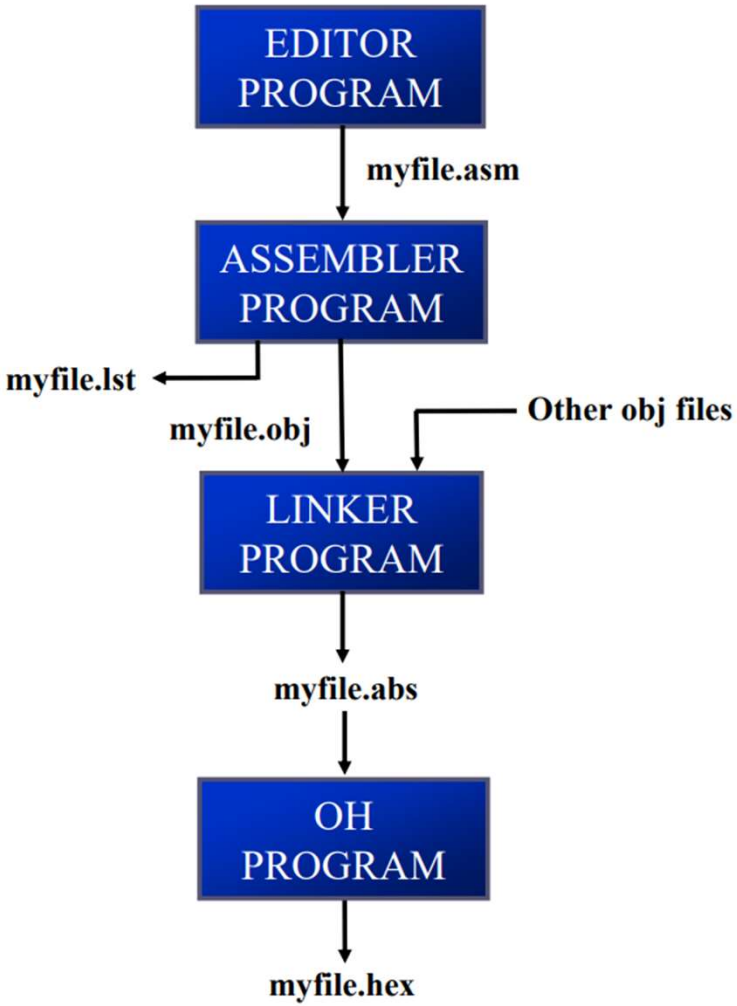
2) **The "asm" source file containing the program code created in step 1 is fed to an 8051 assembler**
   - The assembler converts the instructions into machine code
   - The assembler will produce an object file and a list file
   - The extension for the object file is "obj" while the extension for the list file is "lst"
3) **Assembler require a third step called *linking***
   - The linker program takes one or more object code files and produce an absolute object file with the extension "abs"
   - This abs file is used by 8051 trainers that have a monitor program

ASSEMBLING AND RUNNING AN 8051 PROGRAM (cont')

4) **Next the "abs" file is fed into a program called "OH" (object to hex converter) which creates a file with extension "hex" that is ready to burn into ROM**
   - **This program comes with all 8051 assemblers**
   - **Recent Windows-based assemblers combine step 2 through 4 into one step**

ASSEMBLING AND RUNNING AN 8051 PROGRAM

Steps to Create a Program



PROGRAM COUNTER AND ROM SPACE

Placing Code in ROM

- **Examine the list file and how the code is placed in ROM**

```
1 0000             ORG 0H              ;start (origin) at 0
2 0000  7D25       MOV R5,#25H         ;load 25H into R5
3 0002  7F34       MOV R7,#34H         ;load 34H into R7
4 0004  7400       MOV A,#0            ;load 0 into A
5 0006  2D         ADD A,R5            ;add contents of R5 to A
                                       ;now A = A + R5
6 0007  2F         ADD A,R7            ;add contents of R7 to A
                                       ;now A = A + R7
7 0008  2412       ADD A,#12H          ;add to A value 12H
                                       ;now A = A + 12H
8 000A  80EF       HERE: SJMP HERE     ;stay in this loop
9 000C             END                 ;end of asm source file
```

ROM Address	Machine Language	Assembly Language
0000	7D25	MOV R5, #25H
0002	7F34	MOV R7, #34H
0004	7400	MOV A, #0
0006	2D	ADD A, R5
0007	2F	ADD A, R7
0008	2412	ADD A, #12H
000A	80EF	HERE: SJMP HERE

PROGRAM COUNTER AND ROM SPACE

Placing Code in ROM (cont')

- After the program is burned into ROM, the opcode and operand are placed in ROM memory location starting at 0000

ROM contents

Address	Code
0000	7D
0001	25
0002	7F
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

Overview

- Data transfer instructions
- Addressing modes
- Data processing (arithmetic and logic)
- Program flow instructions

Data Transfer Instructions

- MOV dest, source dest ← source
- Stack instructions

```
PUSH byte ;increment stack pointer,
                ;move byte on stack
POP byte        ;move from stack to byte,
                      ;decrement stack pointer
```

- Exchange instructions

```
XCH a, byte     ;exchange accumulator and byte
XCHD a, byte    ;exchange low nibbles of
                ;accumulator and byte
```

Addressing Modes

Immediate Mode – specify data by its value

```
mov A, #0           ;put 0 in the accumulator

mov R4, #11h        ;put 11hex in the R4 register

mov B, #11          ;put 11 decimal in b register

mov DPTR,#7521h     ;put 7521 hex in DPTR
```

Addressing Modes

Immediate Mode – specify data by its value

```
mov A, #0            ;put 0 in the accumulator
                     ;A = 00000000
mov R4, #11h         ;put 11hex in the R4 register
                     ;R4 = 00010001
mov B, #11           ;put 11 decimal in b register
                     ;B = 00001011
mov DPTR,#7521h      ;put 7521 hex in DPTR
                     ;DPTR = 0111010100100001
```

Addressing Modes

Register Addressing – either source or destination is one of CPU register

```
MOV R0,A
MOV A,R7
ADD A,R4
ADD A,R7
MOV DPTR,#25F5H
MOV R5,DPL
MOV R,DPH
```

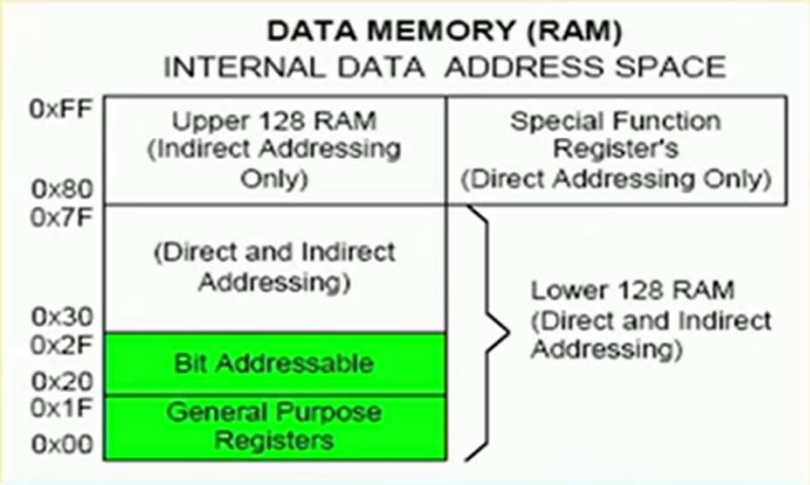
Note that MOV R4,R7 is incorrect

Addressing Modes

Direct Mode – specify data by its 8-bit address

Usually for 30h-7Fh of RAM

```
Mov a, 70h          ; copy contents of RAM at 70h to a
Mov R0,40h          ; copy contents of RAM at 40h to R0
Mov 56h,a           ; put contents of a at 56h
Mov 0D0h,a          ; put contents of a into PSW
```



Addressing Modes

Register Indirect – the address of the source or destination is specified in registers

Uses registers R0 or R1 for 8-bit address:

```
mov psw, #0              ; use register bank 0
mov r0, #0x3C
mov @r0, #3              ; memory at 3C gets #3
                         ; M[3C] ← 3
```

Uses DPTR register for 16-bit addresses:

```
mov dptr, #0x9000        ; dptr ← 9000h
movx a, @dptr            ; a ← M[9000]
```

Note that 9000 is an address in external memory

Addressing Modes

Register Indexed Mode – source or destination address is the sum of the base address and the accumulator(Index)

- Base address can be DPTR or PC

```
mov dptr, #4000h
mov a, #5
movc a, @a + dptr  ;a ← M[4005]
```

Addressing Modes

Register Indexed Mode continue

- Base address can be DPTR or PC

```
        ORG 1000h
        1000   mov a, #5
        1002   movc a, @a + PC    ;a ← M[1008]
PC  →   1003   Nop
```

- Table Lookup
- MOVC only can read internal code memory

Acc Register

- A register can be accessed by direct and register mode

- This 3 instruction has same function with different code

```
0703 E500          mov a,00h
0705 8500E0        mov acc,00h
0708 8500E0        mov 0e0h,00h
```

- Also this 3 instruction

```
070B E9            mov a,r1
070C 89E0          mov acc,r1
070E 89E0          mov 0e0h,r1
```

SFRs Address

- B – always direct mode - except in MUL & DIV

```
0703 8500F0          mov b,00h
0706 8500F0          mov 0f0h,00h

0709 8CF0            mov b,r4
070B 8CF0            mov 0f0h,r4
```

- P0~P3 – are direct address

```
0704 F580            mov p0,a
0706 F580            mov 80h,a
0708 859080          mov p0,p1
```

- Also other SFRs (pcon, tmod, psw,….)

SFRs Address

All SFRs such as

(ACC, B, PCON, TMOD, PSW, P0~P3, …)

are accessible by name and direct address

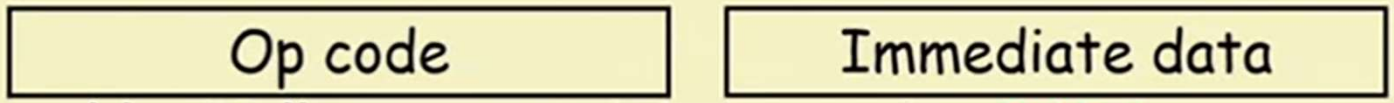
But

both of them

Must be coded as direct address

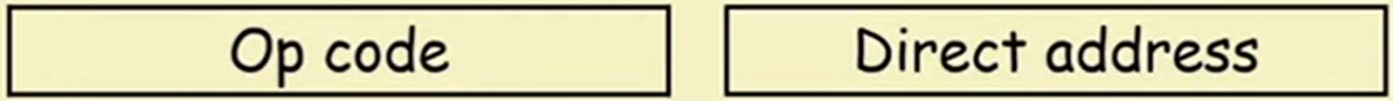
8051 Instruction Format

- immediate addressing




```
add a,#3dh        ;machine code=243d
```

- Direct addressing




```
mov r3,0E8h       ;machine code=ABE8
```

8051 Instruction Format

- Register addressing

Op code	n	n	n

```
070D E8      mov a,r0      ;E8 = 1110 1000
070E E9      mov a,r1      ;E9 = 1110 1001
070F EA      mov a,r2      ;EA = 1110 1010
0710 ED      mov a,r5      ;ED = 1110 1101
0711 EF      mov a,r7      ;Ef = 1110 1111
0712 2F      add a,r7
0713 F8      mov r0,a
0714 F9      mov r1,a
0715 FA      mov r2,a
0716 FD      mov r5,a
0717 FD      mov r5,a
```

8051 Instruction Format

- relative addressing

Op code	Relative address

```
here:  sjmp  here     ;machine code=80FE (FE=-2)
Range = (-128 ~ 127)
```

- Absolute addressing (limited in 2k current mem block)

A10-A8	Op code	A7-A0

```
0700
0700 E106
0702 00
0703 00
0704 00
0705 00
```

```
1          org 0700h
2          ajmp next    ;next=706h
3          nop
4          nop
5          nop
6          nop
7     next:
8          end
```

~~07FEh~~

8051 Instruction Format

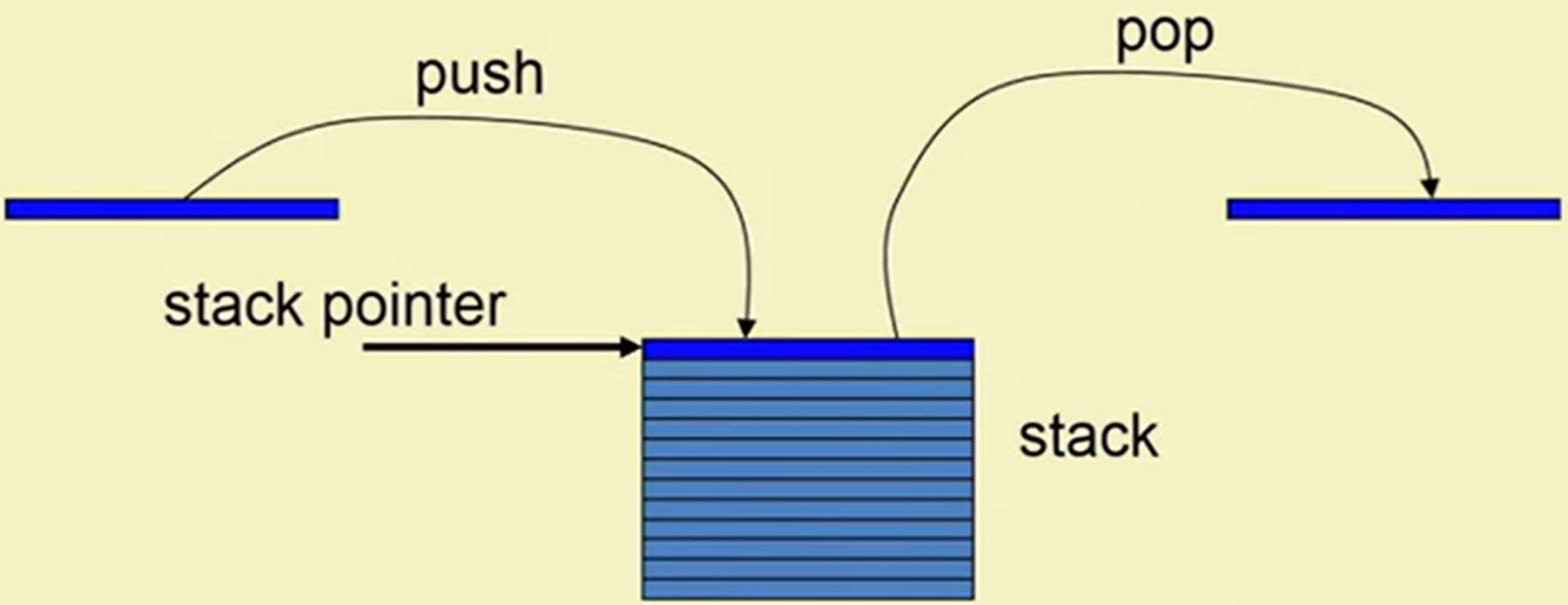
- Long distance address

Op code	A15-A8	A7-A0

Range = (0000h ~ FFFFh)

```
0700                    1           org 0700h
0700 020707             2           ljmp  next    ;next=0707h
0703 00                 3           nop
0704 00                 4           nop
0705 00                 5           nop
0706 00                 6           nop
                        7     next:
                        8           end
```

Stacks



Stack

- Stack-oriented data transfer
  - Only one operand (direct addressing)
  - SP is other operand – register indirect - implied
- Direct addressing mode must be used in Push and Pop

```
mov sp, #0x40   ; Initialize SP
push 0x55       ; SP ← SP+1, M[SP] ← M[55]
                ; M[41] ← M[55]
pop b           ; b ← M[55]
```

Note: can only specify RAM or SFRs (direct mode) to push or pop. Therefore, to push/pop the accumulator, must use acc, not a

Stack (push,pop)

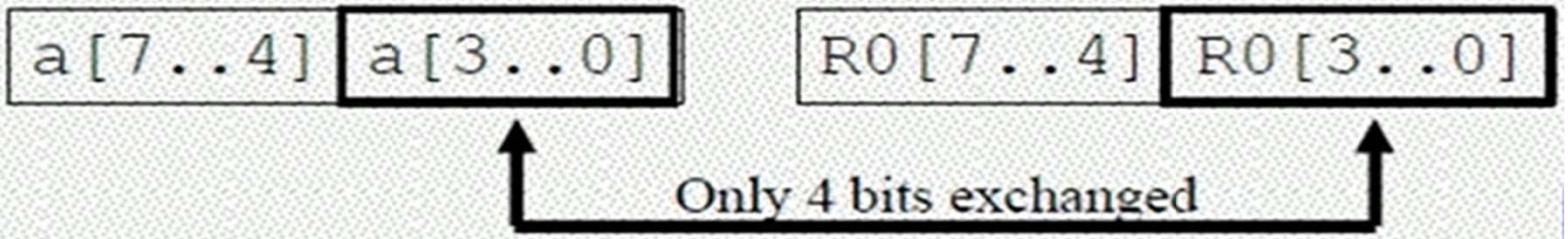
- Therefore

```
Push a      ;is invalid
Push r0     ;is invalid
Push r1     ;is invalid
push acc    ;is correct
Push psw    ;is correct
Push b      ;is correct
Push 13h
Push 0
Push 1
Pop  7
Pop  8
Push 0e0h   ;acc
Pop  0f0h   ;b
```

Exchange Instructions

two way data transfer

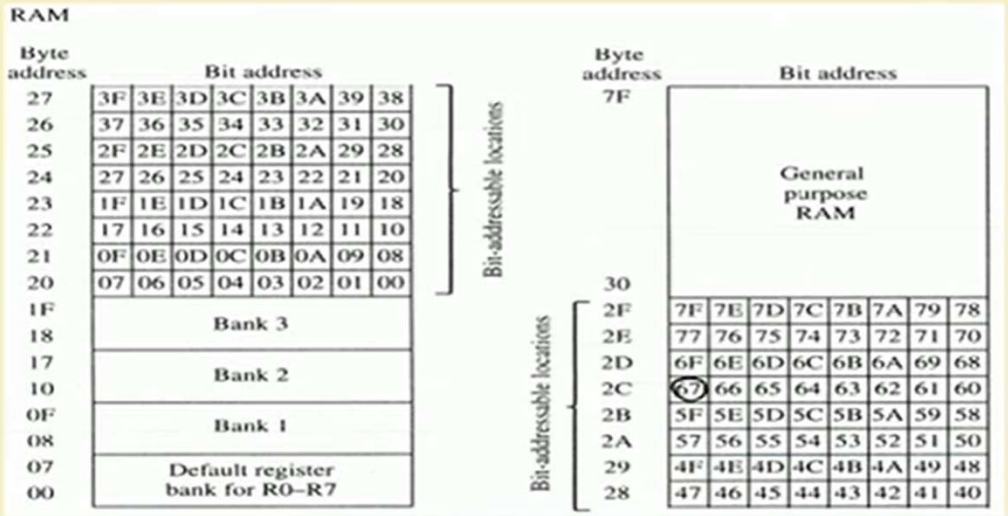
```
XCH a, 30h          ; a ←→ M[30]
XCH a, R0           ; a ←→ R0
XCH a, @R0          ; a ←→ M[R0]
XCHD a, R0          ; exchange "digit"
```



Bit-Oriented Data Transfer

- transfers between individual bits.
- Carry flag (C) (bit 7 in the PSW) is used as a single-bit accumulator
- RAM bits in addresses 20-2F are bit addressable

```
mov C, P0.0

mov C, 67h

mov C, 2ch.7
```



SFRs that are Bit Addressable

SFRs with addresses ending in 0 or 8 are bit-addressable.

(80, 88, 90, 98, etc)

Notice that all 4 parallel I/O ports are bit addressable.

Byte address	Bit address									Byte address	Bit address								
98	9F	9E	9D	9C	9B	9A	99	98	SCON	FF									
										F0	F7	F6	F5	F4	F3	F2	F1	F0	B
90	97	96	95	94	93	92	91	90	P1										
										E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
8D	not bit addressable								TH1										
8C	not bit addressable								TH0	D0	D7	D6	D5	D4	D3	D2	–	D0	PSW
8B	not bit addressable								TL1										
8A	not bit addressable								TL0	B8	–	–	–	BC	BB	BA	B9	B8	IP
89	not bit addressable								TMOD										
88	8F	8E	8D	8C	8B	8A	89	88	TCON	B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
87	not bit addressable								PCON										
										A8	AF	–	–	AC	AB	AA	A9	A8	IE
83	not bit addressable								DPH										
82	not bit addressable								DPL	A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
81	not bit addressable								SP										
80	87	86	85	84	83	82	81	80	P0	99	not bit addressable								SBUF

Data Processing Instructions

Arithmetic Instructions

Logic Instructions

Arithmetic Instructions

- Add
- Subtract
- Increment
- Decrement
- Multiply
- Divide
- Decimal adjust

Arithmetic Instructions

Mnemonic	Description
ADD A, byte	add A to byte, put result in A
ADDC A, byte	add with carry
SUBB A, byte	subtract with borrow
INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte
MUL AB	multiply accumulator by b register
DIV AB	divide accumulator by b register
DA A	decimal adjust the accumulator

ADD Instructions

```
add a, byte          ; a ← a + byte
addc a, byte                ; a ← a + byte + C
```

These instructions affect 3 bits in PSW:

C = 1 if result of add is greater than FF

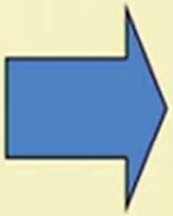
AC = 1 if there is a carry out of bit 3

OV = 1 if there is a carry out of bit 7, but not from bit 6, or visa versa.

Program Status Word (PSW)

Bit	7	6	5	4	3	2	1	0
Flag	CY	AC	F0	RS1	RS0	OV	F1	P
Name	Carry Flag	Auxiliary Carry Flag	User Flag 0	Register Bank Select 1	Register Bank Select 0	Overflow flag	User Flag 1	Parity Bit

Instructions that Affect PSW bits



Instructions that Affect Flag Settings(1)

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

Signed Addition and Overflow

2's complement:

0000 0000	00	0
…		
0111 1111	7F	127
1000 0000	80	-128
…		
1111 1111	FF	-1

```
 0111 1111   (positive 127)
 0111 0011   (positive 115)
 1111 0010   (overflow
cannot represent 242 in 8
bits 2's complement)

 1000 1111   (negative 113)
 1101 0011   (negative  45)
 0110 0010   (overflow)

0011 1111   (positive)
1101 0011   (negative)
0001 0010 (never overflows)
```

Addition Example

```
; Computes Z = X + Y
; Adds values at locations 78h and 79h and puts them in 7Ah
;-----------------------------------------------------------------
X               equ             78h
Y               equ             79h
Z               equ             7Ah
;-----------------------------------------------------------------
                org 00h
                ljmp Main
;-----------------------------------------------------------------
                org 100h
Main:
                mov a, X
                add a, Y
                mov Z, a
                end
```

The 16-bit ADD example

```
; Computes Z = X + Y     (X,Y,Z are 16 bit)
;-----------------------------------------------------------------
X               equ         78h
Y               equ         7Ah
Z               equ         7Ch
;-----------------------------------------------------------------
                org 00h
                ljmp Main
;-----------------------------------------------------------------
                org 100h
Main:
                mov a, X
                add a, Y
                mov Z, a
              mov a, X+1
                adc a, Y+1
                mov Z+1, a
                end
```

Subtract

SUBB A, byte	subtract with borrow

Example:

```
SUBB A, #0x4F     ;A ← A - 4F - C
```

Notice that
There is no subtraction WITHOUT borrow.
Therefore, if a subtraction without borrow is desired,
it is necessary to clear the C flag.

Example:

```
Clr  c
SUBB A, #0x4F     ;A ← A - 4F
```

Increment and Decrement

INC A	increment A
INC byte	increment byte in memory
INC DPTR	increment data pointer
DEC A	decrement accumulator
DEC byte	decrement byte

- The increment and decrement instructions do NOT affect the C flag.
- Notice we can only INCREMENT the data pointer, not decrement.

Example: Increment 16-bit Word

- Assume 16-bit word in R3:R2

```
mov a, r2
add a, #1          ; use add rather than increment to affect C
mov r2, a
mov a, r3
addc a, #0         ; add C to most significant byte
mov r3, a
```

Multiply

When multiplying two 8-bit numbers, the size of the maximum product is 16-bits

FF x FF = FE01

(255 x 255 = 65025)

```
MUL AB          ; BA ← A * B
```

Note : B gets the High byte

A gets the Low byte

Division

- Integer Division

```
DIV AB      ; divide A by B

A ← Quotient(A/B)
B ← Remainder(A/B)
```

OV - used to indicate a divide by zero condition.
C – set to zero

Decimal Adjust

```
DA a            ; decimal adjust a
```

Used to facilitate BCD addition.
Adds "6" to either high or low nibble after an addition
to create a valid BCD number.

Example:

```
        mov a, #23h
        mov b, #29h
        add a, b            ; a ← 23h + 29h = 4Ch (wanted 52)
        DA a                ; a ← a + 6 = 52
```

Logic Instructions

- Bitwise logic operations
  - (AND, OR, XOR, NOT)
- Clear
- Rotate
- Swap

Logic instructions do NOT affect the flags in PSW

Bitwise Logic

ANL → AND

ORL → OR

XRL → XOR

CPL → Complement

Examples:

```
        00001111
ANL     10101100
        00001100

        00001111
ORL     10101100
        10101111

        00001111
XRL     10101100
        10100011

CPL     10101100
        01010011
```

Address Modes with Logic

ANL – AND

ORL – OR

XRL – eXclusive oR

a, byte

direct, reg. indirect, reg, immediate

byte, a

direct

byte, #constant

CPL – Complement

a ex: cpl a

Uses of Logic Instructions

- Force individual bits low, without affecting other bits.

```
anl PSW, #0xE7          ;PSW AND 11100111
```

- Force individual bits high.

```
orl PSW, #0x18          ;PSW OR 00011000
```

- Complement individual bits

```
xrl P1, #0x40                ;P1 XRL 01000000
```

Other Logic Instructions

CLR - clear

RL - rotate left

RLC - rotate left through Carry

RR - rotate right

RRC - rotate right through Carry

SWAP - swap accumulator nibbles

CLR (Set all bits to 0)

CLR A

CLR byte (direct mode)

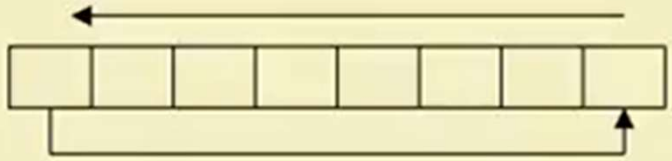
CLR Ri (register mode)

CLR @Ri (register indirect mode)

Rotate

- Rotate instructions operate only on **a**

RL a



```
Mov a,#0xF0      ; a← 11110000
RR a             ; a← 11100001
```

RR a

```
Mov a,#0xF0      ; a← 11110000
RR a             ; a← 01111000
```

Rotate through Carry

RRC a


```
mov a, #0A9h        ; a ← A9
add a, #14h         ; a ← BD (10111101), C←0
rrc a               ; a ←  01011110, C←1
```

RLC a


```
mov a, #3ch         ; a ← 3ch(00111100)
setb c              ; c ← 1
rlc a               ; a ←  01111001, C←1
```

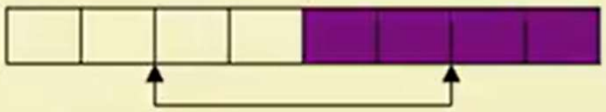
Rotate and Multiplication/Division

- Note that a shift left is the same as multiplying by 2, shift right is divide by 2

```
mov a, #3    ; A← 00000011 (3)
clr C        ; C← 0
rlc a        ; A← 00000110 (6)
rlc a        ; A← 00001100 (12)
rrc a        ; A← 00000110 (6)
```

Swap

SWAP a



```
mov a, #72h      ; a ← 72h
swap a           ; a ← 27h
```

Bit Logic Operations

- Some logic operations can be used with single bit operands

```
ANL C, bit
ORL C, bit
CLR C
CLR bit
CPL C
CPL bit
SETB C
SETB bit
```

- "bit" can be any of the bit-addressable RAM locations or SFRs.

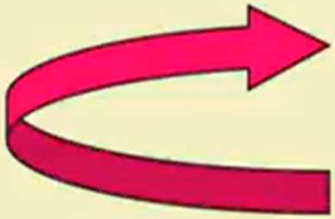
Program Flow Control

- Unconditional jumps ("go to")
- Conditional jumps
- Call and return

Unconditional Jumps

- **SJMP <rel addr>** ; Short jump, relative address is 8-bit 2's complement number, so jump can be up to 127 locations forward, or 128 locations back.
- **LJMP <address 16>** ; Long jump
- **AJMP <address 11>** ; Absolute jump to anywhere within 2K block of program memory
- **JMP @A + DPTR** ; Long indexed jump

Infinite Loops



```
Start: mov C, p3.7
       mov p1.6, C
       sjmp Start
```

Microcontroller application programs are almost always infinite loops!

Re-locatable Code

Memory specific NOT Re-locatable (machine code)

```
        org 8000h
Start:  mov C, p1.6
        mov p3.7, C
        ljmp Start
        end
```

Re-locatable (machine code)

```
        org 8000h
Start:  mov C, p1.6
        mov p3.7, C
        sjmp Start
        end
```

Jump table

```
                Mov dptr,#jump_table
                Mov a,#index_number
                Rl  a
                Jmp @a+dptr
                   ...
Jump_table:     ajmp case0
                ajmp case1
                ajmp case2
                ajmp case3
```

Conditional Jump

- These instructions cause a jump to occur only if a condition is true. Otherwise, program execution continues with the next instruction.

```
loop: mov a, P1
      jz  loop      ; if a=0, goto loop,
                    ; else goto next instruction
      mov b, a
```

- There is no zero flag (z)
- Content of A checked for zero on time

Conditional jumps

Mnemonic	Description
JZ <rel addr>	Jump if a = 0
JNZ <rel addr>	Jump if a != 0
JC <rel addr>	Jump if C = 1
JNC <rel addr>	Jump if C != 1
JB <bit>, <rel addr>	Jump if bit = 1
JNB <bit>,<rel addr>	Jump if bit != 1
JBC <bir>, <rel addr>	Jump if bit =1, &clear bit
CJNE A, direct, <rel addr>	Compare A and memory, jump if not equal

Example: Conditional Jumps

```
if (a = 0) is true
    send a 0 to LED
else
    send a 1 to LED
```

```
              jz led_off
              Setb P1.6
              sjmp skipover
 led_off:     clr P1.6
              mov A, P0
skipover:
```

More Conditional Jumps

Mnemonic	Description
CJNE A, #data <rel addr>	Compare A and data, jump if not equal
CJNE Rn, #data <rel addr>	Compare Rn and data, jump if not equal
CJNE @Rn, #data <rel addr>	Compare Rn and memory, jump if not equal
DJNZ Rn, <rel addr>	Decrement Rn and then jump if not zero
DJNZ direct, <rel addr>	Decrement memory and then jump if not zero

Iterative Loops

For A = 0 to 4 do

{...}

```
        clr a
loop:   ...
        ...
        inc a
        cjne a, #4, loop
```

For A = 4 to 0 do

{...}

```
        mov R0, #4
loop:   ...
        ...
        djnz R0, loop
```

Call and Return

- Call is similar to a jump, but
  - Call pushes PC on stack before branching

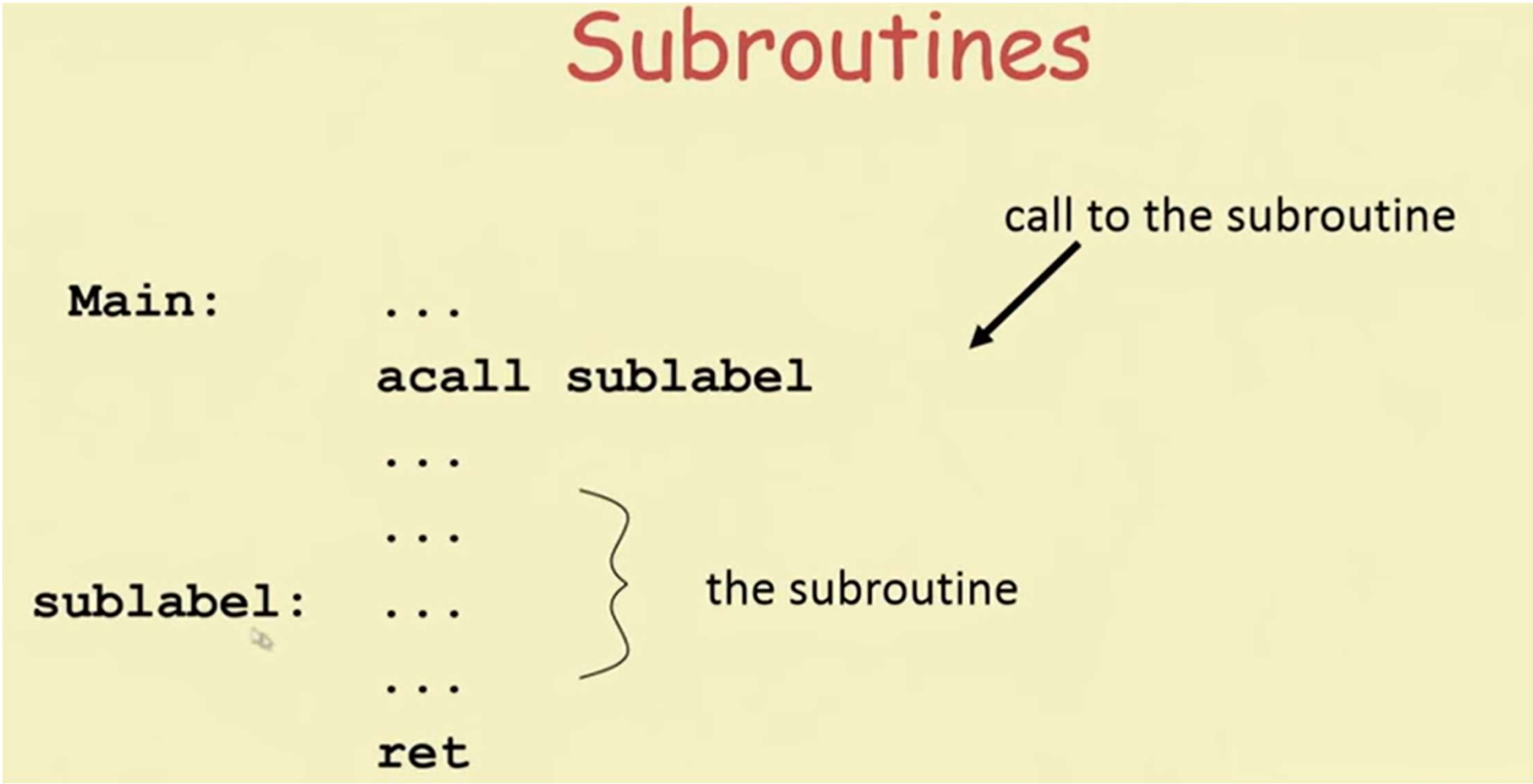
```
acall <address 11>          ; stack ← PC
                              ; PC ← address 11 bit

lcall <address 16>          ; stack ← PC
                              ; PC ← address 16 bit
```

Return

- Return is also similar to a jump, but
  - Return instruction pops PC from stack to get address to jump to

```
ret                    ; PC ← stack
```

Subroutines



Initializing Stack Pointer

- SP is initialized to 07 after reset.(Same address as R7)
- With each push operation 1st , pc is increased
- When using subroutines, the stack will be used to store the PC, so it is very important to initialize the stack pointer. Location 2Fh is often used.

```
mov SP, #2Fh
```

Subroutine - Example

```
square:  push b
         mov  b,a
         mul  ab
         pop  b
         ret
```

- 8 byte and 11 machine cycle

```
square: inc a
         movc a,@a+pc
         ret
table:  db 0,1,4,9,16,25,36,49,64,81
```

- 13 byte and 5 machine cycle

example of delay

```
      mov a,#0aah
Back1:mov p0,a
      lcall delay1
      cpl a
      sjmp back1
Delay1:mov r0,#0ffh;1cycle
Here: djnz r0,here ;2cycle
      ret          ;2cycle
      end
```

Delay=1+255*2+2=513 cycle

```
Delay2:
      mov r6,#0ffh
back1: mov r7,#0ffh ;1cycle
Here:  djnz r7,here ;2cycle
       djnz r6,back1;2cycle
       ret         ;2cycle
       end
```

Delay=1+(1+255*2+2)*255+2
=130818 machine cycle

Example

```
; p0:input  p1:output
          mov a,#0ffh
          mov p0,a
back:     mov a,p0
          mov p1,a
          sjmp back
```

Example

```
; duty cycle 50%
back:           cpl p1.2
                acall delay
                sjmp back
```

```
back:           setb p1.2
                acall delay
                Clr p1.2
                acall delay
                sjmp back
```

Example

```
; duty cycle 66%
back:    setb p1.2
         acall delay
         acall delay
         Clr p1.2
         acall delay
         sjmp back
```