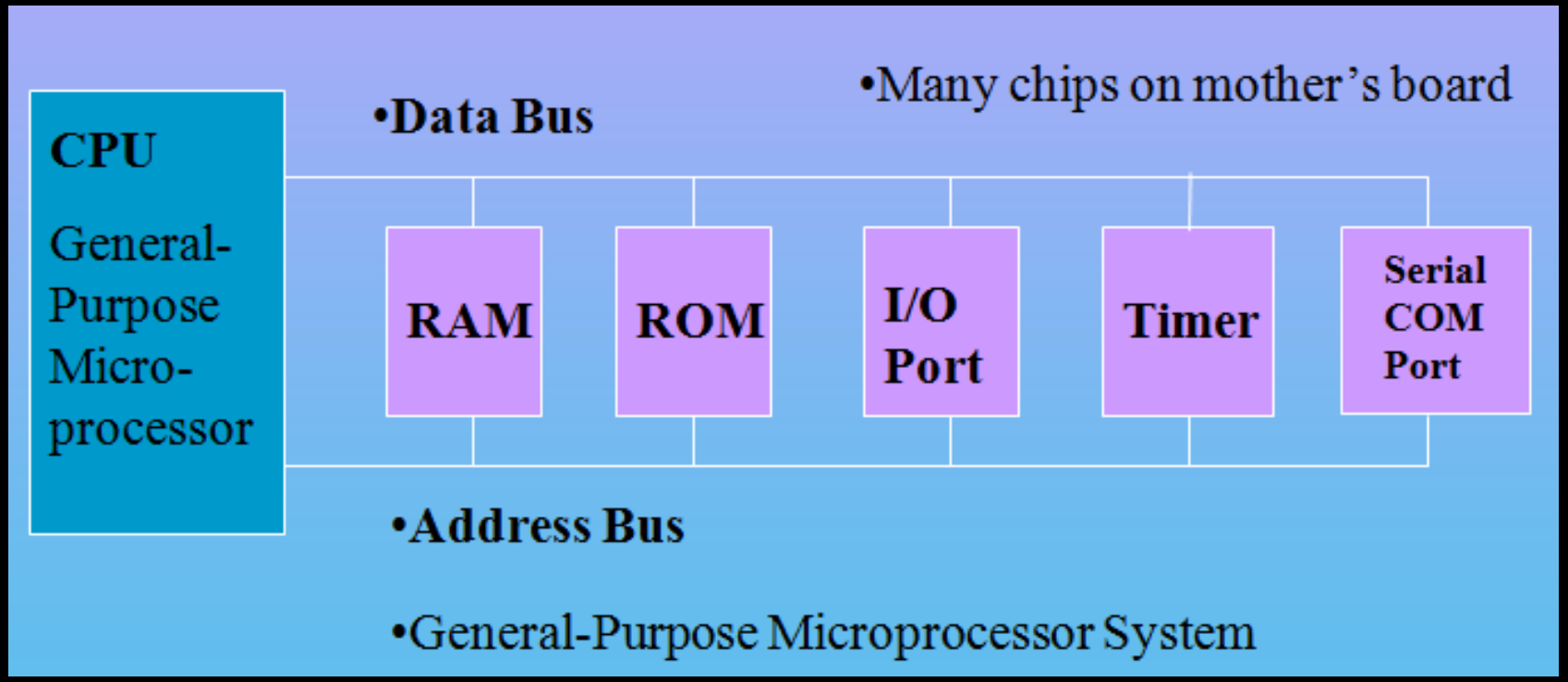
8051 Microcontroller

Microprocessor Based System

CPU

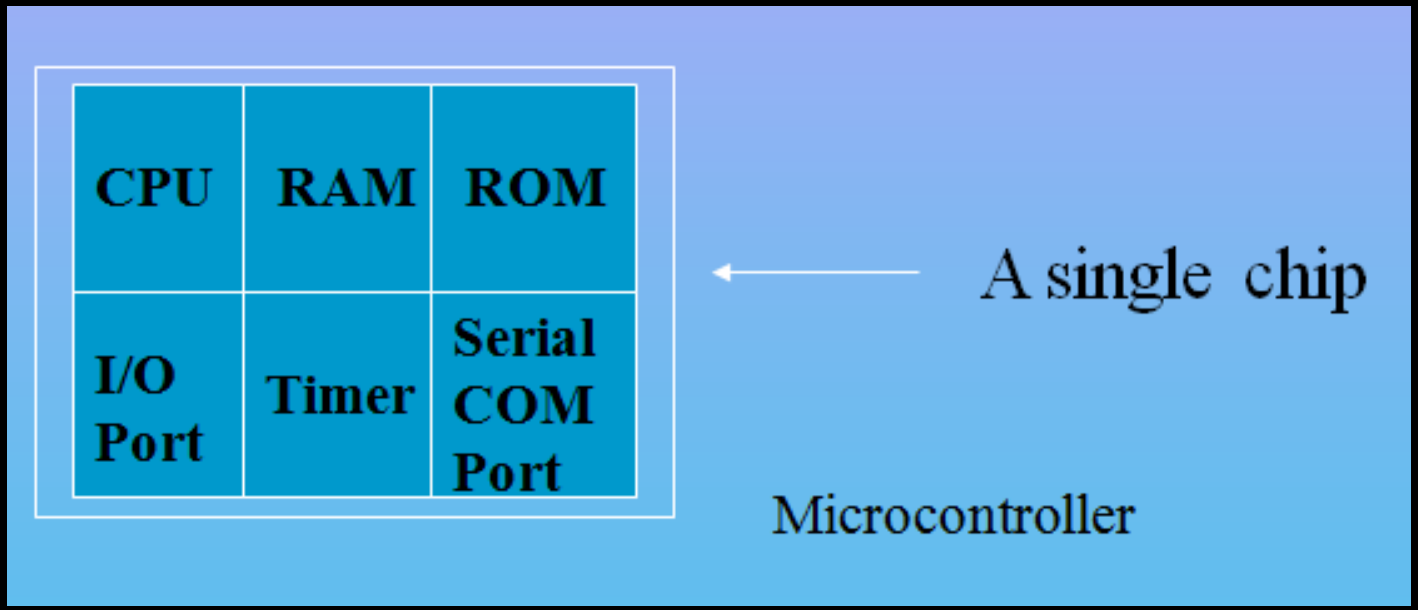
External RAM, ROM, I/O

(No internal RAM, ROM, I/O ports in the CPU)



Microcontroller

- A smaller computer on a CHIP
- On-chip RAM, ROM, I/O Ports, Timer, Serial Controller…
- Example: Motorola's 6811, Intel's 8051, Atmel 32



Microprocessor vs. Microcontroller

Microprocessor

- CPU is stand-alone, RAM, ROM, I/O, timer are separate
- Designer can decide on the amount of ROM, RAM and I/O ports.
- Expansive
- Versatility
- General-purpose

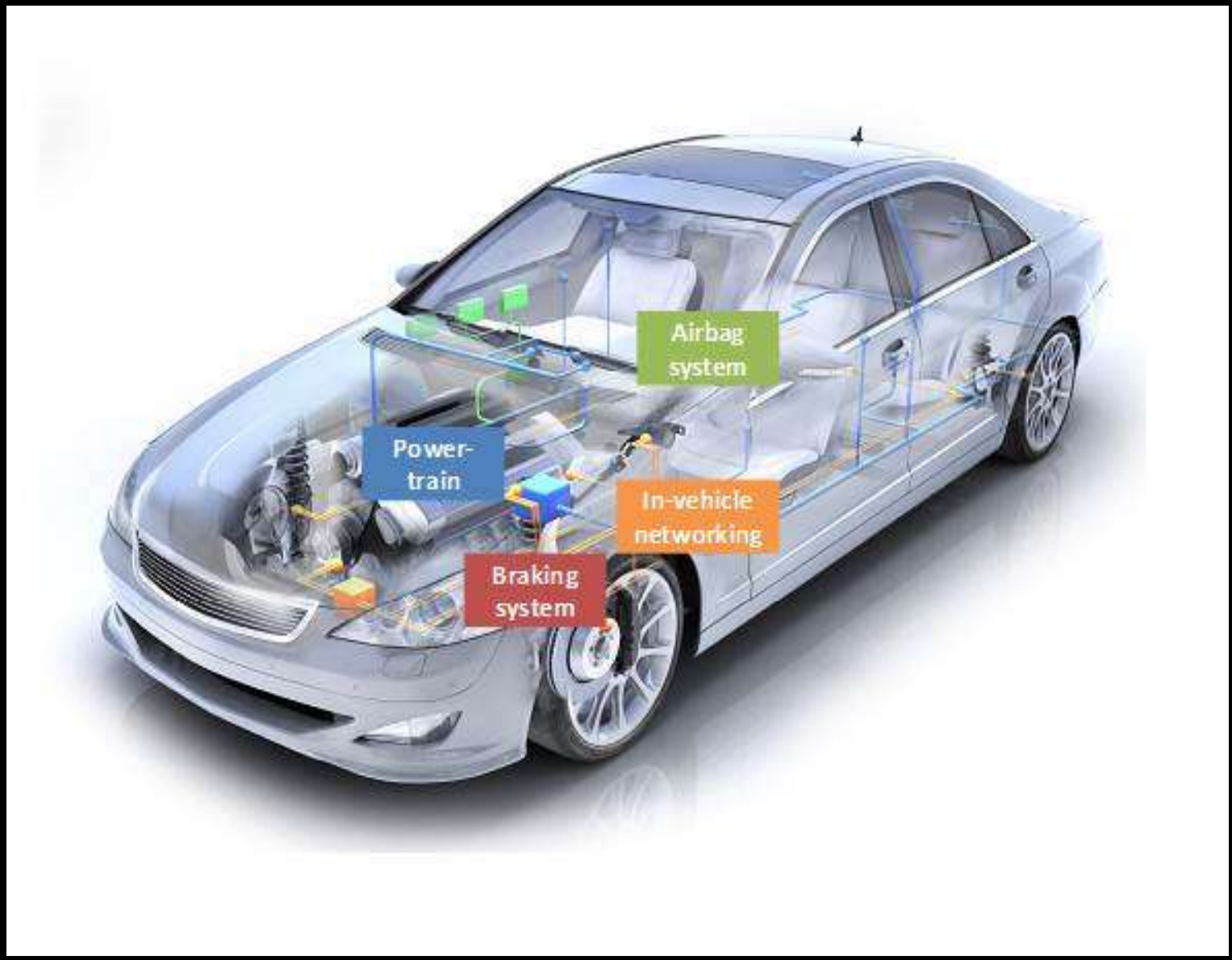
Microcontroller

- CPU, RAM, ROM, I/O and timer are all on a single chip
- Fixed amount of on-chip ROM, RAM, I/O ports
- Not Expansive
- Single-purpose
- Special Purpose.

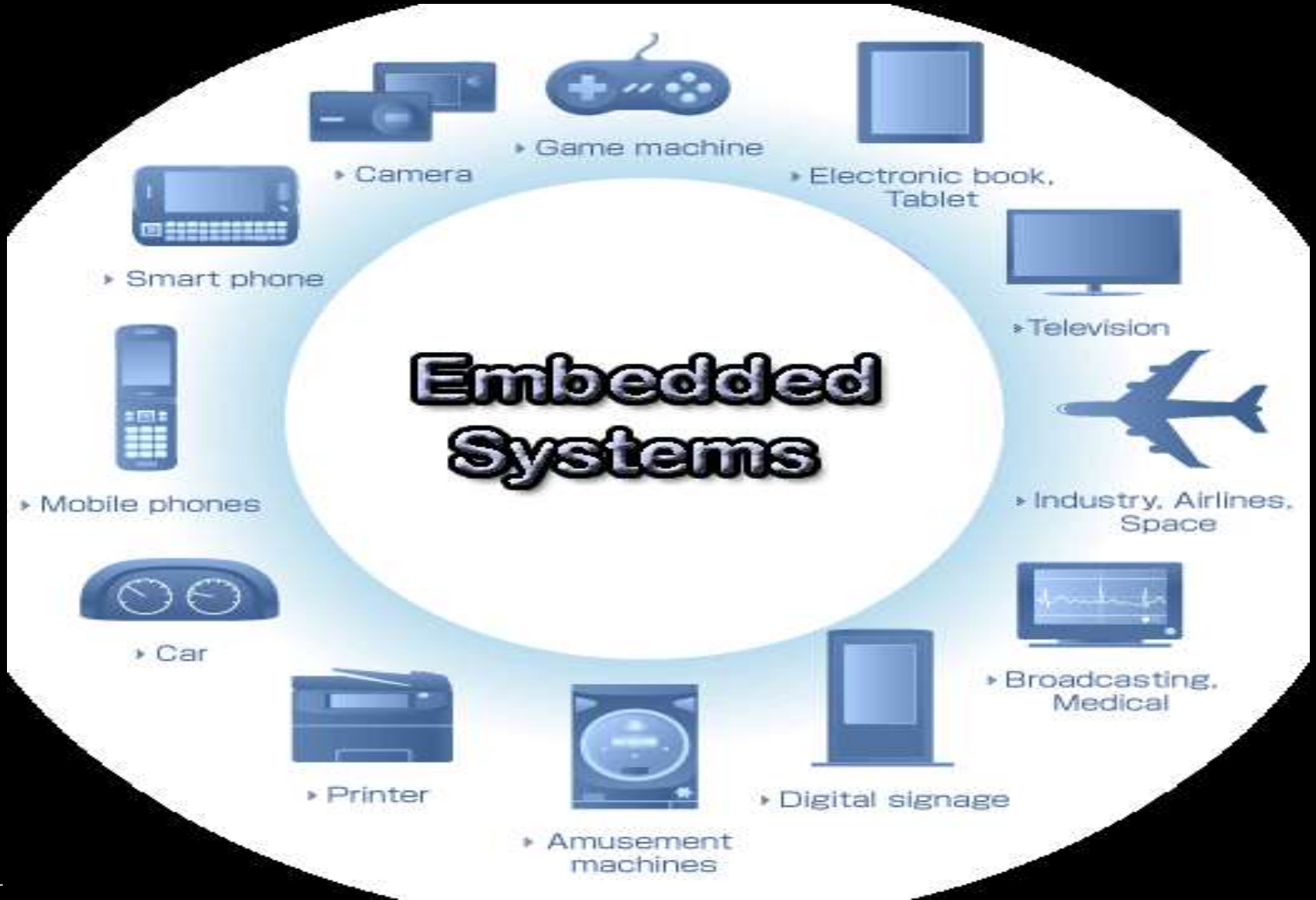
µC based Embedded Systems

- Special purpose computer system usually completely inside the device it controls
- Has specific requirements and performs pre-defined tasks
- Cost reduction compared to general purpose processor
- Different design criteria
  - Performance
  - Reliability
  - Availability
  - Safety

Embedded Systems Examples



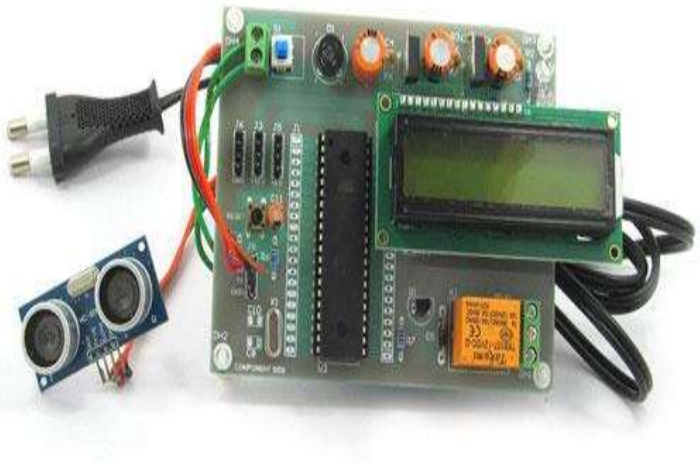
Examples


Embedded Systems
Camera
Game machine
Electronic book, Tablet
Smart phone
Television
Mobile phones
Industry, Airlines, Space
Car
Broadcasting, Medical
Printer
Digital signage
Amusement machines

ULTRA SONIC RANGE/DISTANCE FINDER USING 8051

Ultrasonic range finder system consists of an ultrasonic sensor which operates like SONAR technology. The system consists of an 8051 microcontroller, LCD Display and an ultrasonic sensor.

VIEW KIT



WATER LEVEL INDICATOR KIT

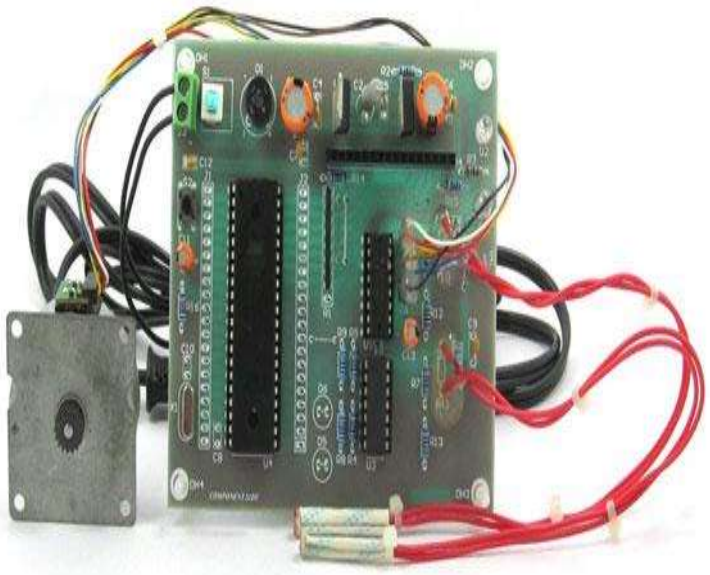
This system monitors the water level of the tank and automatically switches ON the motor whenever tank is empty.

VIEW KIT

SOLAR PANEL/SUN TRACKER USING 8051

The solar tracker is a sun tracker system for solar panel to achieve maximum efficiency. The system consists of two LDR's which has to be mounted on the sides of a solar panel.

VIEW KIT

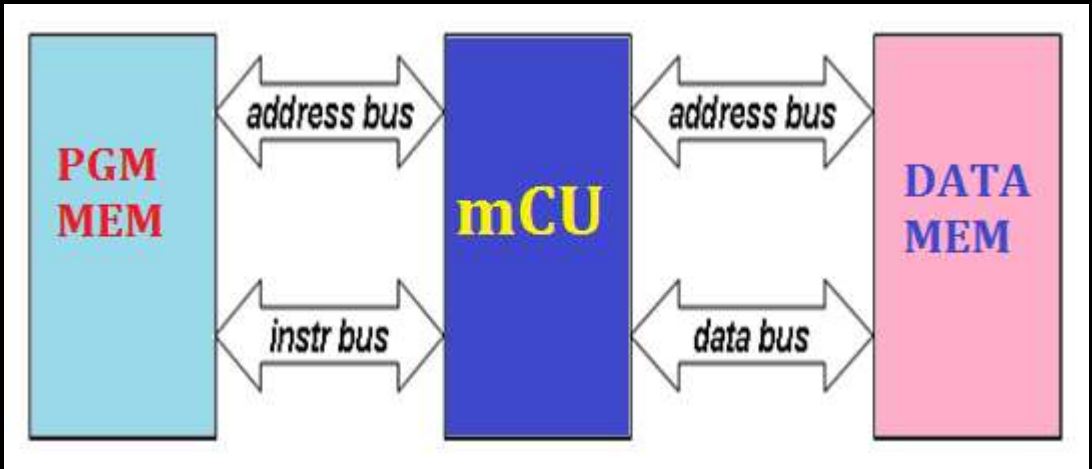


RFID BASED ACCESS CONTROL SYSTEM USING 8051

RFID based access control can be implemented in schools, colleges, offices, and so on. Nowadays RFID tags are associated with ID cards. Each tag has a unique id which can picked up by an RFID reader. Our system consists of such an RFID reader which operates on 125KHz.

VIEW KIT

Harvard Architecture



In Harvard Architecture the data and instructions are stored in separate memory units each with their own bus.

Advantages:

- Speeding up the data transfer rate,
- Permits the designer to implement different bus widths and word sizes for program and data memory space.

8051 CPU Operation

1. Features
2. Pin Diagram
3. Block Diagram

8051 Microcontroller

- Intel introduced 8051, referred as MCS- 51, in 1981.
- The 8051 is an **8-bit processor**
  - The CPU can work on only 8 bits of data at a time
- The 8051 became widely popular after allowing other manufactures to make and market any flavor of the 8051.

Features of 8051

8 bit Processor

4KB Internal ROM

128 Bytes Internal RAM

Four 8 BIT I/O PORTS (32 I/O LINES)

Two 16 Bit Timers/Counters

On Chip Full Duplex UART for Serial Communication

5 Vector Interrupts (2 External, 3 Internal - Timer0,Timer1,Serial)

On Chip Clock Oscillator

16 bit Address bus

- 64k External Code Memory
- 64k External Data Memory

16-bit program counter to access external Code Memory and

16 bit Data Pointer to access external Data Memory

128 user defined flags

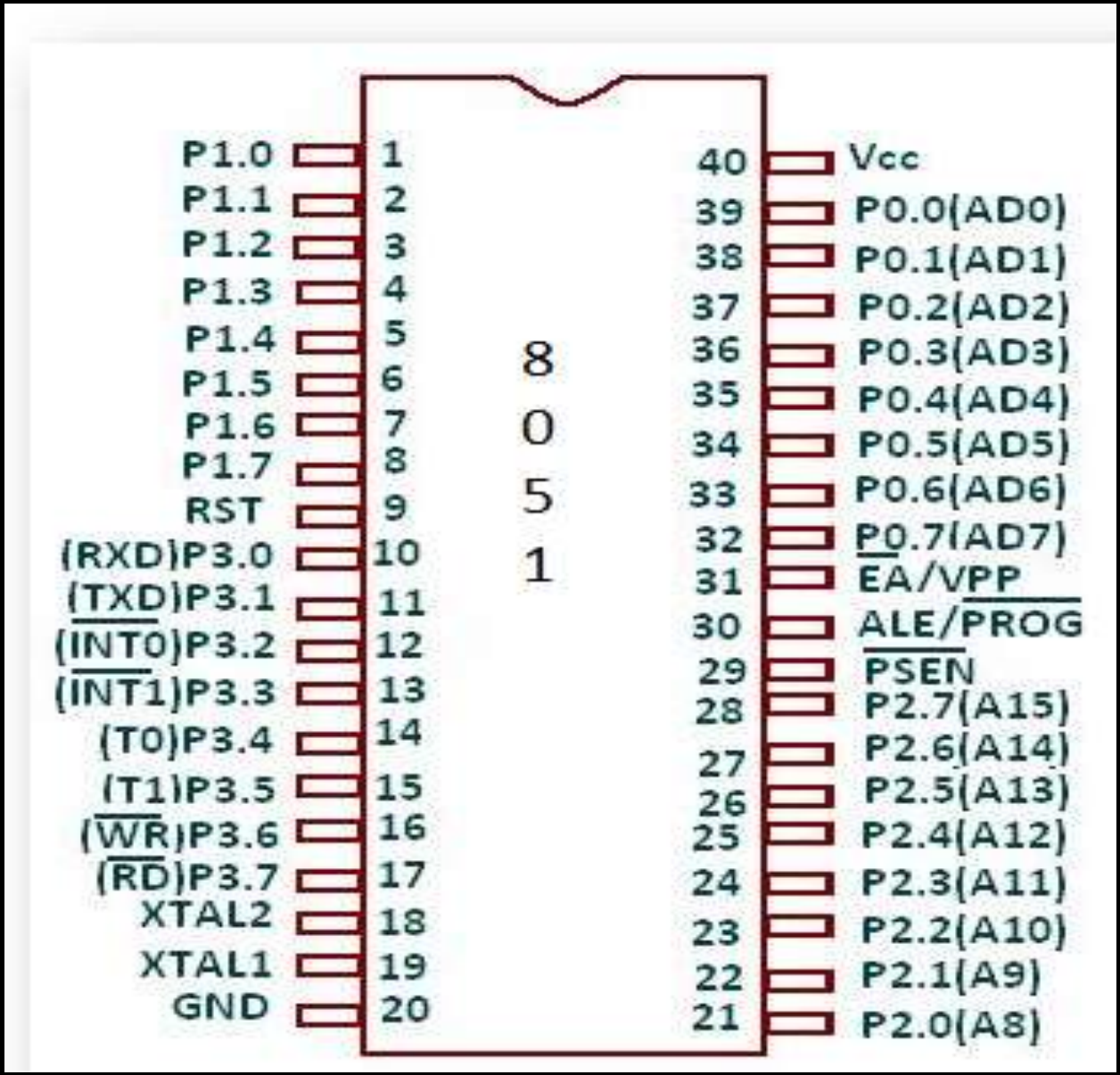
32 General Purpose Registers each of 8 bits

8051 Family

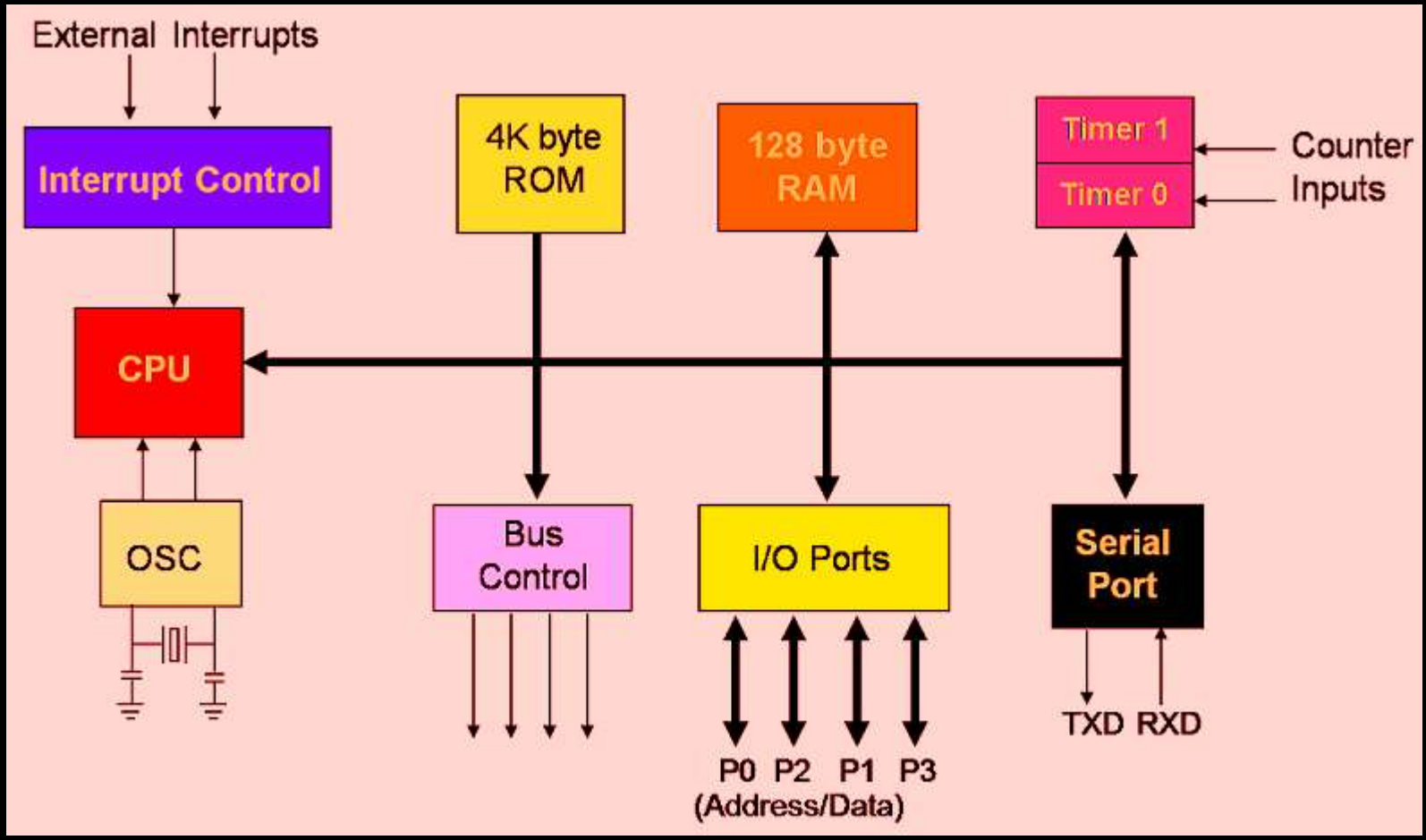
- The 8051 is a subset of the 8052
- The 8031 is a ROM-less 8051
  - Add external ROM to it
  - You lose two ports, and leave only 2 ports for I/O operations

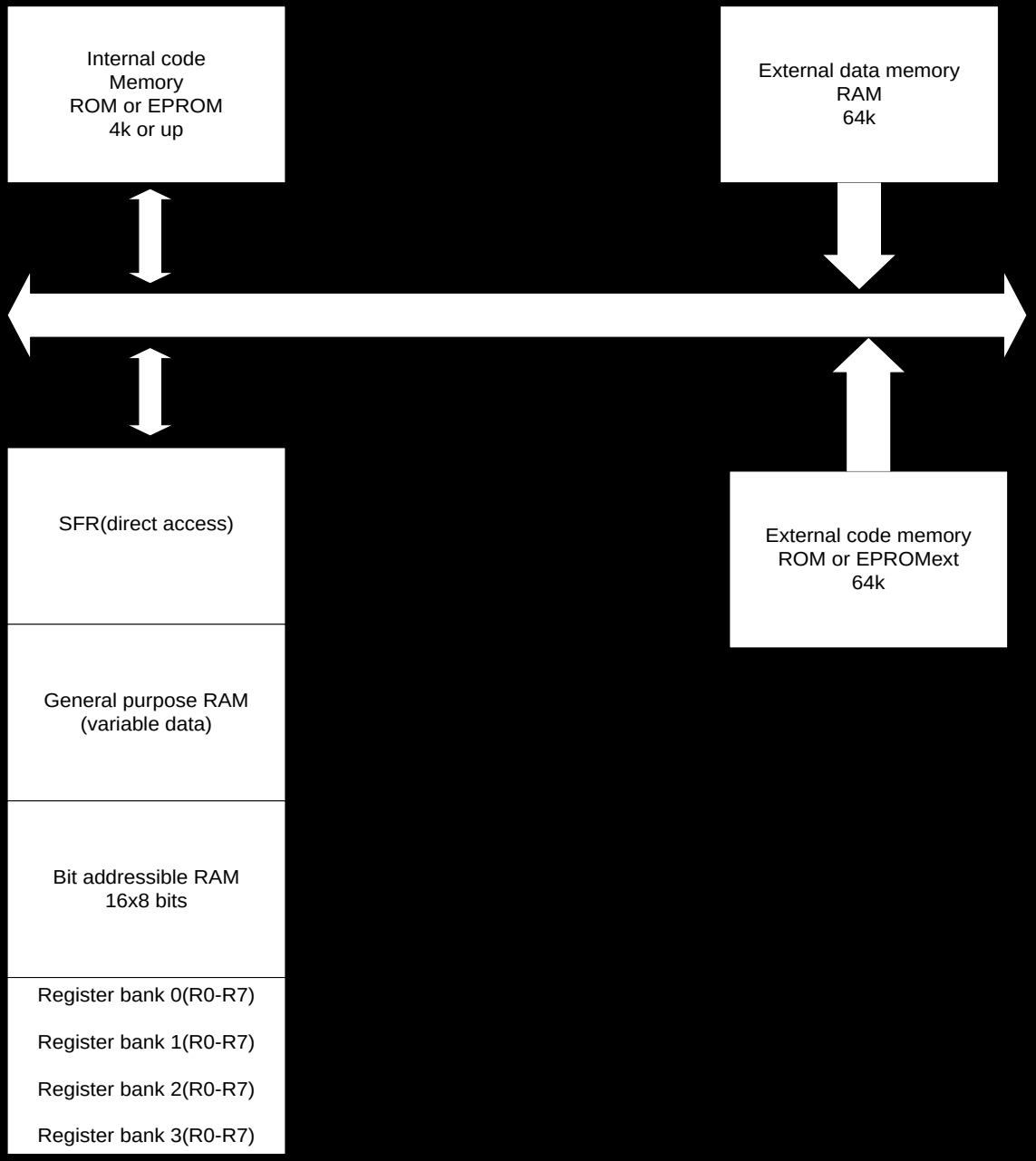
Feature	8051	8052	8031
ROM (on-chip program space in bytes)	4K	8K	0K
RAM (bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

Pin Diagram


P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD)P3.0 10
(TXD)P3.1 11
(INT0)P3.2 12
(INT1)P3.3 13
(T0)P3.4 14
(T1)P3.5 15
(WR)P3.6 16
(RD)P3.7 17
XTAL2 18
XTAL1 19
GND 20
8051
40 Vcc
39 P0.0(AD0)
38 P0.1(AD1)
37 P0.2(AD2)
36 P0.3(AD3)
35 P0.4(AD4)
34 P0.5(AD5)
33 P0.6(AD6)
32 P0.7(AD7)
31 EA/VPP
30 ALE/PROG
29 PSEN
28 P2.7(A15)
27 P2.6(A14)
26 P2.5(A13)
25 P2.4(A12)
24 P2.3(A11)
23 P2.2(A10)
22 P2.1(A9)
21 P2.0(A8)

Block Diagram of 8051



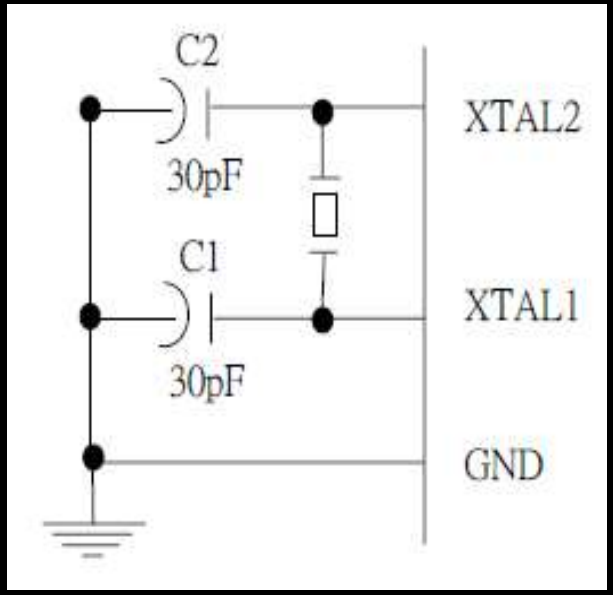

Internal code
Memory
ROM or EPROM
4k or up
External data memory
RAM
64k
SFR(direct access)
General purpose RAM
(variable data)
Bit addressible RAM
16x8 bits
Register bank 0(R0-R7)
Register bank 1(R0-R7)
Register bank 2(R0-R7)
Register bank 3(R0-R7)
External code memory
ROM or EPROMext
64k

Pin Description of the 8051

- 8051 family members (e.g., 8751, 89C51, 89C52, DS89C4x0)
  - Have **40 pins** dedicated for various functions such as I/O, RD, WR, address, data, and interrupts.
  - Come in different packages, such as
    - *DIP(dual in-line package),*
    - *QFP(quad flat package), and*
    - *LLC(leadless chip carrier)*
- Some companies provide a 20-pin version of the 8051 with a reduced number of I/O ports for less demanding applications

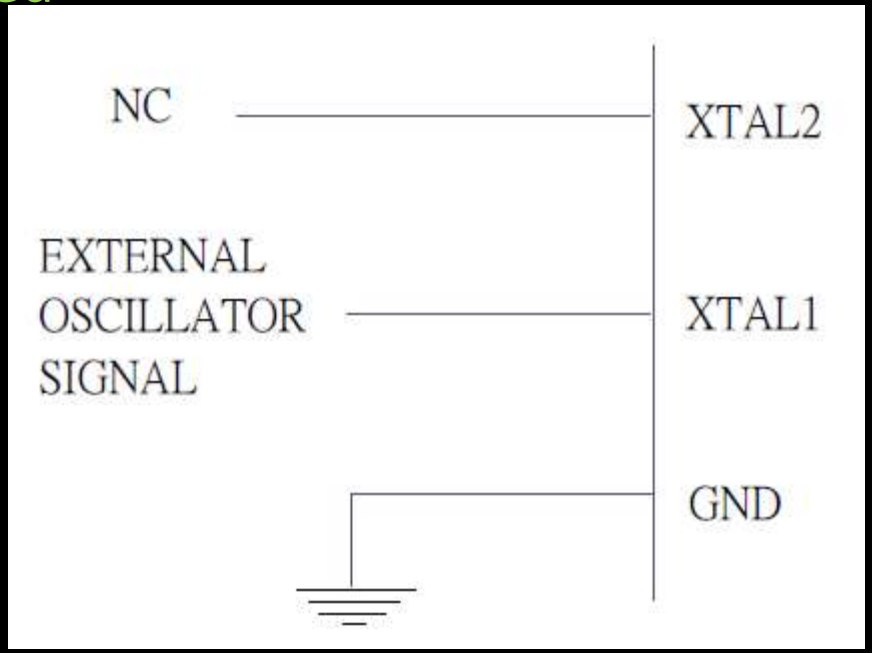
XTAL1 and XTAL2

- The 8051 has an on-chip oscillator but requires an external crystal to run it
  - A quartz crystal oscillator is connected to inputs XTAL1 (pin19) and XTAL2 (pin18)
  - The quartz crystal oscillator also needs two capacitors of 30 pF value
  - The original 8051 operates at 12 MHZ



XTAL1 and XTAL2 …..

- If you use a frequency source other than a crystal oscillator, such as a TTL oscillator:
  - It will be connected to XTAL1
  - XTAL2 is left unconnected



RST

- RESET pin is an input and is active high (normally low)
- Upon applying a high pulse to this pin, the microcontroller will reset and terminate all activities
- This is often referred to as a power-on reset
- Activating a power-on reset will cause all values in the registers to be lost

RESET value of some 8051 registers

we must place the first line of source code in ROM location 0

Register	Reset Value
PC	0000
DPTR	0000
ACC	00
PSW	00
SP	07
B	00
P0-P3	FF

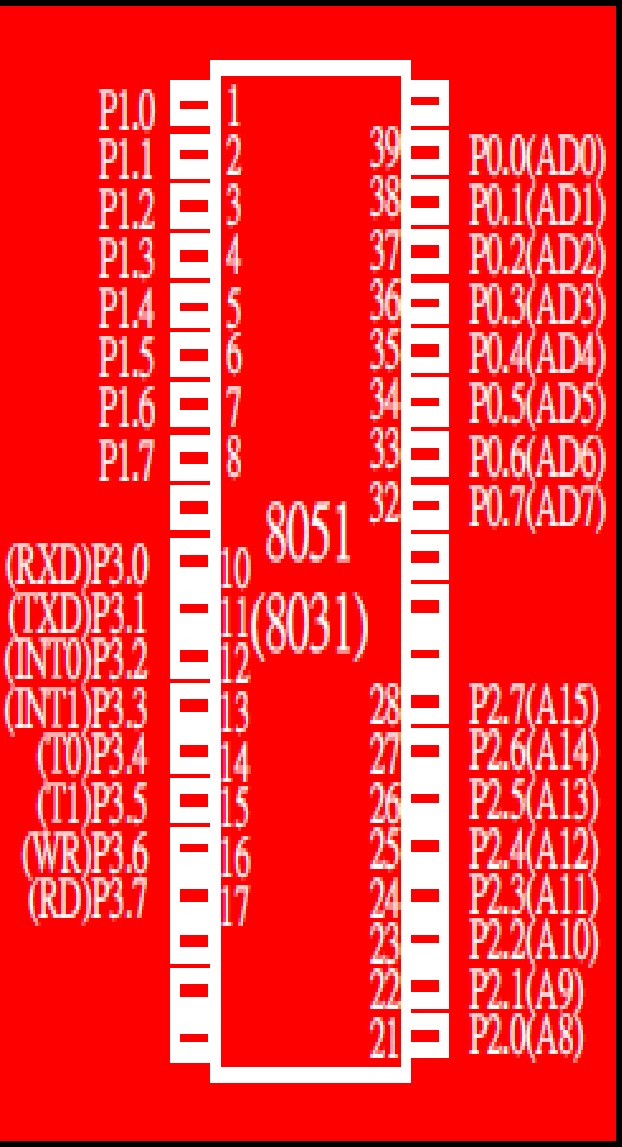
EA’

- EA’, **“external access”**, is an input pin and must be connected to Vcc or GND
- The 8051 family members all come with on-chip ROM to store programs and also have an external code and data memory.
- Normally EA pin is connected to Vcc (Internal Access)
- EA pin must be connected to GND to indicate that the code or data is stored externally.

PSEN' and ALE

- PSEN, **"program store enable"**, is an output pin
- This pin is connected to the OE pin of the external memory.
- For External Code Memory, PSEN' = 0
- For External Data Memory, PSEN' = 1
- ALE pin is used for demultiplexing the address and data.

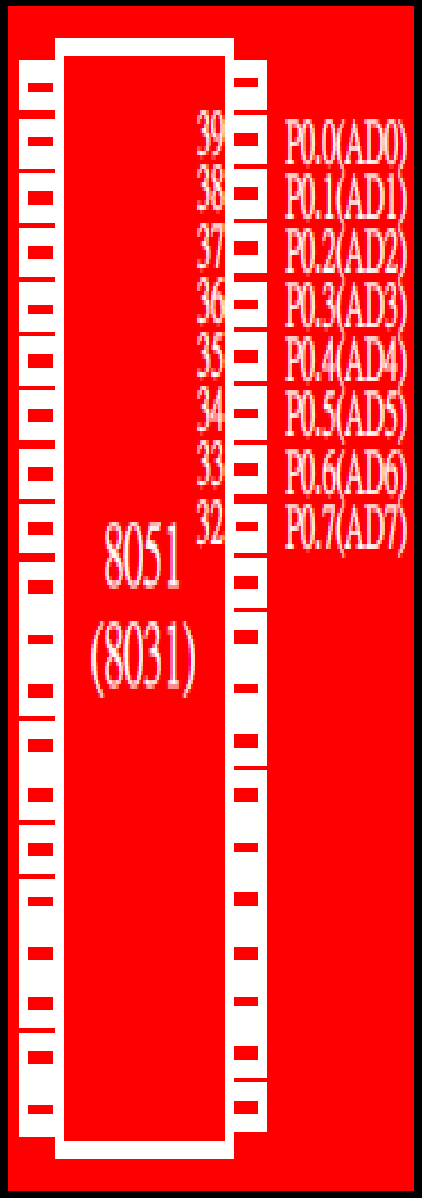
I/O Port Pins



The four 8-bit I/O ports **P0, P1, P2 and P3** each uses 8 pins.

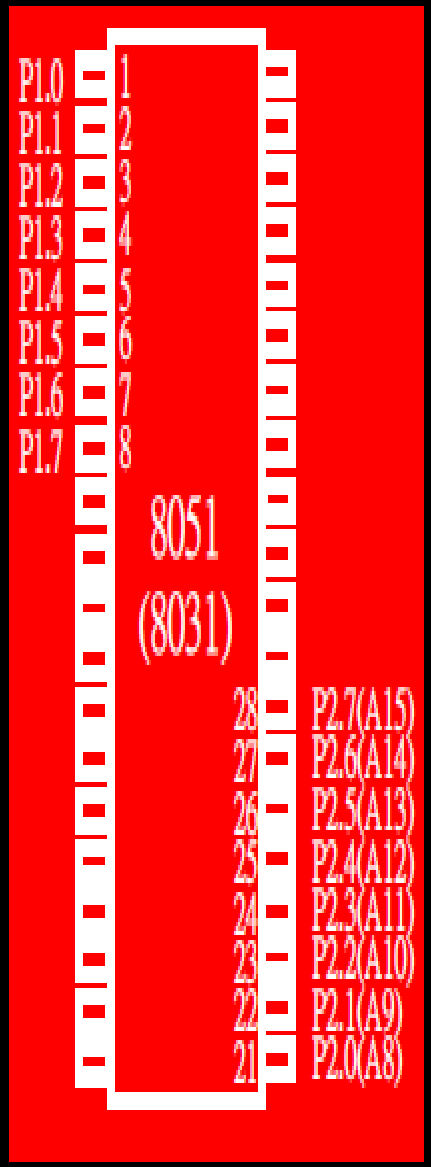
All the ports upon RESET are configured as output, ready to be used as input ports by the external device.

Port 0



- Port 0 is **also** designated as **AD0-AD7**.
- When connecting an 8051 to an external memory, port 0 provides both address and data.
- The 8051 multiplexes address and data through port 0 to save pins.
- **ALE** indicates if P0 has address or data.
  - *When ALE=0, it provides data D0-D7*
  - *When ALE=1, it has address A0-A7*

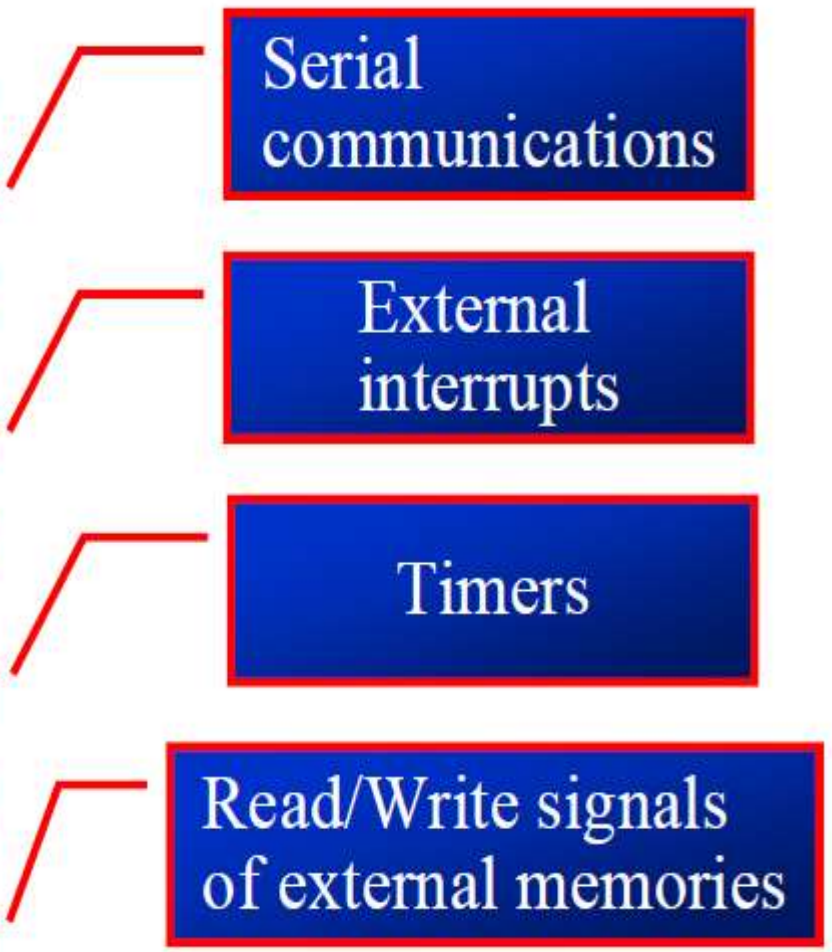
Port 1 and Port 2



- In 8051-based systems **with no external memory connection**:
  - Both P1 and P2 are used as simple I/O.
- In 8051-based systems **with external memory connections**:
  - Port 2 must be used along with P0 to provide the 16-bit address for the external memory.
  - P0 provides the lower 8 bits via A0 – A7.
  - P2 is used for the upper 8 bits of the 16-bit address, designated as A8 – A15, and it cannot be used for I/O.

Port 3

P3 Bit	Function	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17



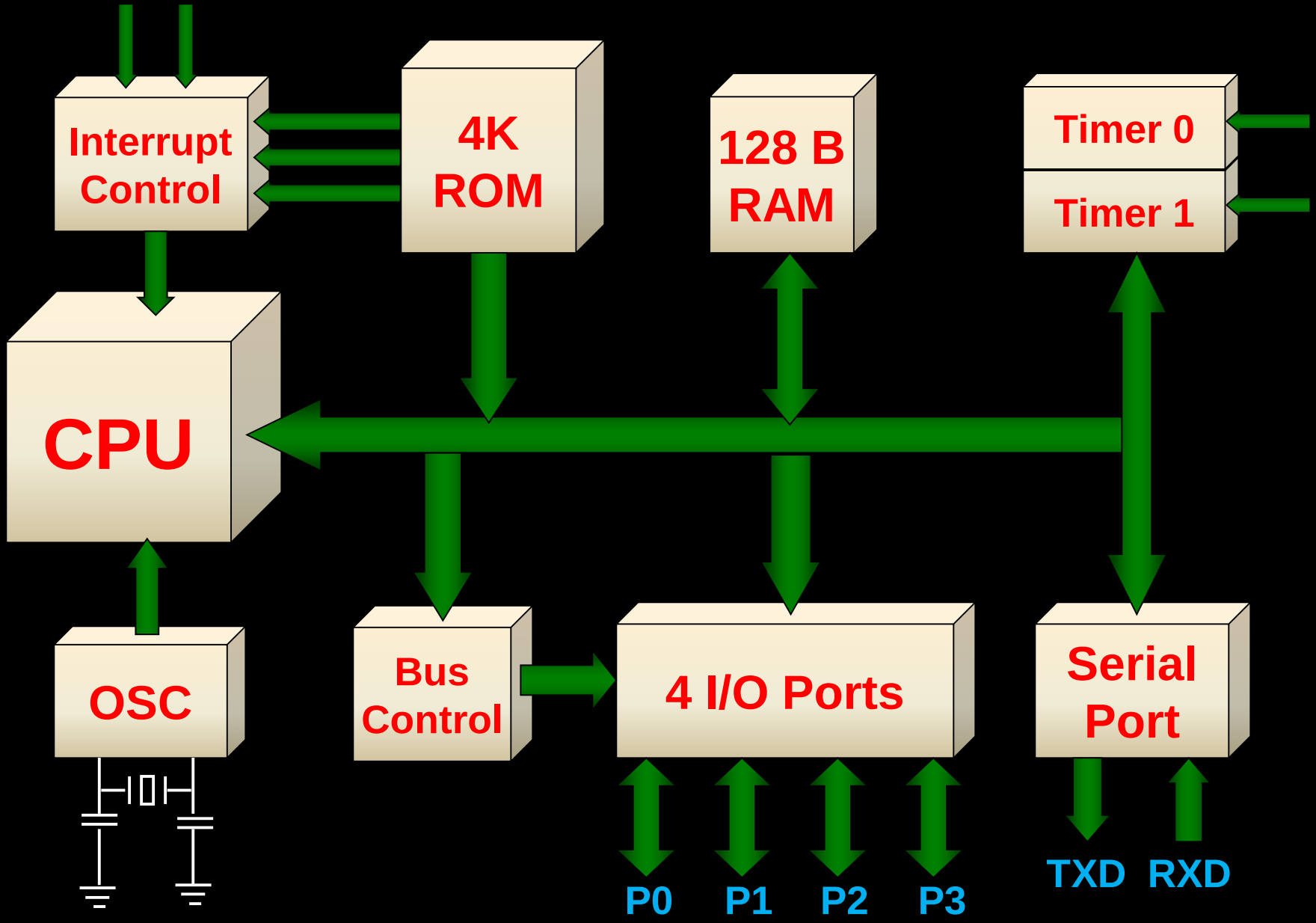
Pin Description Summary

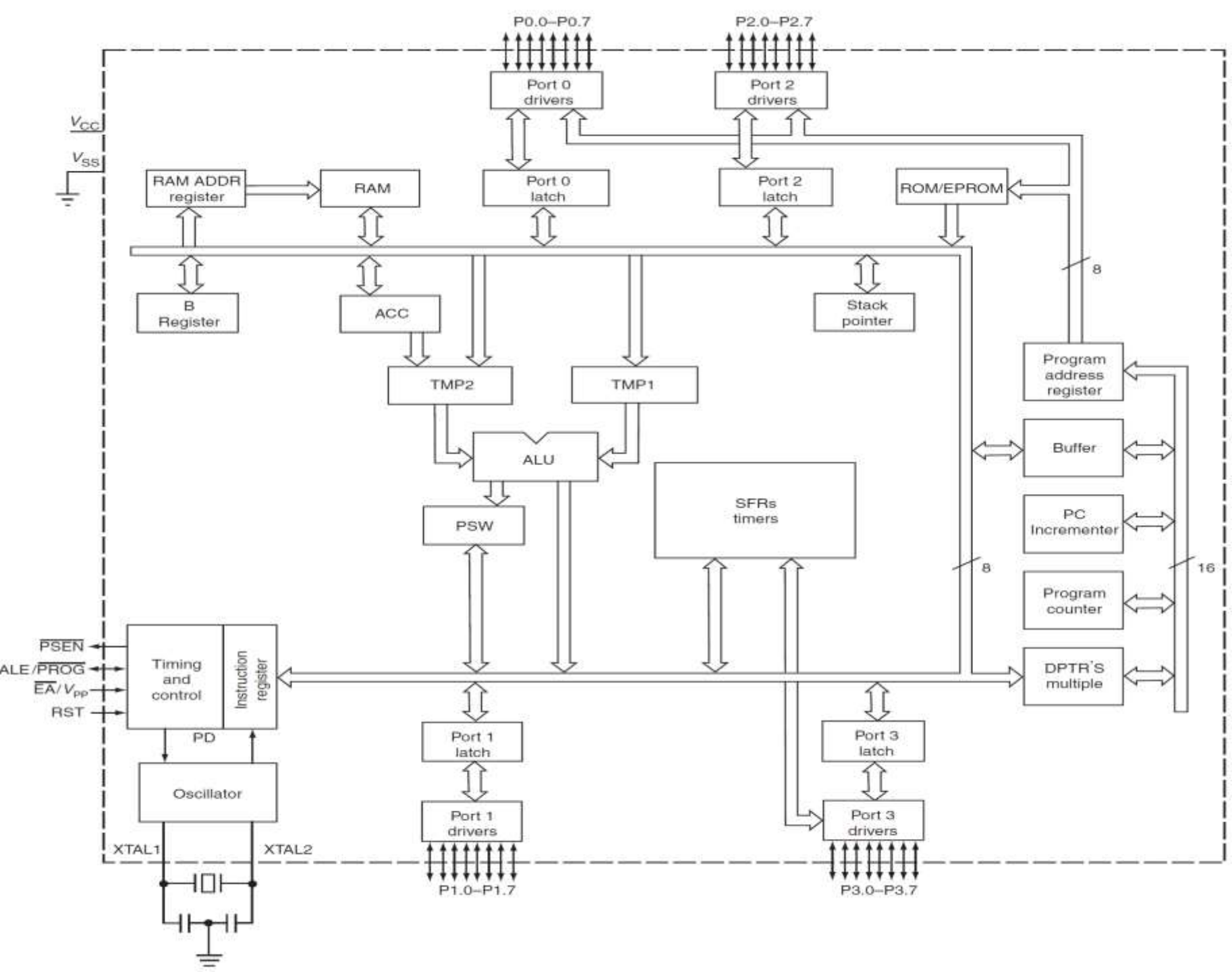
PIN	TYPE	NAME AND FUNCTION
Vss	**I**	**Ground:** 0 V reference.
Vcc	**I**	**Power Supply:** This is the power supply voltage for normal, idle, and power-down operation.
P0.0 - P0.7	**I/O**	**Port 0:** Port 0 is an open-drain, bi-directional I/O port. Port 0 is also the multiplexed low-order address and data bus during accesses to external program and data memory.
P1.0 - P1.7	**I/O**	**Port 1:** Port I is an 8-bit bi-directional I/O port.
P2.0 - P2.7	**I/O**	**Port 2:** Port 2 is an 8-bit bidirectional I/O. Port 2 emits the high order address byte during fetches from external program memory and during accesses to external data memory that use 16 bit addresses.
P3.0 - P3.7	**I/O**	**Port 3:** Port 3 is an 8 bit bidirectional I/O port. Port 3 also serves special features as explained.

Pin Description Summary

PIN	TYPE	NAME AND FUNCTION
RST	**I**	**Reset:** A high on this pin for two machine cycles while the oscillator is running, resets the device.
ALE	**O**	**Address Latch Enable:** Output pulse for latching the low byte of the address during an access to external memory.
PSEN*	**O**	**Program Store Enable:** The read strobe to external program memory. When executing code from the external program memory, PSEN* is activated twice each machine cycle, except that two PSEN* activations are skipped during each access to external data memory.
EA*/VPP	**I**	**External Access Enable/Programming Supply Voltage:** EA* must be externally held low to enable the device to fetch code from external program memory locations. If EA* Is held high, the device executes from internal program memory. This pin also receives the programming supply voltage Vpp during Flash programming. (applies for 89c5x MCU's)

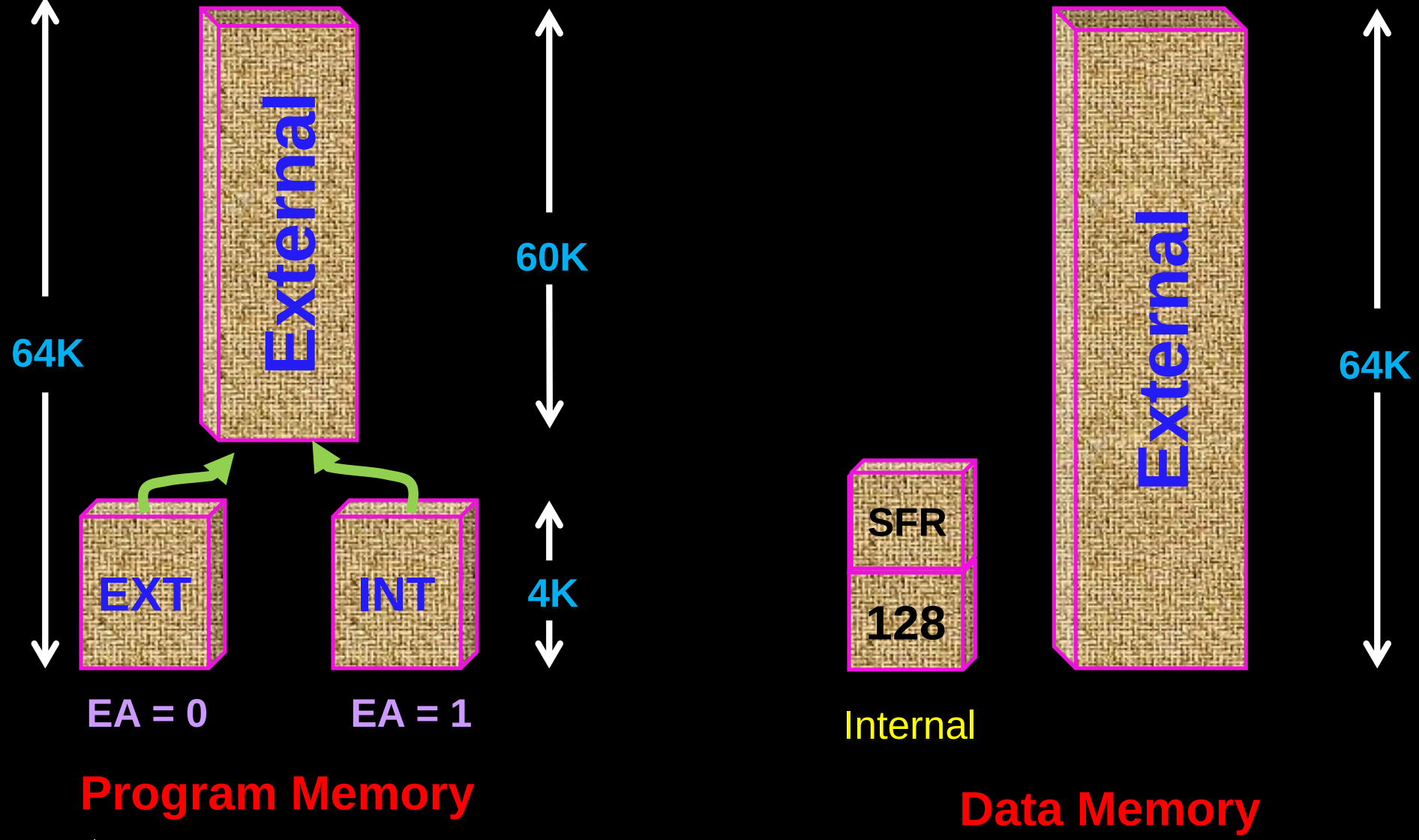
General Block Diagram of 8051



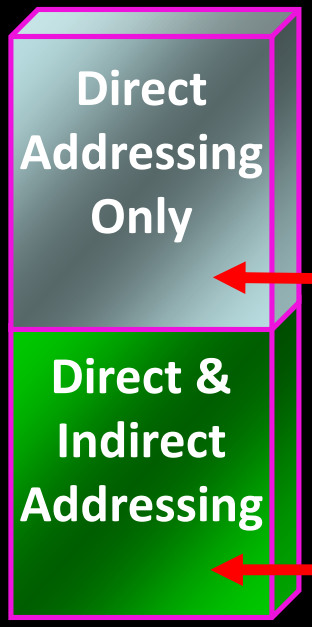

P0.0–P0.7
P2.0–P2.7
Port 0 drivers
Port 2 drivers
V_{CC}
V_{SS}
RAM ADDR register
RAM
Port 0 latch
Port 2 latch
ROM/EPROM
B Register
ACC
Stack pointer
8
Program address register
TMP2
TMP1
Buffer
ALU
SFRs timers
PC Incrementer
PSW
8
16
Program counter
DPTR'S multiple
PSEN
ALE/PROG
EA/V_{PP}
RST
Timing and control
Instruction register
PD
Port 1 latch
Port 3 latch
Oscillator
Port 1 drivers
Port 3 drivers
XTAL1
XTAL2
P1.0–P1.7
P3.0–P3.7

8051 Memory Space

8051 Memory Structure



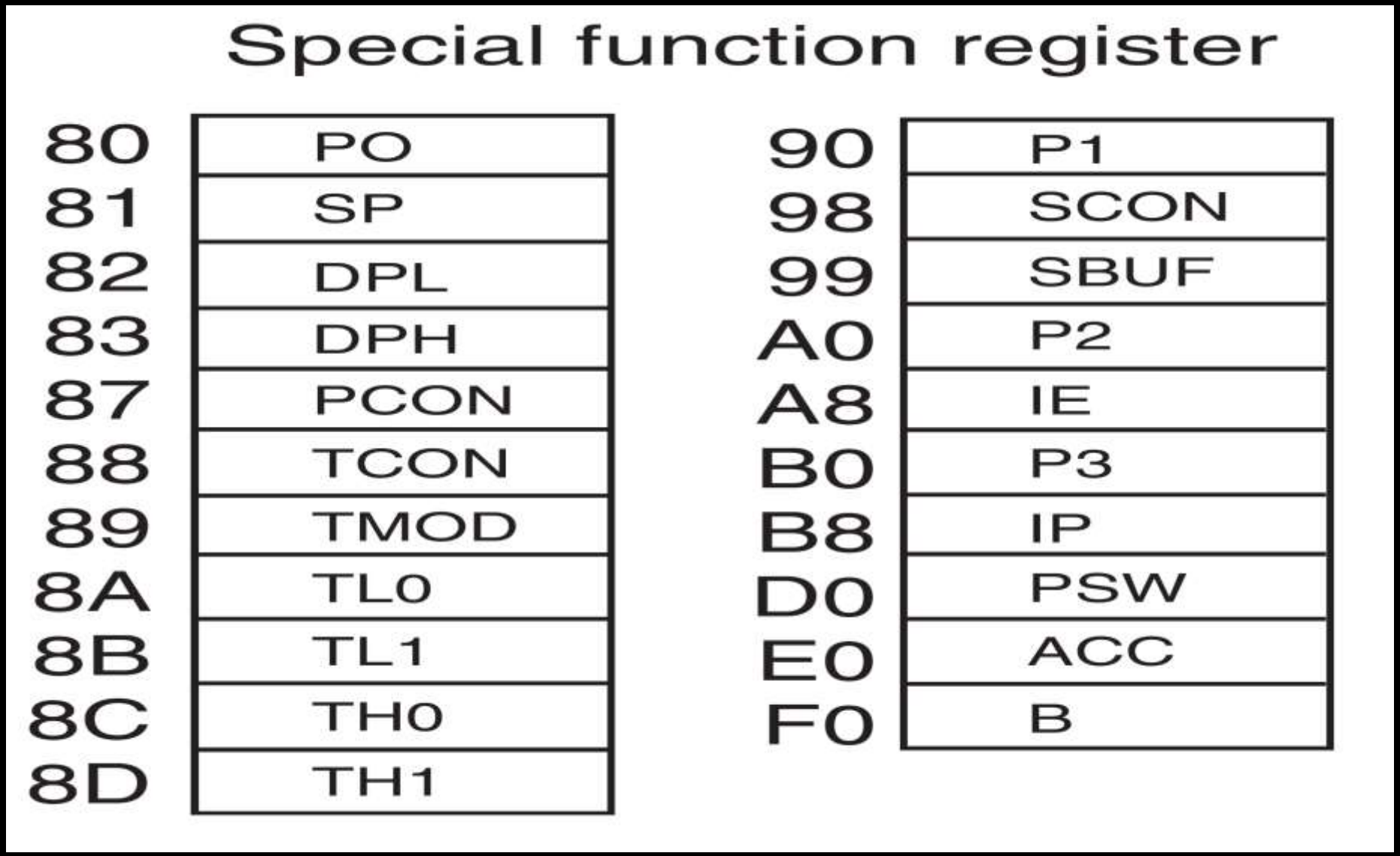
Internal RAM Structure



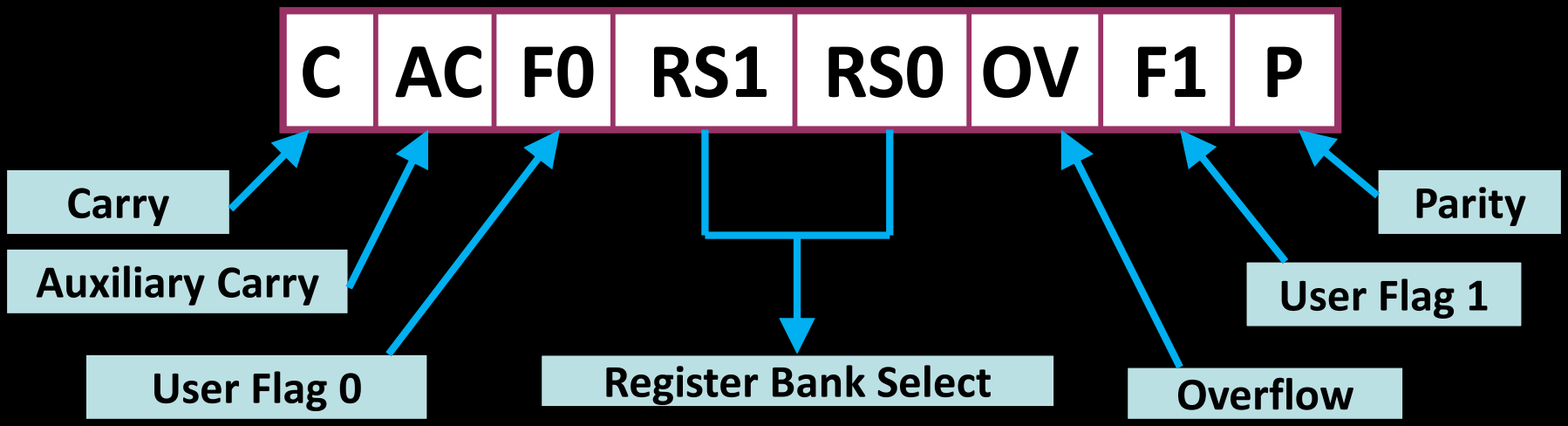
SFR [Special Function Registers]

128 Byte Internal RAM

Special Function Registers [SFR]



Program Status Word [PSW]



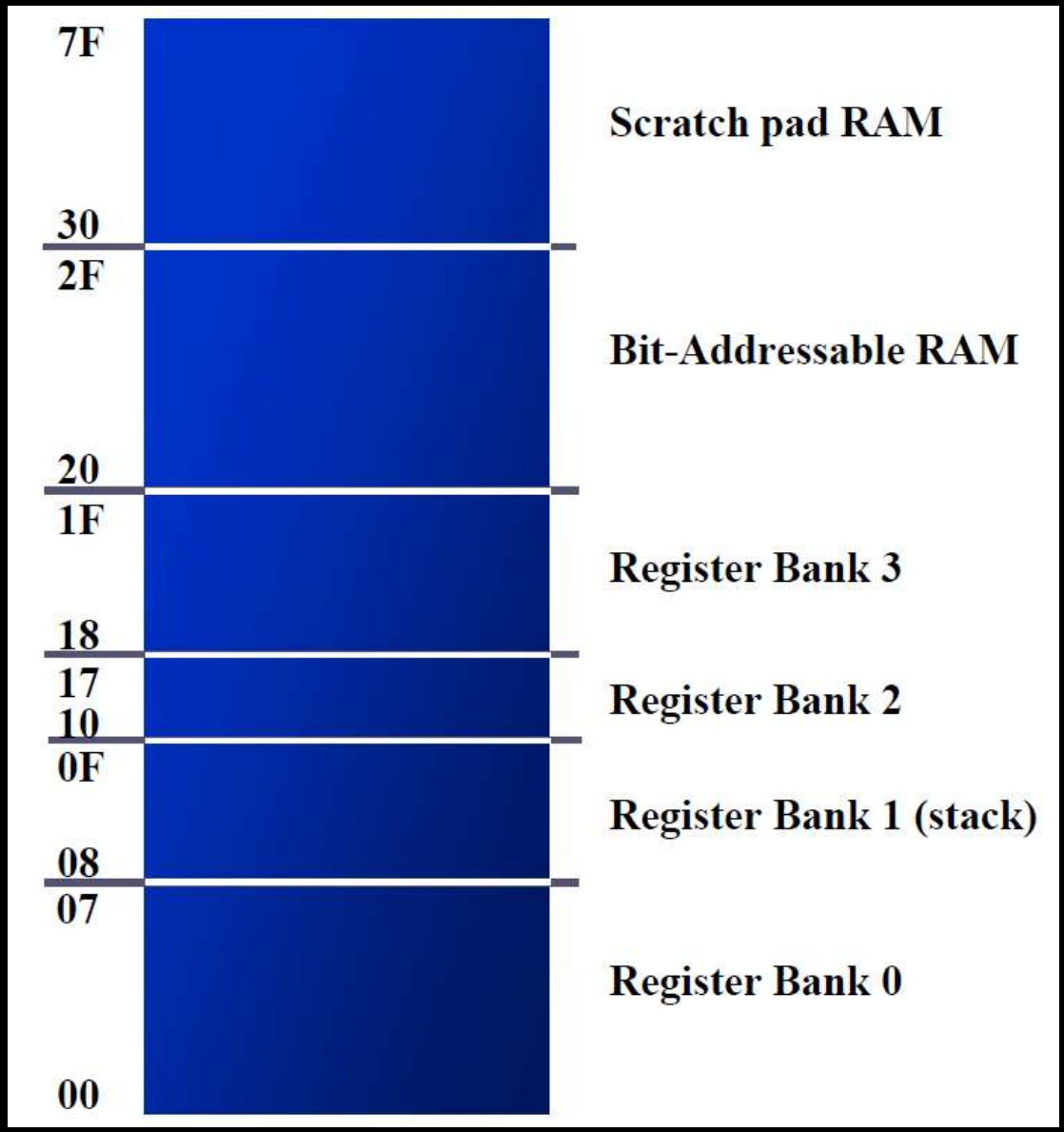
8051 instructions that affects flag

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		

128 Byte RAM

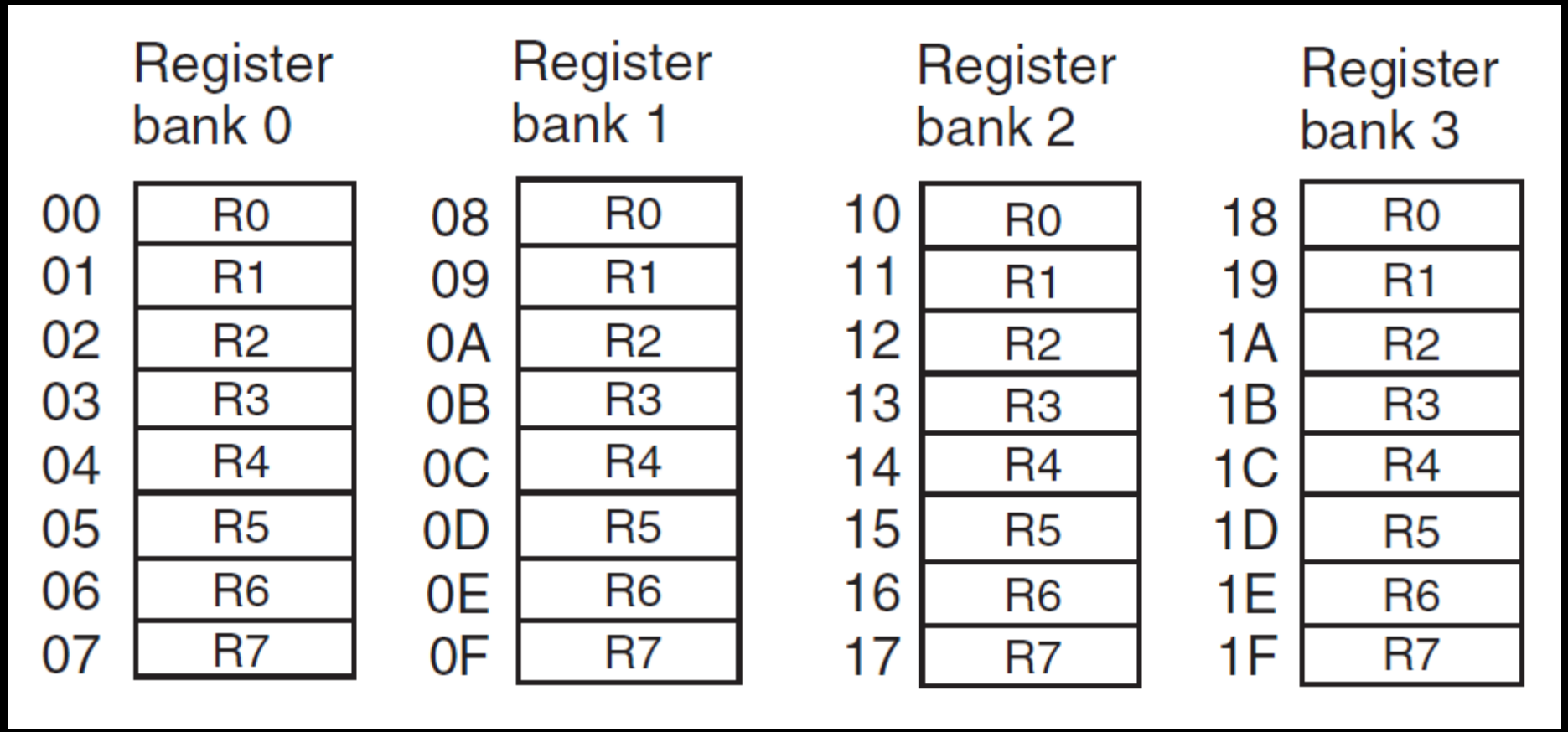
- There are 128 bytes of RAM in the 8051.
  - Assigned addresses 00 to 7FH
- The **128 bytes are divided into 3 different groups** as follows:
  1. A total of **32 bytes** from locations 00 to 1F hex are set aside for ***register banks*** and the ***stack.***
  2. A total of **16 bytes** from locations 20H to 2FH are set aside for ***bit-addressable*** read/write memory.
  3. A total of **80 bytes** from locations 30H to 7FH are used for read and write storage, called ***scratch pad.***

8051 RAM with addresses



8051 Register Bank Structure

8051 Register Banks with address



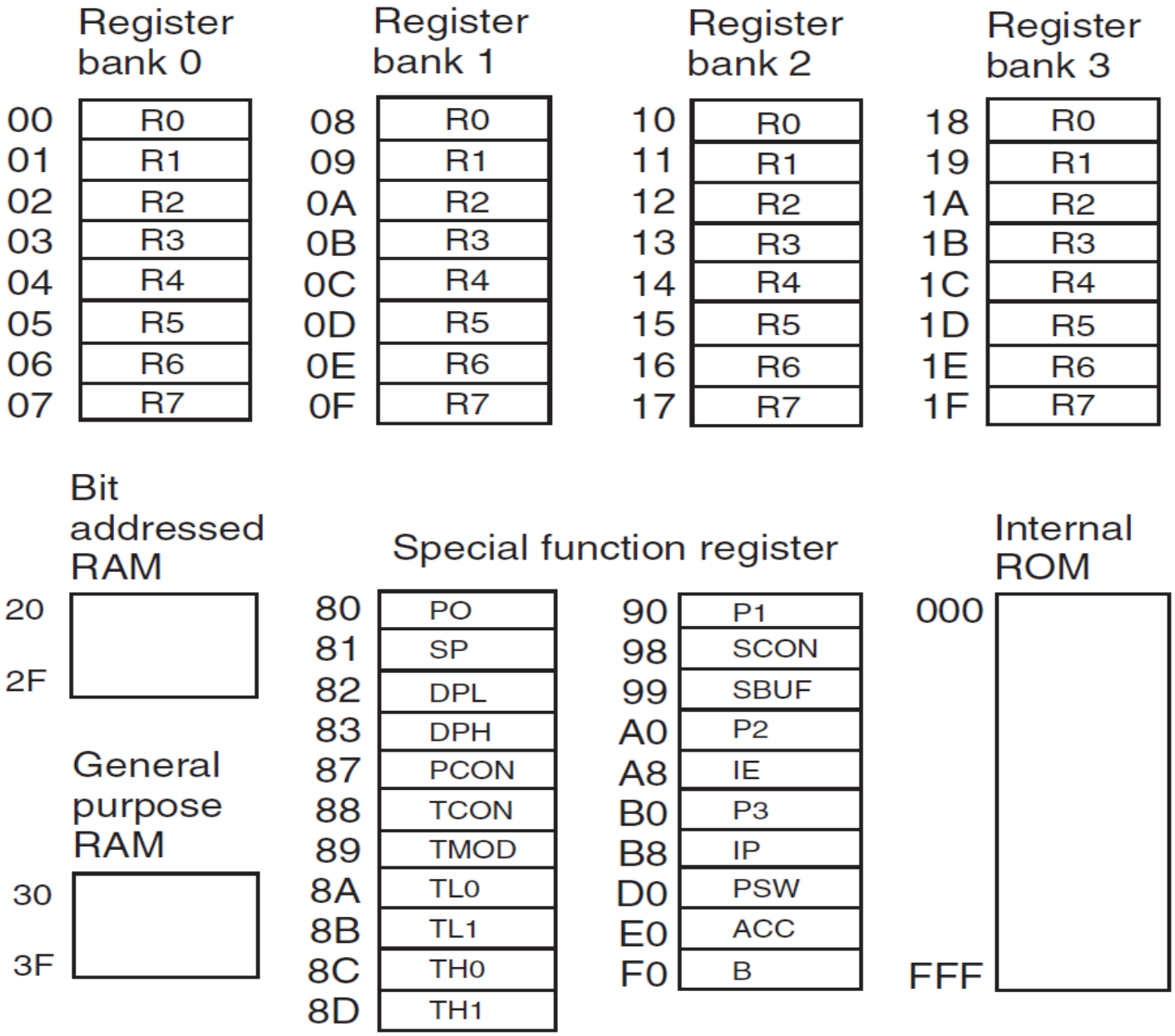


Figure C.3 80C51 programming model

8051 Stack

- The **stack** is a section of RAM used by the CPU to store information **temporarily.**
  - This information could be data or an address

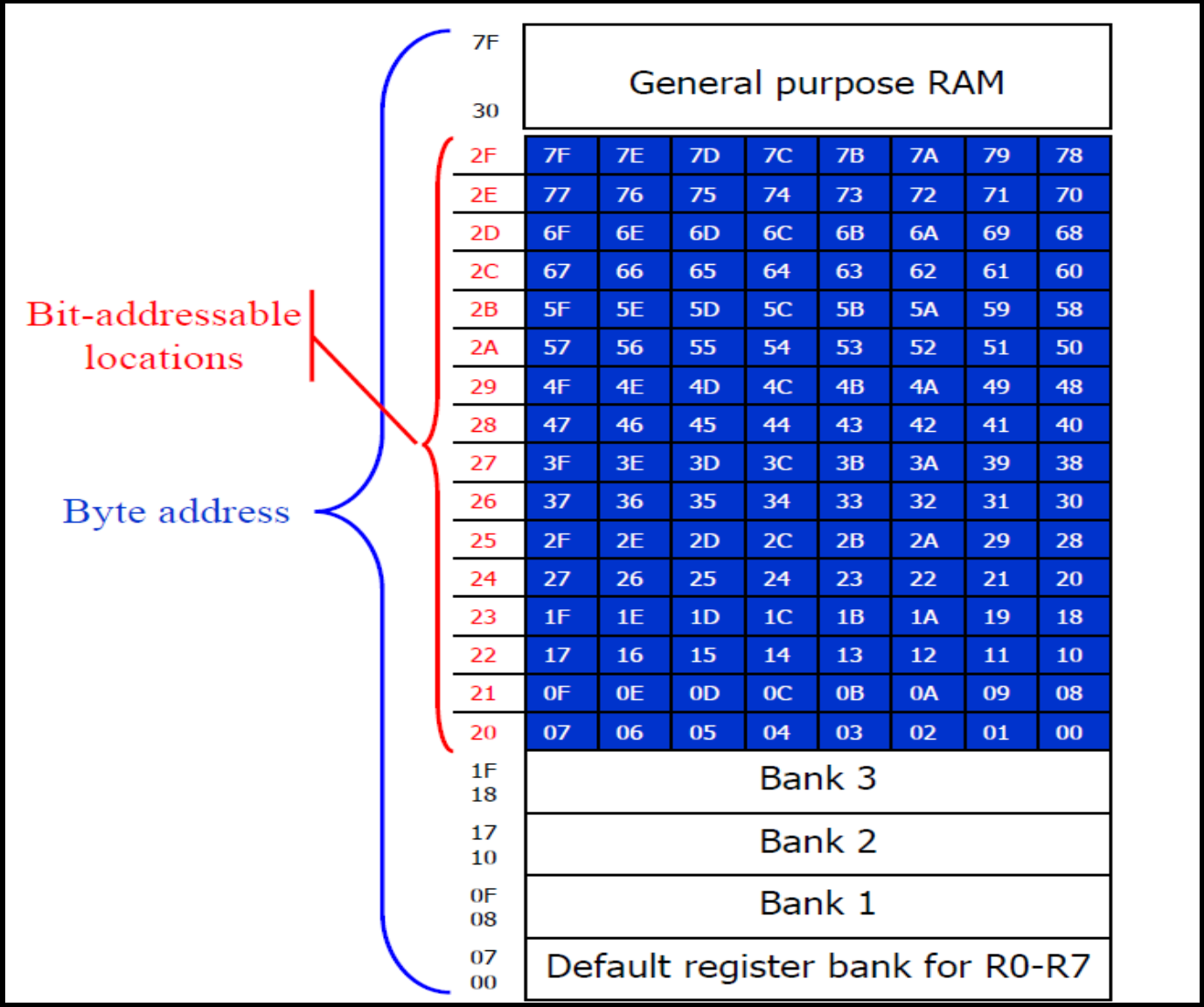
- The register used to access the stack is called the **SP (stack pointer) register**
  - The stack pointer in the 8051 is **only 8 bit wide**, which means that it can take value of 00 to FFH
  - When the 8051 is powered up, the SP register contains value 07
  - RAM location 08 is the first location begin used for the stack by the 8051

8051 Stack

- The storing of a CPU register in the stack is called a **PUSH**
  - SP is pointing to the last used location of the stack
  - As we push data onto the stack, the **SP is incremented** by one
  - This is **different** from many microprocessors

- Loading the contents of the stack back into a CPU register is called a **POP**
  - With every pop, the top byte of the stack is copied to the register specified by the instruction and the stack pointer is **decremented** once

Bit Addressable & Byte Addressable



Single bit Instructions

Single-Bit Instructions

Instruction	Function
SETB bit	Set the bit (bit = 1)
CLR bit	Clear the bit (bit = 0)
CPL bit	Complement the bit (bit = NOT bit)
JB bit, target	Jump to target if bit = 1 (jump if bit)
JNB bit, target	Jump to target if bit = 0 (jump if no bit)
JBC bit, target	Jump to target if bit = 1, clear bit (jump if bit, then clear)

Bit Addressable Programming

- **Example:** Find out to which by each of the following bits belongs. Give the address of the RAM byte in hex

(a) SETB 42H, (b) CLR 67H, (c) CLR 0FH (d) SETB 28H, (e) CLR 12, (f) SETB 05

Solution:

(a) D2 of RAM location 28H

(b) D7 of RAM location 2CH

(c) D7 of RAM location 21H

(d) D0 of RAM location 25H

(e) D4 of RAM location 21H

(f) D5 of RAM location 20H

	D7	D6	D5	D4	D3	D2	D1	D0
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00

8051 Software Overview

1. Addressing Modes
2. Instruction Set
3. Programming

8051 Addressing Modes

- The CPU can access data in various ways, which are called addressing modes

1. **Immediate**
2. **Register**
3. **Direct**
4. **Register indirect**
5. **External Direct**

Immediate Addressing Mode

- The source operand is a **constant.**
- The immediate data must be preceded by the pound sign, **"#"**
- Can load information into **any registers**, including 16-bit DPTR register
  - DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV A,#25H       ;load 25H into A
MOV R4,#62       ;load 62 into R4
MOV B,#40H       ;load 40H into B
MOV DPTR,#4521H  ;DPTR=4512H
MOV DPL,#21H     ;This is the same
MOV DPH,#45H     ;as above

;illegal!! Value > 65535 (FFFFH)
MOV DPTR,#68975
```

Register Addressing Mode

- Use registers to hold the data to be manipulated.

```
MOV A,R0     ;copy contents of R0 into A
MOV R2,A     ;copy contents of A into R2
ADD A,R5     ;add contents of R5 to A
ADD A,R7     ;add contents of R7 to A
MOV R6,A     ;save accumulator in R6
```

- The source and destination registers must match in size. **MOV DPTR,A** will give an error

```
MOV DPTR,#25F5H
MOV R7,DPL
MOV R6,DPH
```

- The movement of data between Rn registers is not allowed **MOV R4,R7** is invalid

Direct Addressing Mode

- It is most often used the direct addressing mode to access RAM locations 30 – 7FH.
- The entire 128 bytes of RAM can be accessed.
- Contrast this with immediate addressing mode, there is no "#" sign in the operand.

```
MOV R0,40H  ;save content of 40H in R0
MOV 56H,A   ;save content of A in 56H
```

SFR Registers & their Addresses

```
MOV   0E0H,#55H     ;is the same as
MOV   A,#55H        ;which means load 55H into A (A=55H)

MOV   0F0H,#25H     ;is the same as
MOV    B,#25H       ;which means load 25H into B (B=25H)

MOV   0E0H,R2       ;is the same as
MOV   A,R2          ;which means copy R2 into A

MOV   0F0H,R0       ;is the same as
MOV   B,R0          ;which means copy R0 into B
```

SFR Addresses (1 of 2)

Table 5-1: Special Function Register (SFR) Addresses

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
TMOD	Timer/counter mode control	89H

SFR Addresses (2 of 2)

TCON*	Timer/counter control	88H
T2CON*	Timer/counter 2 control	0C8H
T2MOD	Timer/counter mode control	0C9H
TH0	Timer/counter 0 high byte	8CH
TL0	Timer/counter 0 low byte	8AH
TH1	Timer/counter 1 high byte	8DH
TL1	Timer/counter 1 low byte	8BH
TH2	Timer/counter 2 high byte	0CDH
TL2	Timer/counter 2 low byte	0CCH
RCAP2H	T/C 2 capture register high byte	0CBH
RCAP2L	T/C 2 capture register low byte	0CAH
SCON*	Serial control	98H
SBUF	Serial data buffer	99H
PCON	Power control	87H

* Bit addressable (discussed further in Chapter 8)

Example

Example 5-1

Write code to send 55H to ports P1 and P2, using (a) their names (b) their addresses.

Solution:

(a)
```
MOV  A,#55H      ;A=55H
MOV  P1,A        ;P1=55H
MOV  P2,A        ;P2=55H
```

(b) From Table 5-1, P1 address = 80H; P2 address = A0H
```
MOV  A,#55H      ;A=55H
MOV  80H,A       ;P1=55H
MOV  0A0H,A      ;P2=55H
```

Stack and Direct Addressing Mode

Show the code to push R5 and A onto the stack and then pop them back them into R2 and B, where B = A and R2 = R5

Solution:

```
PUSH 05          ;push R5 onto stack
PUSH 0E0H        ;push register A onto stack
POP  0F0H        ;pop top of stack into B
                 ;now register B = register A
POP  02          ;pop top of stack into R2
                 ;now R2=R6
```

Register Indirect Addressing Mode

- A **register** is used as a pointer to the data.
- Only register **R0** and **R1** are used for this purpose.
- **R2 – R7** cannot be used to hold the address of an operand located in RAM.
- When **R0** and **R1** hold the addresses of RAM locations, they must be preceded by the **"@"** sign.

```
MOV A,@R0   ;move contents of RAM whose
            ;address is held by R0 into A
MOV @R1,B   ;move contents of B into RAM
            ;whose address is held by R1
```

Register Indirect Addressing Mode

- Write a program to copy the value 55H into RAM memory locations 40H to 41H using **(a) direct addressing mode**, **(b) register indirect addressing mode without a loop**, and **(c) with a loop.**

Solution:

```
(a)
    MOV A,#55H   ;load A with value 55H
    MOV 40H,A    ;copy A to RAM location 40H
    MOV 41H.A    ;copy A to RAM location 41H
(b)
    MOV A,#55H   ;load A with value 55H
    MOV R0,#40H  ;load the pointer. R0=40H
    MOV @R0,A    ;copy A to RAM R0 points to
    INC R0       ;increment pointer. Now R0=41h
    MOV @R0,A    ;copy A to RAM R0 points to
(c)
        MOV A,#55H     ;A=55H
        MOV R0,#40H    ;load pointer.R0=40H,
        MOV R2,#02     ;load counter, R2=3
AGAIN:  MOV @R0,A      ;copy 55 to RAM R0 points to
        INC R0         ;increment R0 pointer
        DJNZ R2,AGAIN  ;loop until counter = zero
```

Register Indirect Addressing Mode

- The advantage is that it makes accessing data dynamic rather than static as in **direct addressing mode**.
- **Looping is not possible in direct addressing mode.**
- Write a program to clear 16 RAM locations starting at RAM address 60H.

```
        CLR A           ;A=0
        MOV R1,#60H     ;load pointer. R1=60H
        MOV R7,#16      ;load counter, R7=16
AGAIN:  MOV @R1,A       ;clear RAM R1 points to
        INC R1          ;increment R1 pointer
        DJNZ R7,AGAIN   ;loop until counter=zero
```

External Direct

- External Memory is accessed.
- There are only two commands that use External Direct addressing mode:
  - **MOVX A, @DPTR**
    MOVX @DPTR, A
- DPTR must first be loaded with the address of external memory.

8051 Instruction Set

- 8051 instructions have 8-bit opcode
- There are 256 possible instructions of which 255 are
- implemented

MOV Instruction

- **MOV destination, source** ; copy source to destination.

- ```
  MOV  A,#55H   ;load value 55H into reg. A
  MOV  R0,A     ;copy contents of A into R0
                   ;(now A=R0=55H)
  MOV  R1,A     ;copy contents of A into R1
                   ;(now A=R0=R1=55H)
  MOV  R2,A     ;copy contents of A into R2
                   ;(now A=R0=R1=R2=55H)
  MOV  R3,#95H  ;load value 95H into R3
                   ;(now R3=95H)
  MOV  A,R3     ;copy contents of R3 into A
                   ;now A=R3=95H
  ```
  ```

ADD Instruction

- **ADD A, source** ;ADD the source operand to the accumulator

- MOV A, #25H ;load 25H into A

  MOV R2,#34H ;load 34H into R2

  ADD A,R2 ;add R2 to accumulator

  ;(A = A + R2)

Structure of Assembly Language

```
        ORG   0H          ;start (origin) at location 0
        MOV  R5,#25H      ;load 25H into R5
        MOV  R7,#34H      ;load 34H into R7
        MOV  A,#0         ;load 0 into A
        ADD  A,R5         ;add contents of R5 to A
                          ;now A = A + R5
        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              ;end of asm source file
```

Data Types & Directives

```
        ORG   500H
DATA1:  DB    28                ;DECIMAL (1C in Hex)
DATA2:  DB    00110101B         ;BINARY (35 in Hex)
DATA3:  DB    39H               ;HEX
        ORG   510H
DATA4:  DB    "2591"            ; ASCII NUMBERS
        ORG   518H
DATA6:  DB    "My name is Joe"  ;ASCII CHARACTERS
```

ADD Instruction and PSW

Example 2-2

Show the status of the CY, AC, and P flags 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 bit
P = 1 since the accumulator has an odd number of 1s (it has five 1s).

ADD Instruction and PSW

Example 2-3

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

```
MOV A,#9CH
ADD A,#64H      ;after addition A=00 and 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 bit
P=0 since the accumulator has an even number of 1s (it has zero 1s).

Multiplication of Unsigned Numbers

```
MUL   AB      ; A × B, place 16-bit result in B and A
```

```
MOV   A,#25H      ;load 25H to reg. A
MOV   B,#65H      ;load 65H in reg. B
MUL   AB          ;25H * 65H = E99 where B = 0EH and A = 99H
```

Table 6-1:Unsigned Multiplication Summary (MUL AB)

Multiplication	Operand 1	Operand 2	Result
byte byte	A	B	A=low byte, B=high byte

Division of Unsigned Numbers

DIV AB ; divide A by B

- MOV A,#95H ;load 95 into A
- MOV B,#10H ;load 10 into B
- DIV AB ;now A = 09 (quotient) and B = 05 (remainder)

Table 6-2:Unsigned Division Summary (DIV AB)

Division	Numerator	Denominator	Quotient	Remainder
byte / byte	A	B	A	B

Checking an input bit

JNB (jump if no bit) ; JB (jump if bit = 1)

Example 8-3

Assume that bit P2.3 is an input and represents the condition of an oven. If it goes high, it means that the oven is hot. Monitor the bit continuously. Whenever it goes high, send a high-to-low pulse to port P1.5 to turn on a buzzer.

Solution:

```
HERE:JNB  P2.3,HERE      ;keep monitoring for high
     SETB P1.5           ;set bit P1.5=1
     CLR  P1.5           ;make high-to-low
```

Switch Register Banks

Example 2-7

State the contents of the RAM locations after the following program:

```
SETB PSW.4      ;select bank 2
MOV R0,#99H     ;load R0 with value 99H
MOV R1,#85H     ;load R1 with value 85H
MOV R2,#3FH     ;load R2 with value 3FH
MOV R7,#63H     ;load R7 with value 63H
MOV R5,#12H     ;load R5 with value 12H
```

Solution:

By default, PSW.3=0 and PSW.4=0; therefore, the instruction "SETB PSW.4" sets RS1=1 and RS0=0, thereby selecting register bank 2. Register bank 2 uses RAM locations 10H - 17H. After the execution of the above program we have the following:

RAM location 10H has value 99H
RAM location 11H has value 85H
RAM location 12H has value 3FH
RAM location 17H has value 63H
RAM location 15H has value 12H

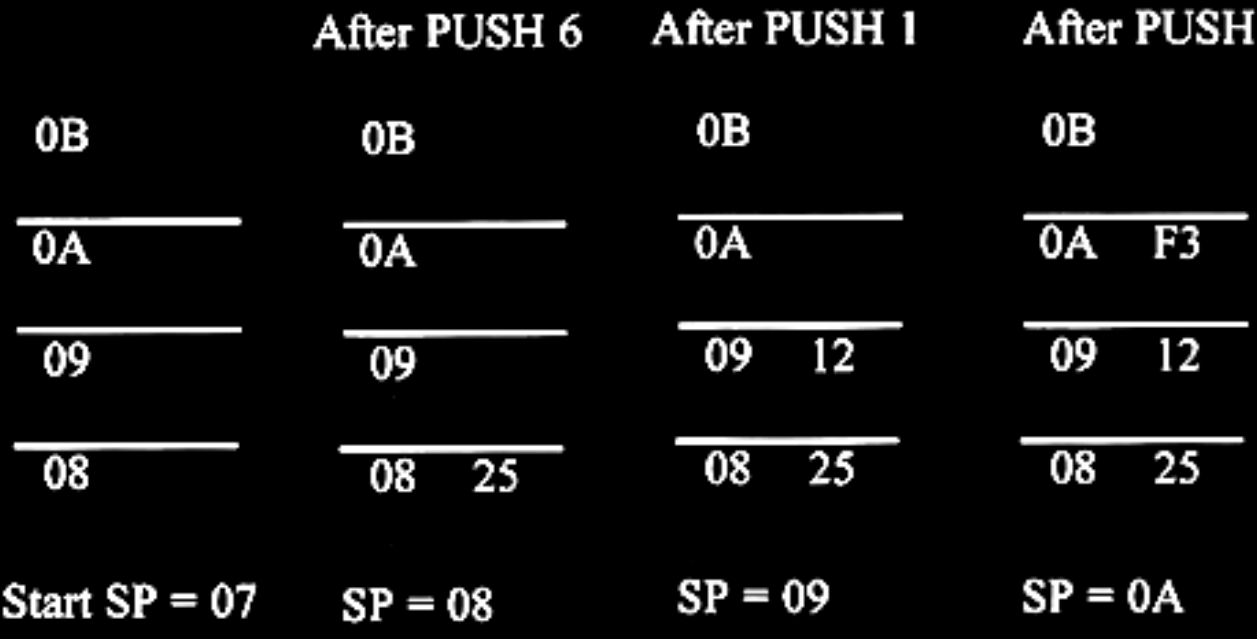
Pushing onto Stack

Example 2-8

Show the stack and stack pointer for the following. Assume the default stack area.

```
MOV  R6,#25H
MOV  R1,#12H
MOV  R4,#0F3H
PUSH 6
PUSH 1
PUSH 4
```

Solution:



Popping from Stack

Example 2-9

Examining the stack, show the contents of the registers and SP after execution of the following instructions. All values are in hex.

```
POP  3     ;POP stack into R3
POP  5     ;POP stack into R5
POP  2     ;POP stack into R2
```

Solution:

	After POP 3	After POP 5	After POP 2
0B 54	0B	0B	0B
0A F9	0A F9	0A	0A
09 76	09 76	09 76	09
08 6C	08 6C	08 6C	08 6C
Start SP = 0B	SP = 0A	SP = 09	SP = 08

Looping

Example 3-1

Write a program to
(a) clear ACC, then
(b) add 3 to the accumulator ten times.

Solution:

```
;This program adds value 3 to the ACC ten times

          MOV   A,#0        ;A=0, clear ACC
          MOV   R2,#10      ;load counter R2=10
AGAIN:    ADD   A,#03       ;add 03 to ACC
          DJNZ  R2,AGAIN    ;repeat until R2=0(10 times)
          MOV   R5,A        ;save A in R5
```

Loop inside a Loop (Nested Loop)

Example 3-3

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times.

Solution:

Since 700 is larger than 255 (the maximum capacity of any register), we use two registers to hold the count. The following code shows how to use R2 and R3 for the count.

```
        MOV   A,#55H      ;A=55H
        MOV   R3,#10      ;R3=10, the outer loop count
NEXT:   MOV   R2,#70      ;R2=70, the inner loop count
AGAIN:  CPL   A           ;complement A register
        DJNZ  R2,AGAIN    ;repeat it 70 times (inner loop)
        DJNZ  R3,NEXT
```

In this program, R2 is used to keep the inner loop count. In the instruction "`DJNZ R2,AGAIN`", whenever R2 becomes 0 it falls through and "`DJNZ R3,NEXT`" is executed. This instruction forces the CPU to load R2 with the count 70 and the inner loop starts again. This process will continue until R3 becomes zero and the outer loop is finished.

8051 Conditional Jump Instructions

Table 3-1: 8051 Conditional Jump Instructions

Instruction	Action
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and jump if A ≠ 0
CJNE A,byte	Jump if A ≠ byte
CJNE reg,#data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

Conditional Jump

- The 8051 offers a variety of conditional jump instructions
- JZ and JNZ tests the accumulator for a particular condition
- DJNZ (decrement and jump if not zero) is a useful instruction for building loops
- To execute a loop N times, load a register with N and terminate the loop with a DJNZ to the beginning of the loop
- CJNE (compare and jump if not equal) is another conditional jump instruction
- CJNE: two bytes in the operand field are taken as unsigned integers. If the first one is less than the second one, the carry is set
- Example: It is desired to jump to BIG if the value of the accumulator is greater than or equal to 20_H

  CJNE A,#20_H,$+3

  JNC BIG

  – $ is an assembler symbol representing the address of the current instruction
  – Since CJNE is a 3-byte instruction, $+3 is the address of next instruction JNC

Conditional Jump Example

Example 3-4

Write a program to determine if R5 contains the value 0. If so, put 55H in it.

Solution:

```
        MOV  A,R5       ;copy R5 to A
        JNZ  NEXT       ;jump if A is not zero
        MOV R5,#55H
NEXT:   ...
```

Conditional Jump Example

Example 3-5

Find the sum of the values 79H, F5H, and E2H. Put the sum in registers R0 (low byte) and R5 (high byte).

Solution:

```
        MOV   A,#0        ;clear A(A=0)
        MOV   R5,A        ;clear R5
        ADD   A,#79H      ;A=0+79H=79H
        JNC   N_1         ;if no carry, add next number
        INC   R5          ;if CY=1, increment R5
N_1:    ADD   A,#0F5H     ;A=79+F5=6E and CY=1
        JNC   N_2         ;jump if CY=0
        INC   R5          ;If CY=1 then increment R5(R5=1)
N_2:    ADD   A,#0E2H     ;A=6E+E2=50 and CY=1
        JNC   OVER        ;jump if CY=0
        INC   R5          ;if CY=1, increment 5
OVER:MOV  R0,A            ;Now R0=50H, and R5=02
```

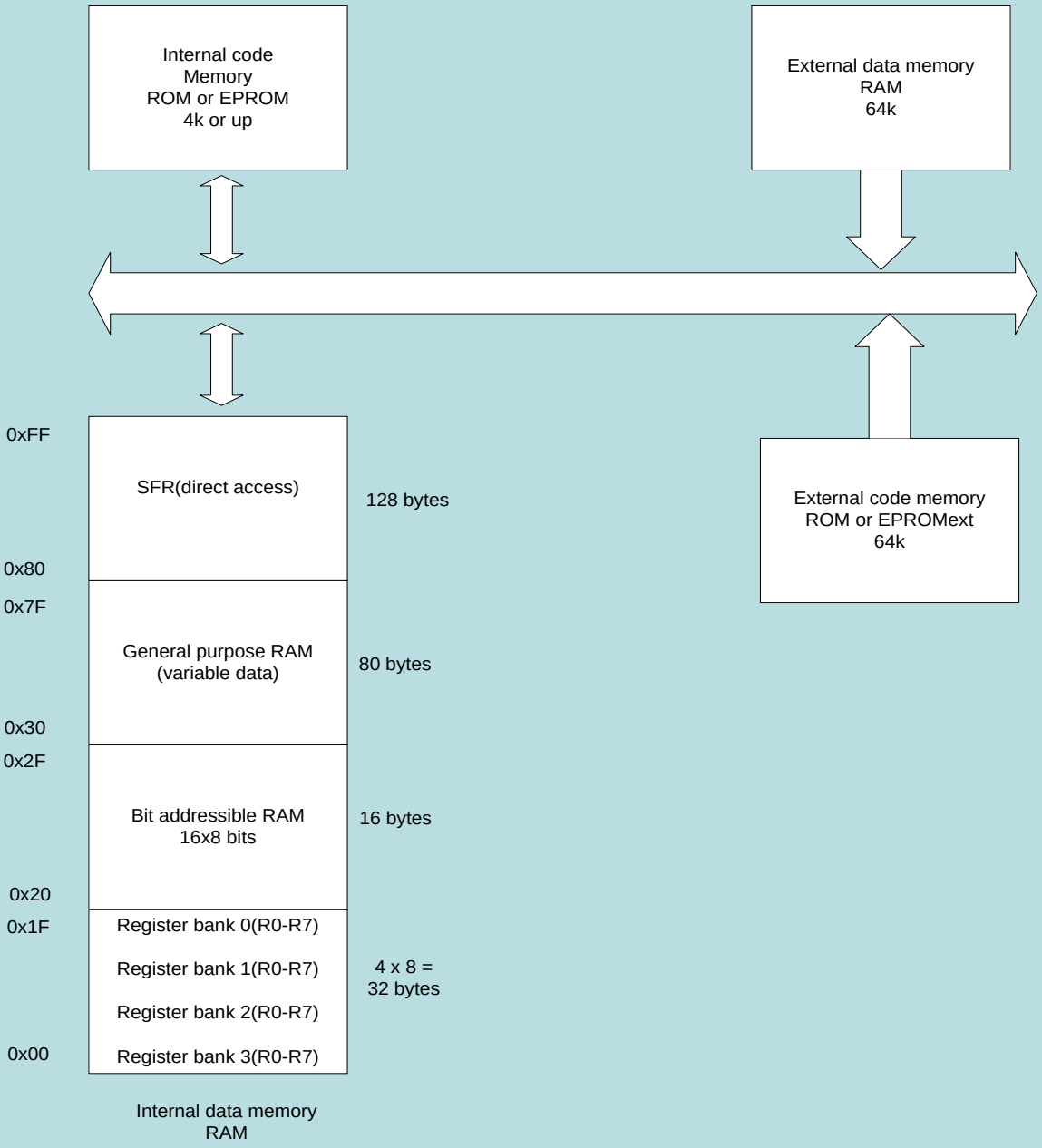
Unconditional Jump Instructions

- All conditional jumps are short jumps
  - Target address within -128 to +127 of PC
- **LJMP** (long jump): 3-byte instruction
  - 2-byte target address: 0000 to FFFFH
  - Original 8051 has only 4KB on-chip ROM
- **SJMP** (short jump): 2-byte instruction
  - 1-byte relative address: -128 to +127

Call Instructions

- LCALL (long call): 3-byte instruction
  - 2-byte address
  - Target address within 64K-byte range
- ACALL (absolute call): 2-byte instruction
  - 11-bit address
  - Target address within 2K-byte range

Separate read instructions for external data and code memory.



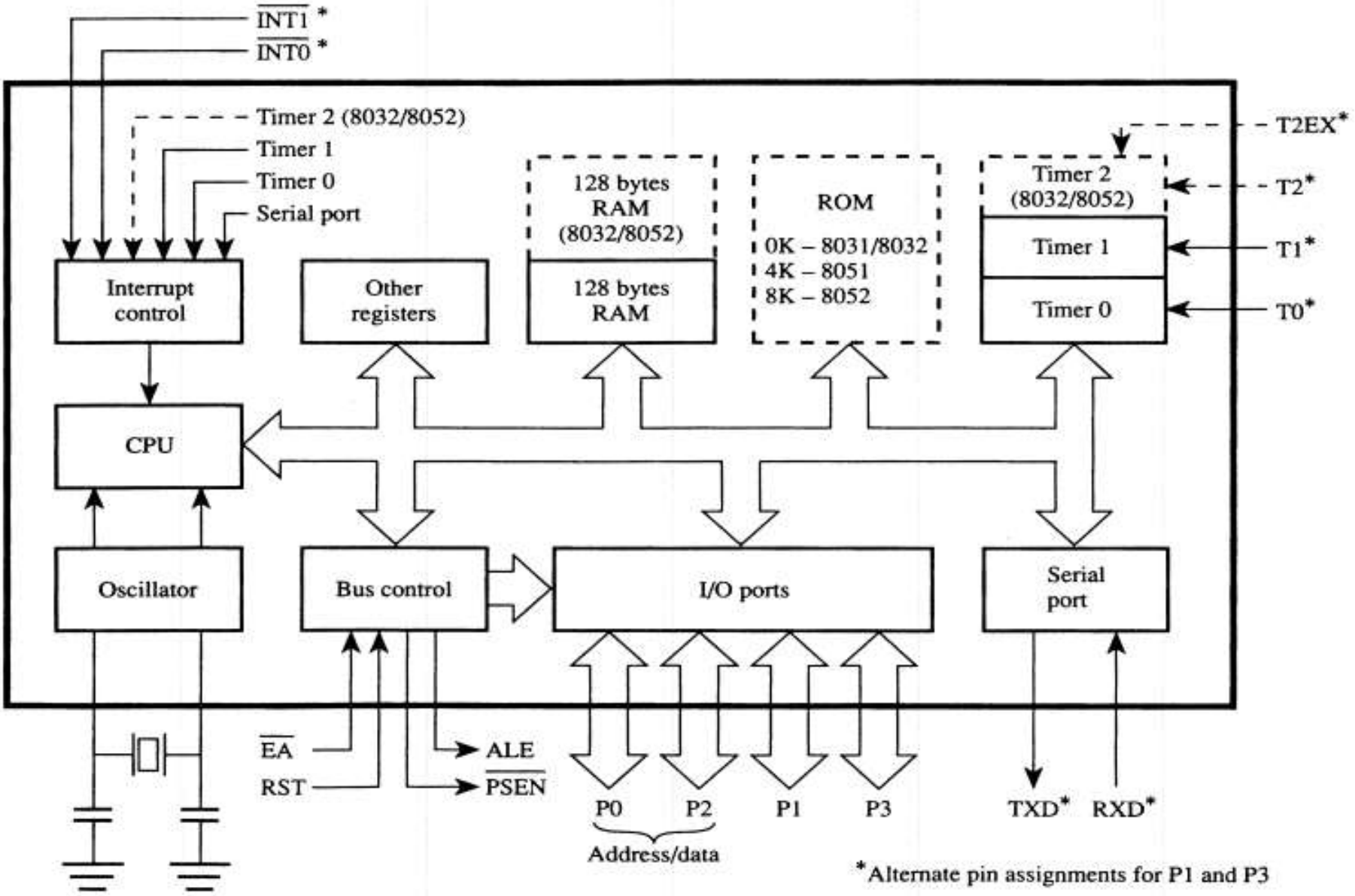


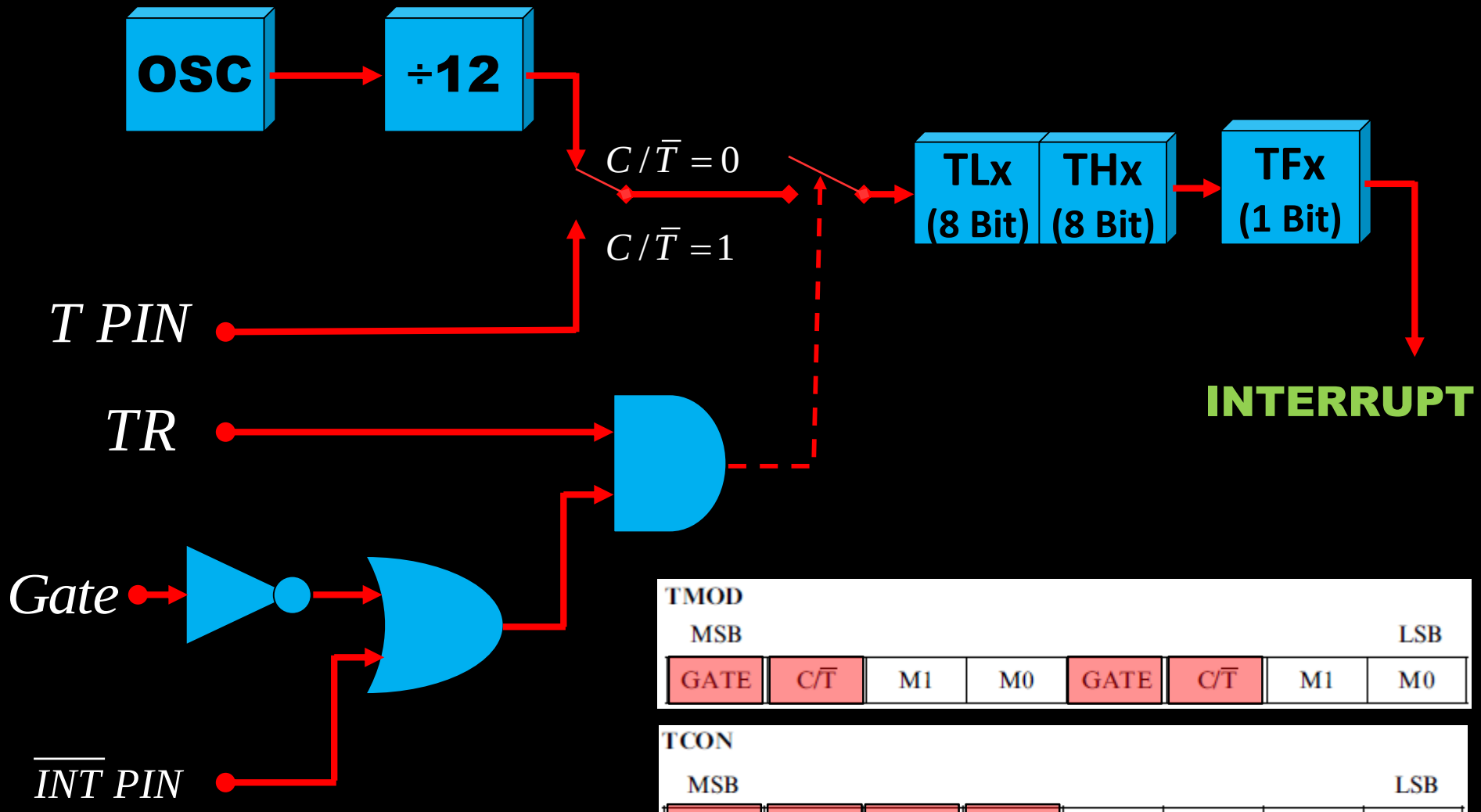
FIGURE 2–1
8051 block diagram

8051 Peripheral Overview

1. Timers
2. Serial Port
3. Interrupts

8051 TIMERS

8051 Timer/Counter



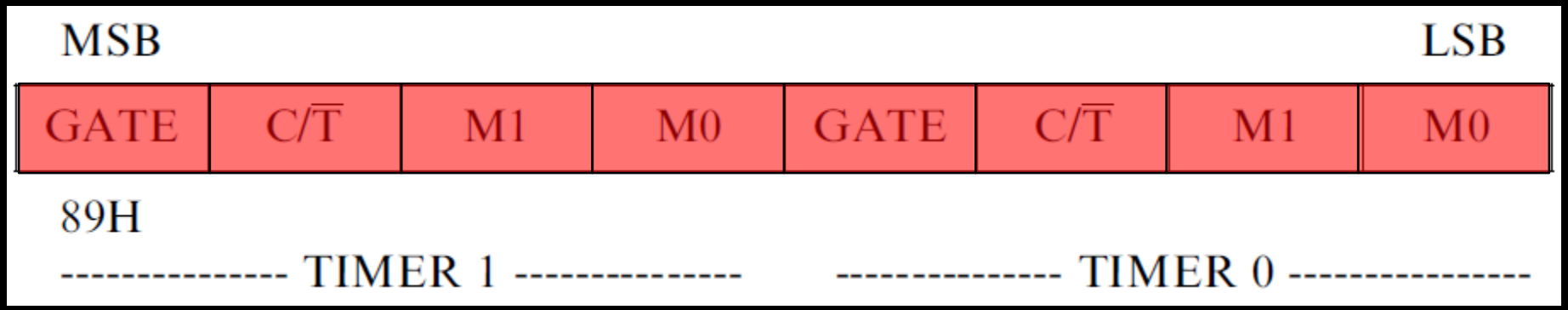
TMOD

MSB							LSB
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0

TCON

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TMOD Register



GATE:

When set, timer/counter x is enabled, **if** INTx pin is high and TRx is set.

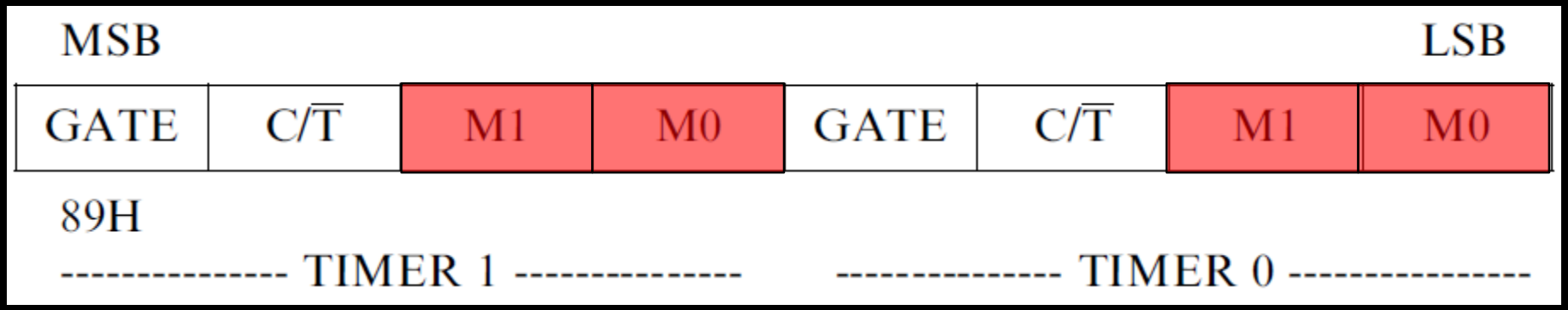
When cleared, timer/counter x is enabled, **if** TRx bit set.

C/T*:

When set, counter operation (input from Tx input pin).

When cleared, timer operation (input from internal clock).

TMOD Register



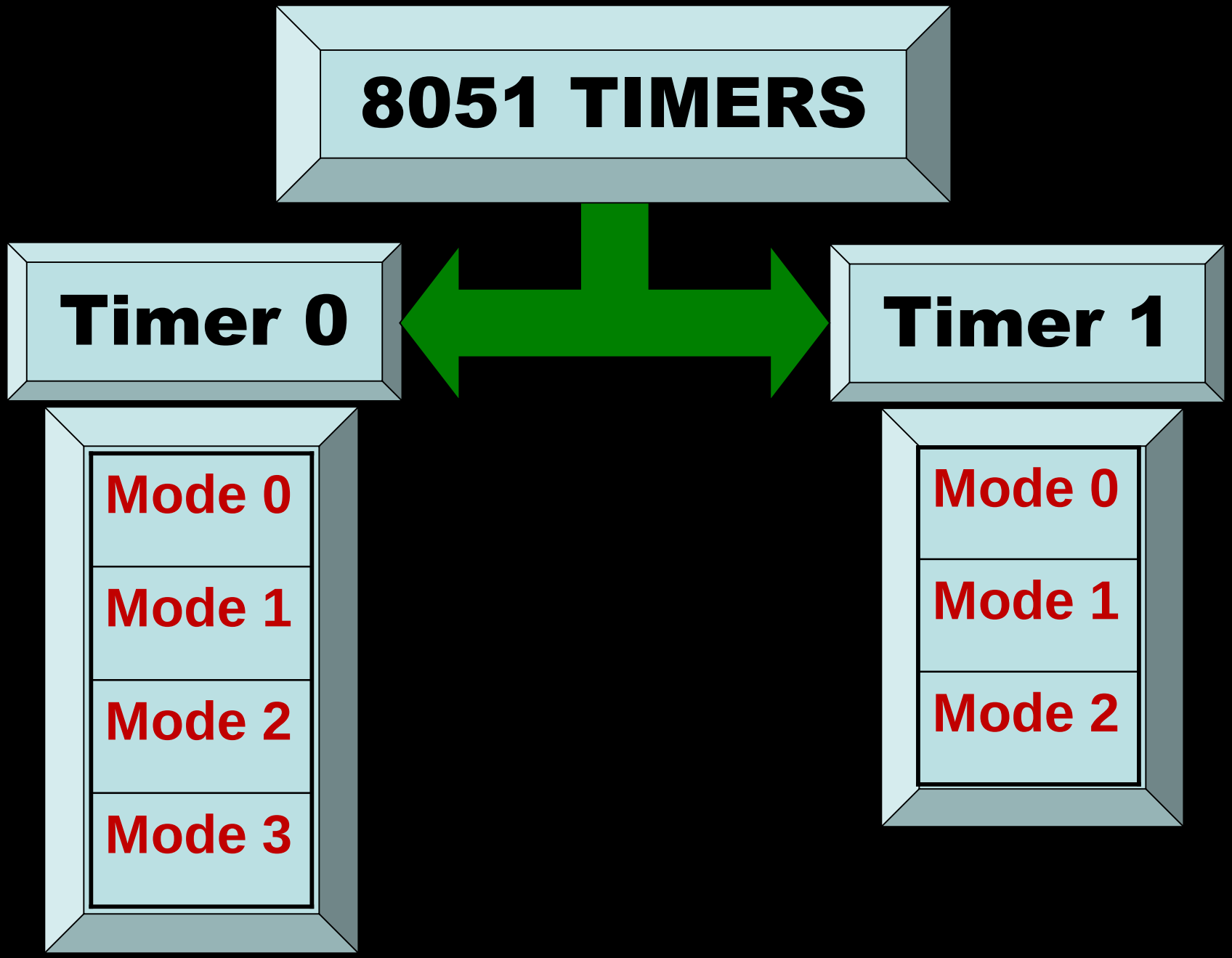
The TMOD byte is not bit addressable.

M1	M0	Operation
0	0	8048 8-bit timer TLx serves as 5-bit prescaler
0	1	16-bit timer/counter. THx and TLx are cascaded. No prescaler
1	0	8-bit autoreload timer/counter. THx contents loaded into TLx when it overflows
1	1	TL0 is 8-bit counter controlled by timer 0 control bits. TH0 is 8-bit timer controlled by timer 1 control bits
1	1	Timer 1 off

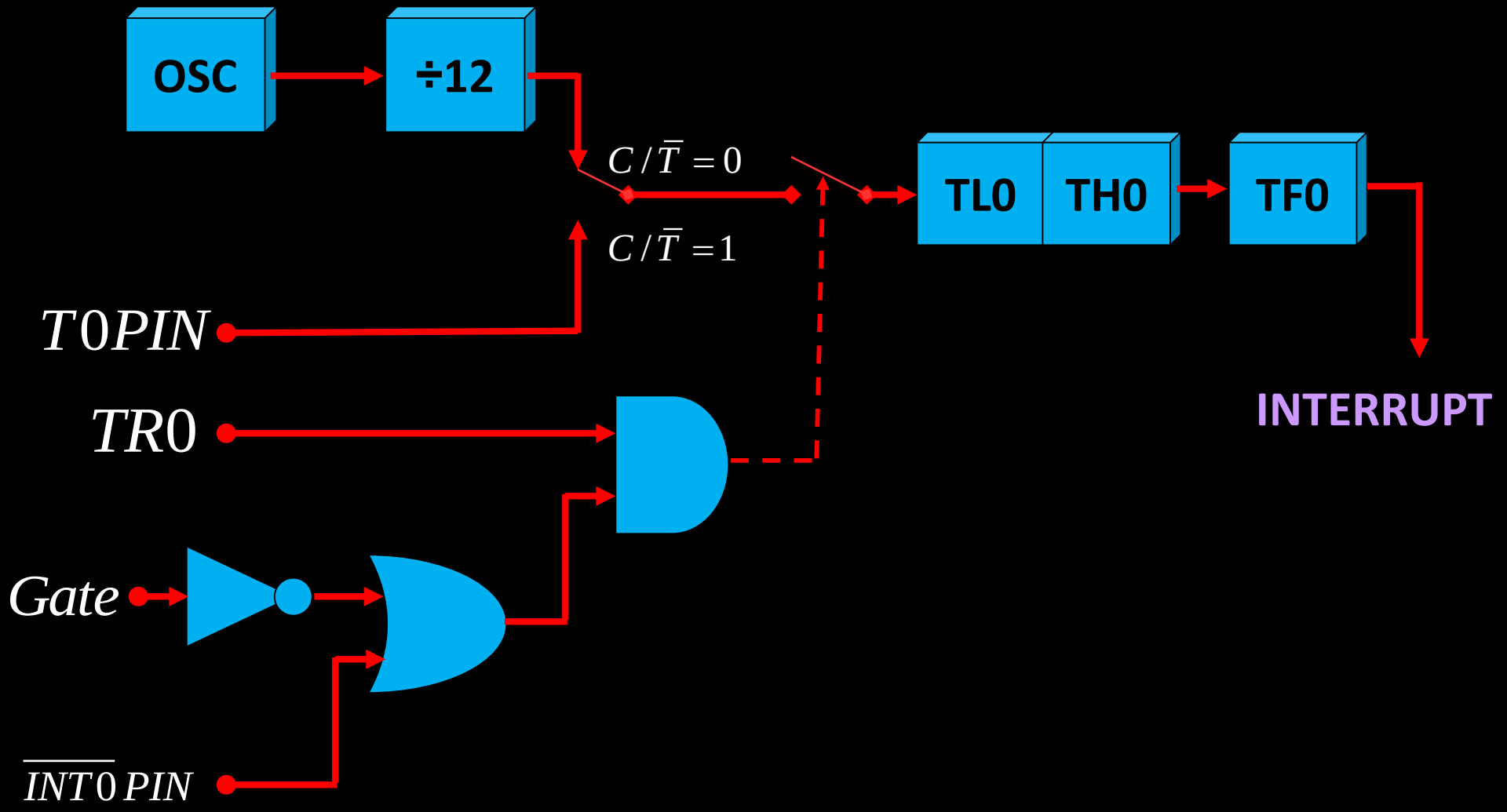
TCON Register

MSB							LSB
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
8FH	8EH	8DH	8CH	8BH	8AH	89H	88H

Bit	Function
TF1/0	Timer 1/0 overflow flag. Set by hardware on timer/counter overflow. Cleared when CPU vectors to interrupt routine
TR1/0	Timer 1/0 run control bit. Set/cleared by software to turn timer/counter on/off
IE1/0	Interrupt 1/0 edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed
IT1/0	Interrupt 1/0 control bit. Set/cleared by software to specify falling edge/ low level triggered external interrupts

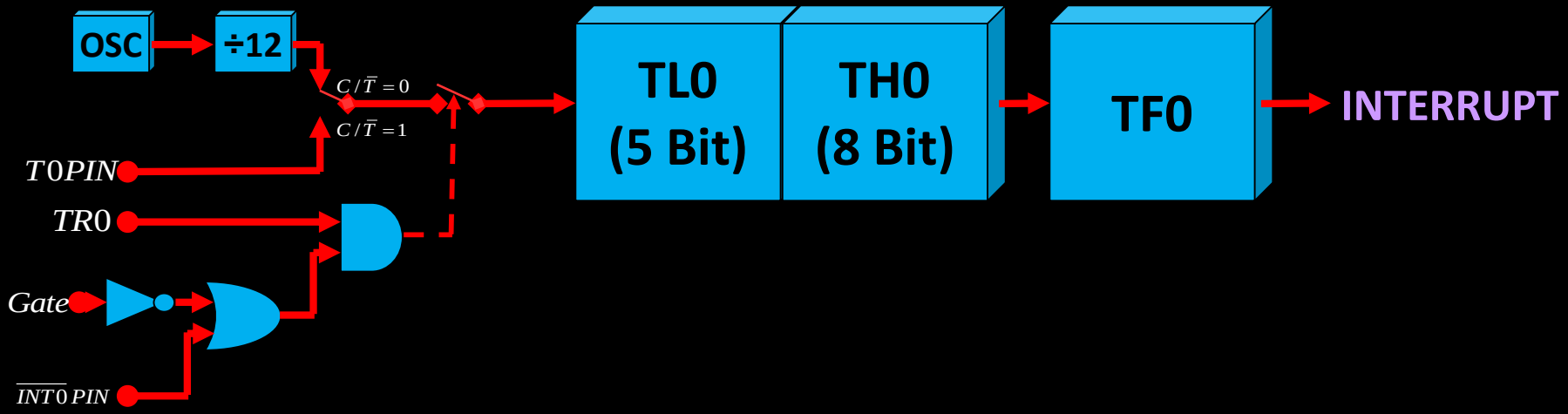

8051 TIMERS
Timer 0
Timer 1
Mode 0
Mode 1
Mode 2
Mode 3
Mode 0
Mode 1
Mode 2

TIMER 0


OSC
÷12
$C/\overline{T}=0$
$C/\overline{T}=1$
TL0
TH0
TF0
INTERRUPT
$T0PIN$
$TR0$
$Gate$
$\overline{INT0}\,PIN$

TIMER 0 – Mode 0

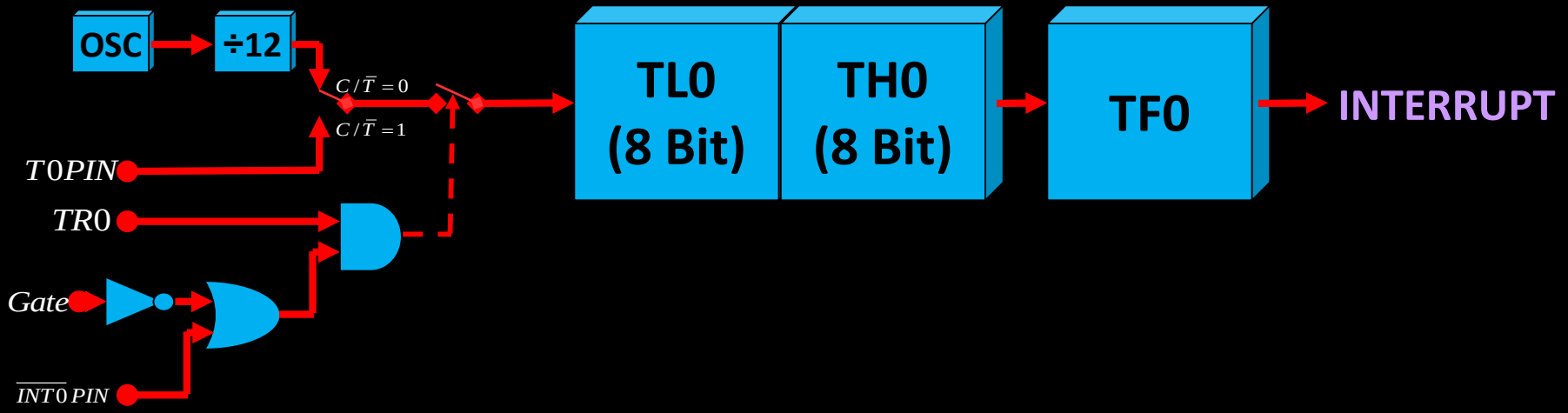
13 Bit Timer / Counter



Maximum Count = 1FFFh (0001111111111111)

TIMER 0 – Mode 1

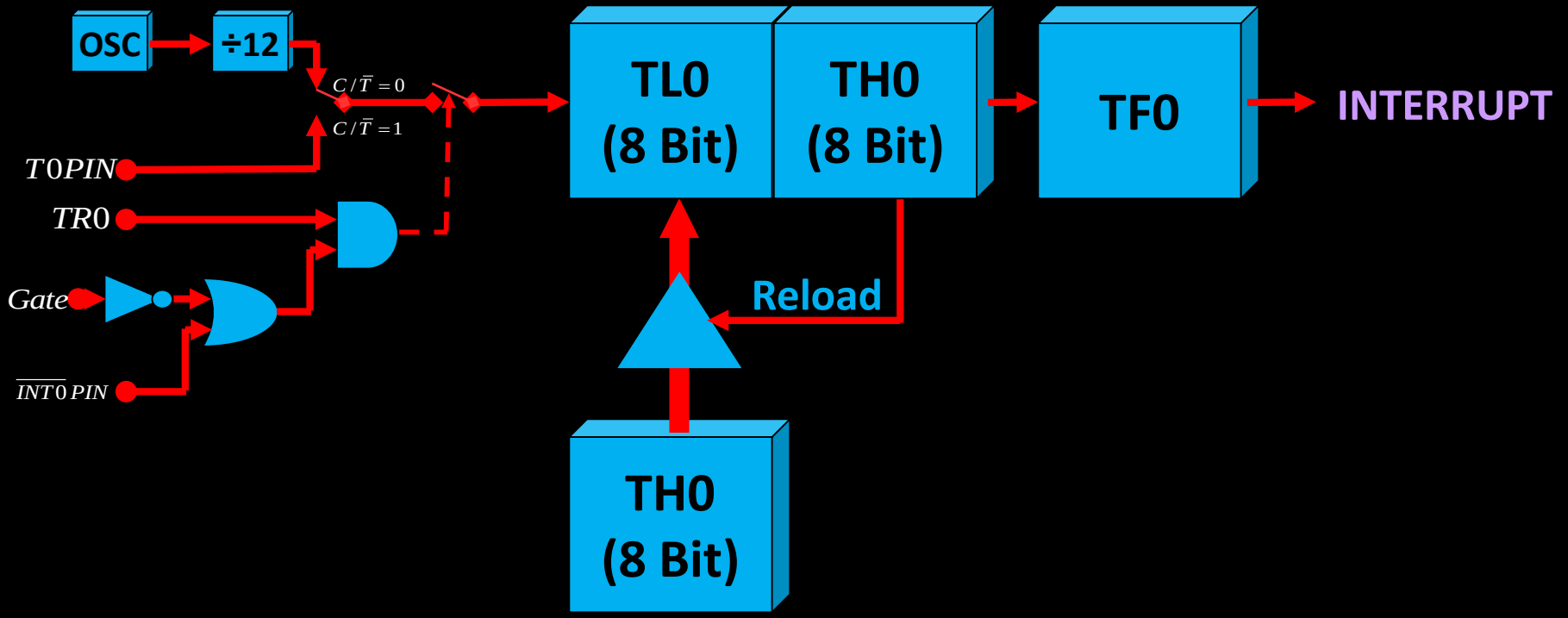
16 Bit Timer / Counter



Maximum Count = FFFFh (1111111111111111)

TIMER 0 – Mode 2

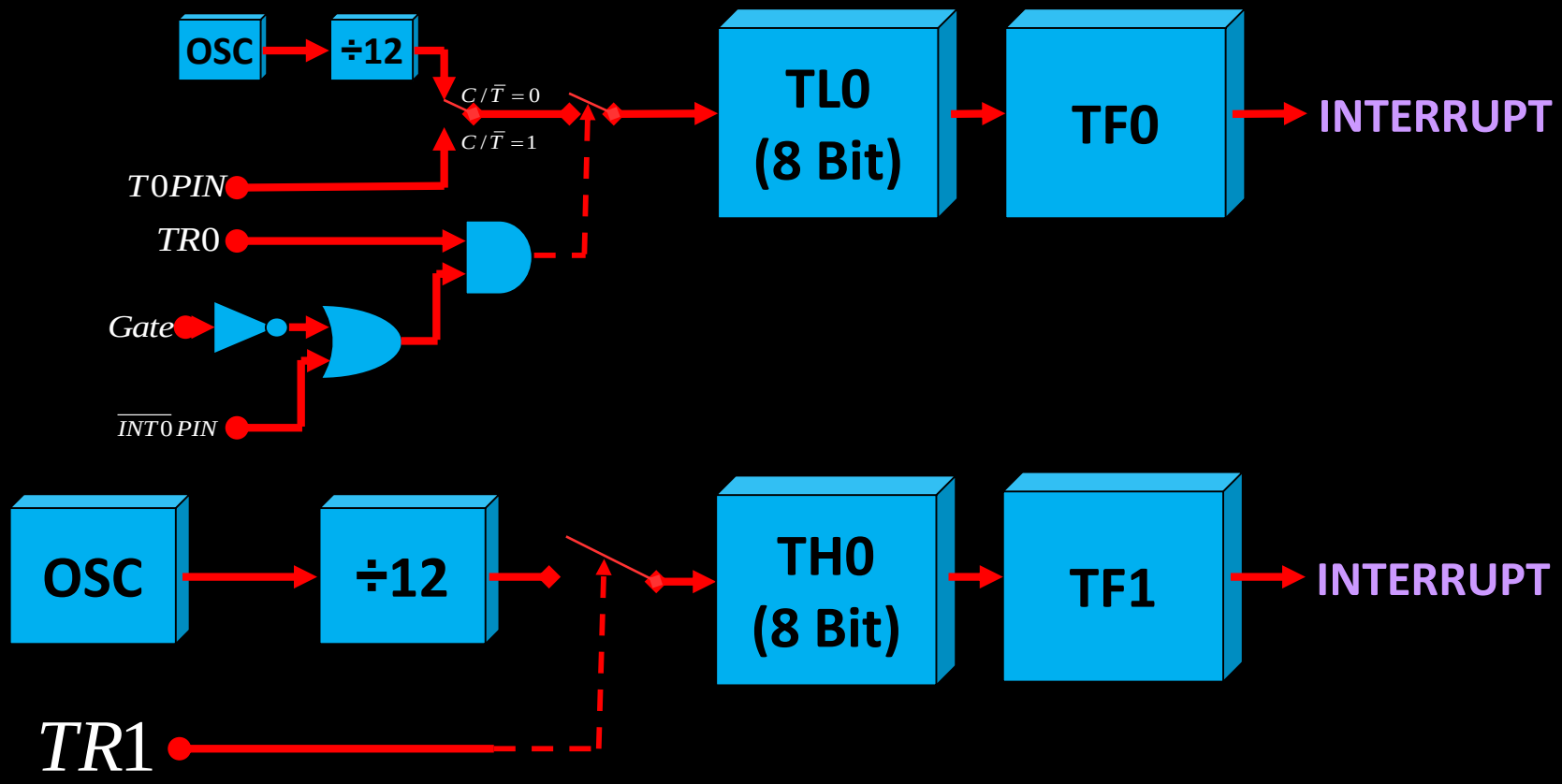
8 Bit Timer / Counter with AUTORELOAD



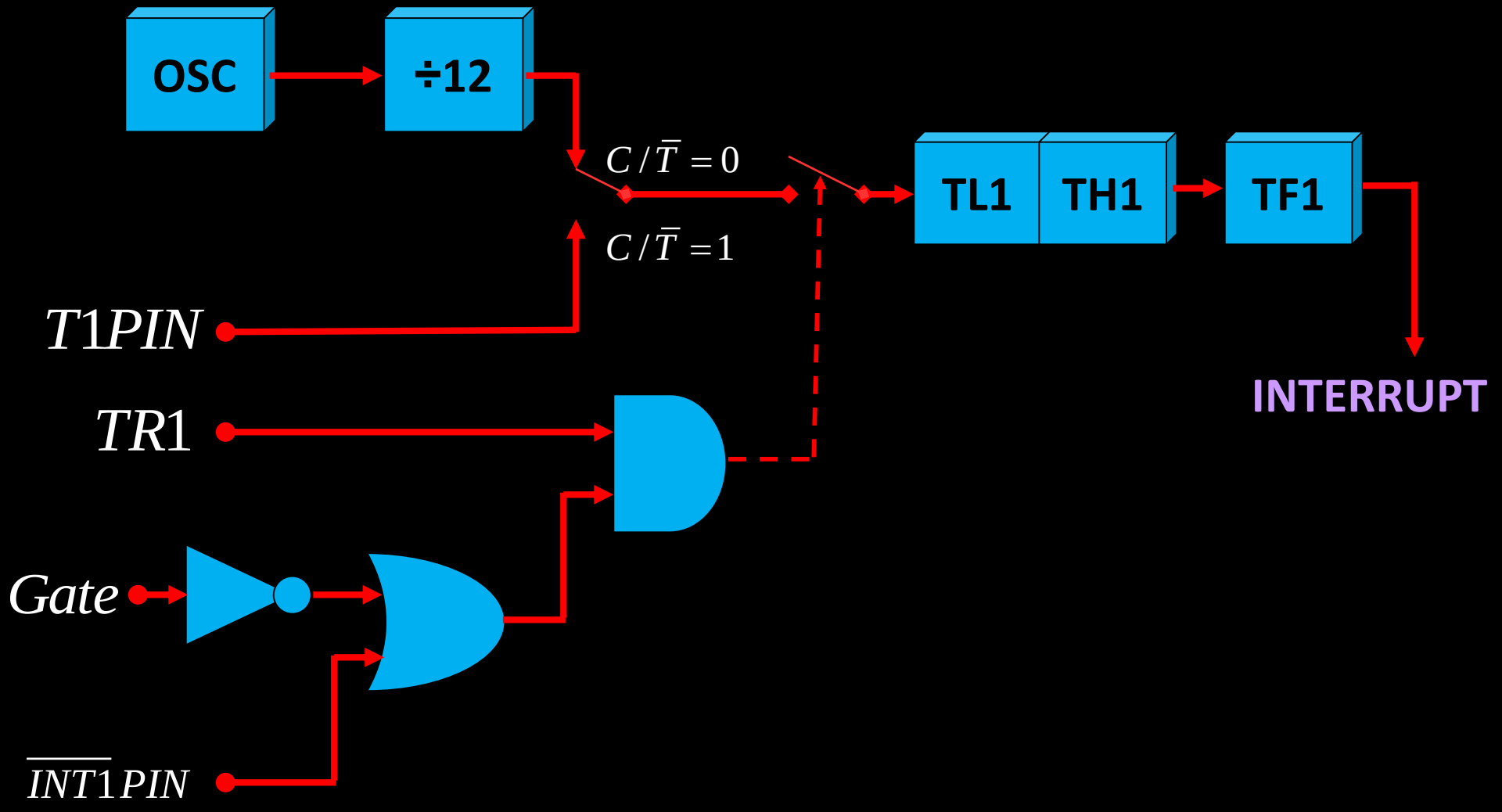
Maximum Count = FFh (11111111)

TIMER 0 – Mode 3

Two - 8 Bit Timer / Counter

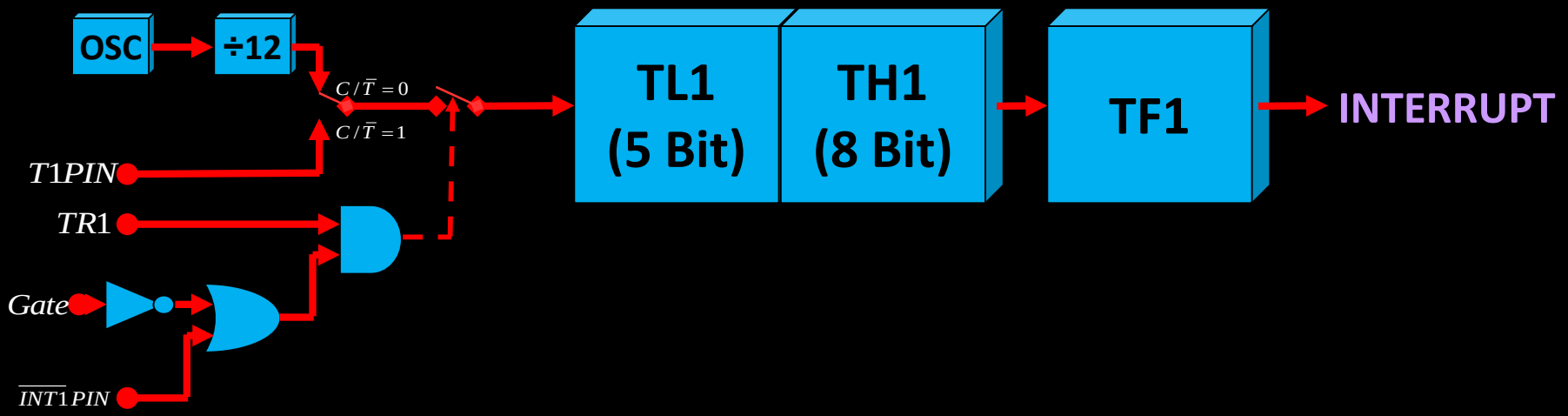


TIMER 1


OSC
÷12
$C/\overline{T}=0$
$C/\overline{T}=1$
TL1
TH1
TF1
INTERRUPT
T1PIN
TR1
Gate
$\overline{INT1}$ PIN

TIMER 1 – Mode 0

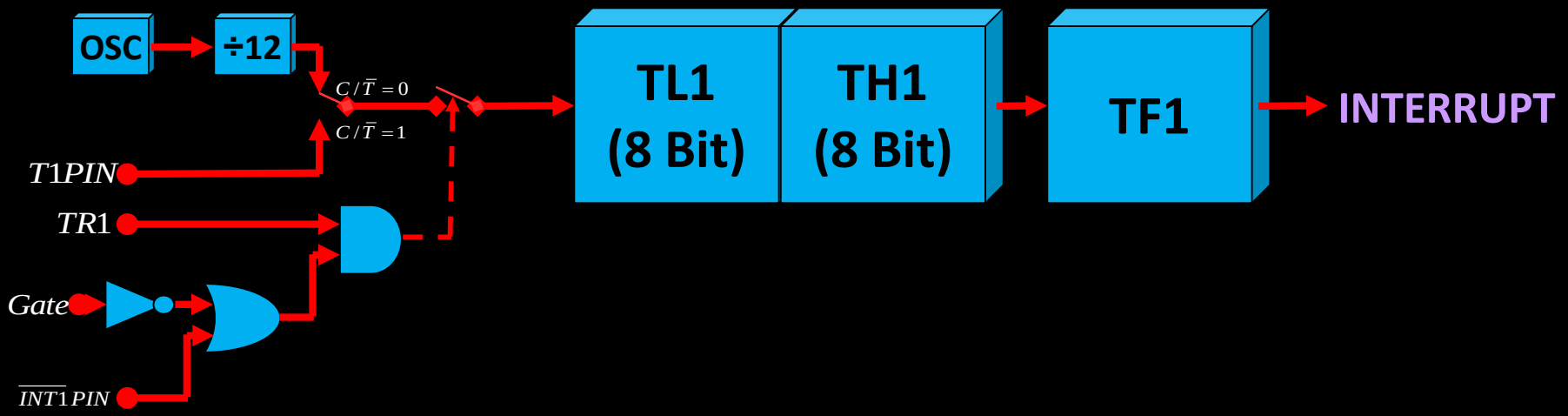
13 Bit Timer / Counter



Maximum Count = 1FFFh (0001111111111111)

TIMER 1 – Mode 1

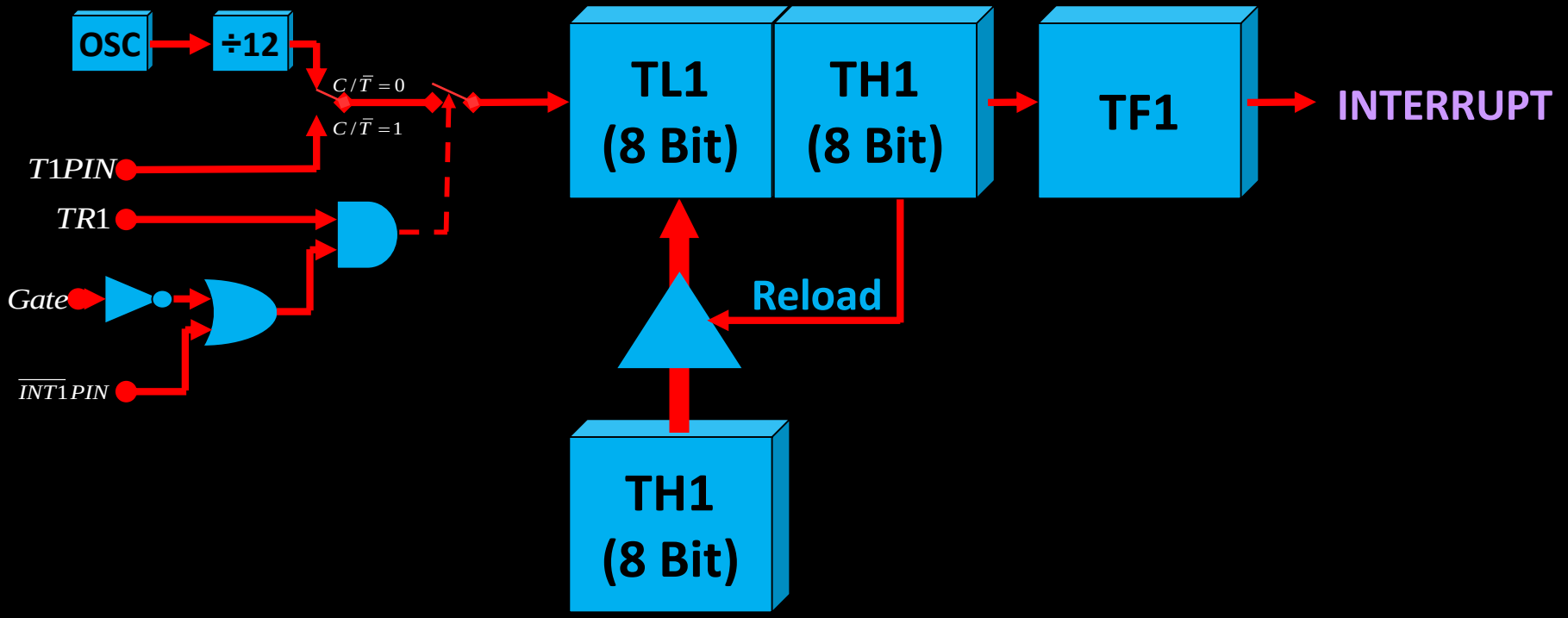
16 Bit Timer / Counter



Maximum Count = FFFFh (1111111111111111)

TIMER 1 – Mode 2

8 Bit Timer / Counter with AUTORELOAD



Maximum Count = FFh (11111111)

Programming Timers

- **Example:** Indicate which mode and which timer are selected for each of the following.

  (a) MOV TMOD, #01H (b) MOV TMOD, #20H (c) MOV TMOD, #12H

- **Solution:** We convert the value from hex to binary.

  (a) TMOD = 00000001, mode 1 of timer 0 is selected.

  (b) TMOD = 00100000, mode 2 of timer 1 is selected.

  (c) TMOD = 00010010, mode 2 of timer 0, and mode 1 of timer 1 are selected.

Programming Timers

- Find the timer's clock frequency and its period for various 8051-based system, with the crystal frequency 11.0592 MHz when C/T bit of TMOD is 0.

- **Solution:**

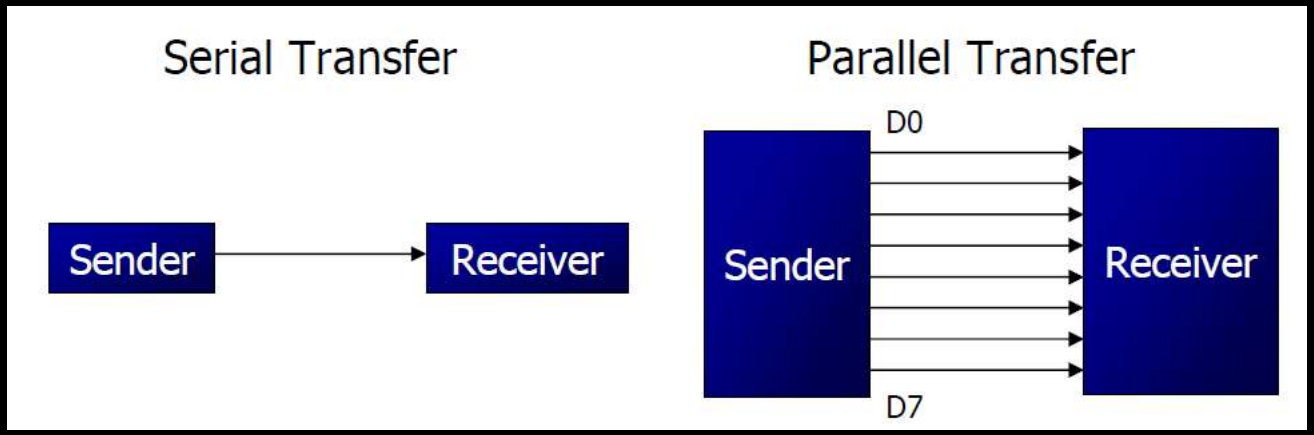
1/12 × 11.0529 MHz = 921.6 MHz;

T = 1/921.6 kHz = 1.085 us

8051 Serial Port

Basics of Serial Communication

- Computers transfer data in **two** ways:
  - **Parallel:** Often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away.
  - **Serial:** To transfer to a device located many meters away, the serial method is used. The data is sent one bit at a time.



Basics of Serial Communication

- Serial data communication uses **two** methods
  - **Synchronous** method transfers a block of data at a time
  - **Asynchronous** method transfers a single byte at a time

- There are **special IC's** made by many manufacturers for serial communications.
  - **UART** (universal asynchronous Receiver transmitter)
  - **USART** (universal synchronous-asynchronous Receiver-transmitter)

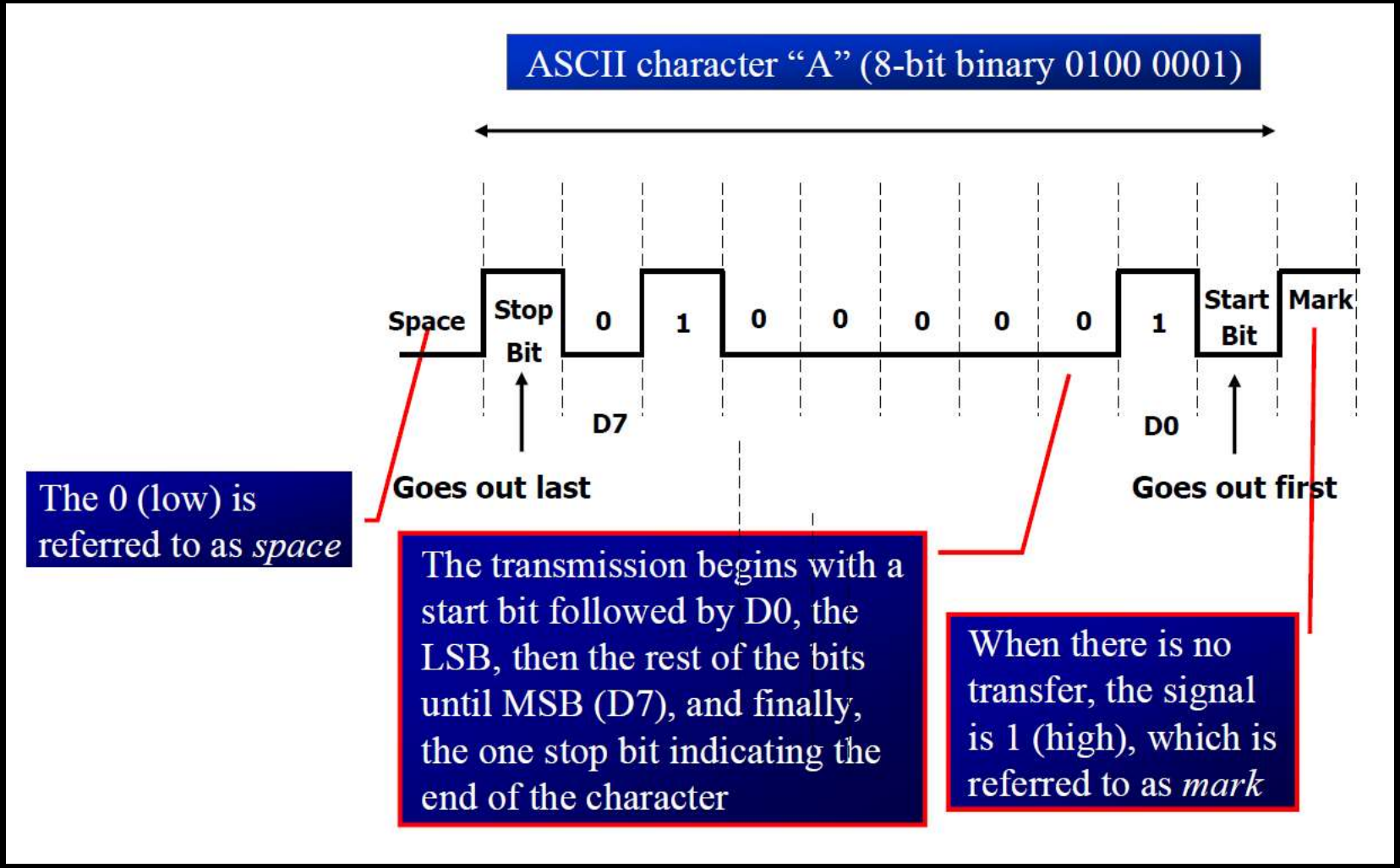
Asynchronous – Start & Stop Bit

- Asynchronous serial data communication is widely used for **character-oriented** transmissions
  - Each character is placed in between **start and stop bits**, this is called **framing.**
  - **Block-oriented** data transfers use the synchronous method.

- **The start bit is always one bit, but the stop bit can be one or two bits**

- **The start bit is always a 0 (low) and the stop bit(s) is 1 (high)**

Asynchronous – Start & Stop Bit



Data Transfer Rate

- The rate of data transfer in serial data communication is stated in **bps (bits per second)**.

- Another widely used terminology for bps is **baud rate.**
  - It is modem terminology and is defined as **the number of signal changes per second**
  - In modems, there are occasions when a single change of signal transfers several bits of data

- As far as the **conductor wire** is concerned, **the baud rate and bps are the same.**

8051 Serial Port

- Synchronous and Asynchronous
- SCON Register is used to Control
- Data Transfer through TXd & RXd pins
- Some time - Clock through TXd Pin
- Four Modes of Operation:

Mode 0	**:Synchronous Serial Communication**
Mode 1	**:8-Bit UART with Timer Data Rate**
Mode 2	**:9-Bit UART with Set Data Rate**
Mode 3	**:9-Bit UART with Timer Data Rate**

Registers related to Serial Communication

1. SBUF Register
2. SCON Register
3. PCON Register

SBUF Register

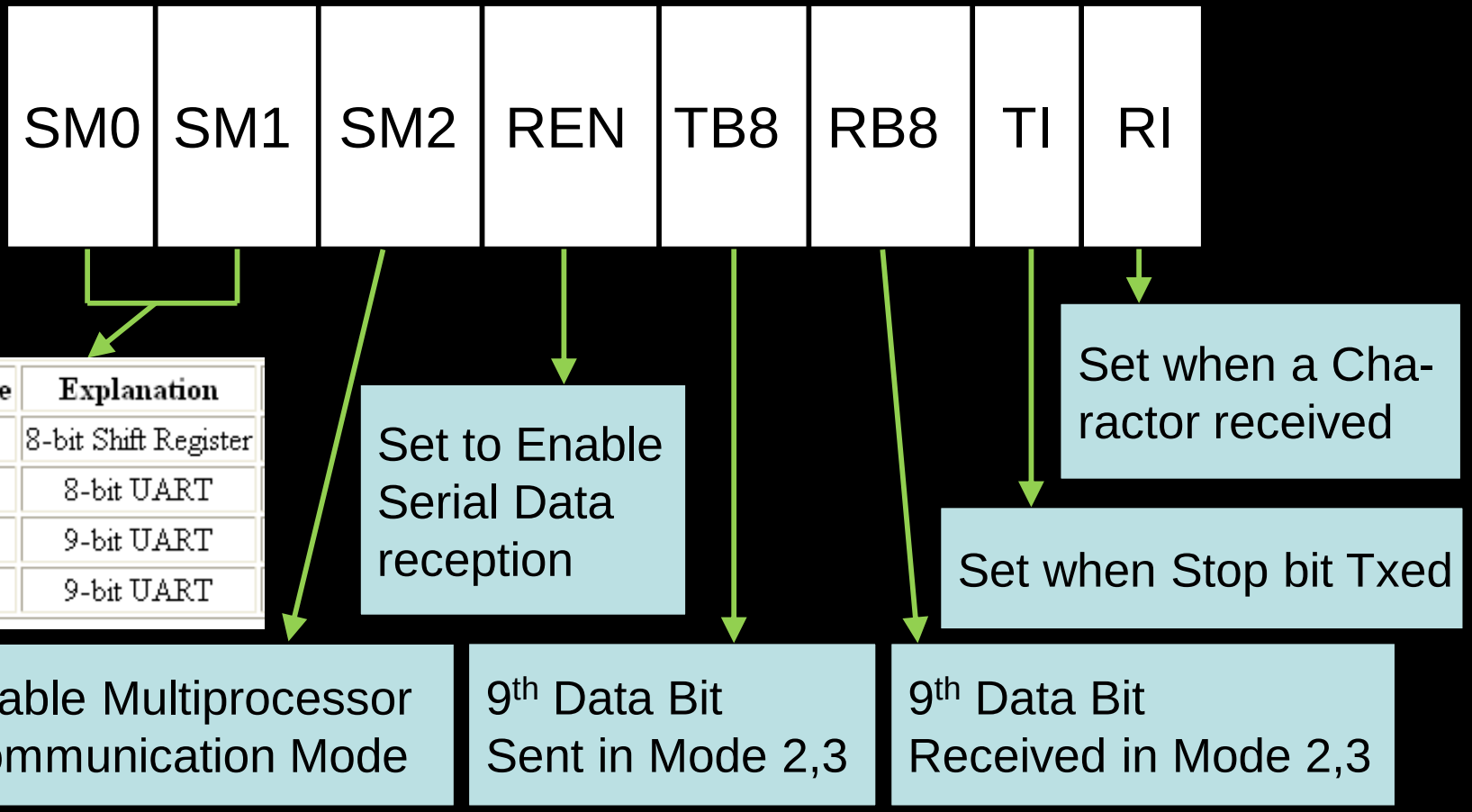
- **SBUF** is an **8-bit register** used solely for serial communication.
- For a byte data to be transferred via the **TxD line**, it must be placed in the **SBUF register**.
- The moment a byte is written into SBUF, it is framed with the start and stop bits and transferred serially via the TxD line.
- SBUF holds the byte of data when it is received by 8051 **RxD** line.
- When the bits are received serially via RxD, the **8051 deframes** it by eliminating the stop and start bits, making a byte out of the data received, and then placing it in SBUF.

SBUF Register

- **Sample Program:**

```
MOV SBUF,#'D'   ;load SBUF=44h, ASCII for 'D'
MOV SBUF,A      ;copy accumulator into SBUF
MOV A,SBUF      ;copy SBUF into accumulator
```

SCON Register



Serial Mode	Explanation
0	8-bit Shift Register
1	8-bit UART
2	9-bit UART
3	9-bit UART

8051 Serial Port – Mode 0

The Serial Port in Mode-0 has the following features:

1. Serial data **enters and exits through RXD**

2. **TXD** outputs the **clock**

3. 8 bits are transmitted / received

4. The baud rate is fixed at (1/12) of the oscillator frequency

8051 Serial Port – Mode 1

The Serial Port in Mode-1 has the following features:

1. Serial data **enters through RXD**
2. Serial data **exits through TXD**
3. On receive, the stop bit goes into RB8 in SCON
4. **10 bits** are transmitted / received
    1. *Start bit (0)*
    2. *Data bits (8)*
    3. *Stop Bit (1)*
5. Baud rate is determined by the Timer 1 over flow rate.

8051 Serial Port – Mode 2

The Serial Port in Mode-2 has the following features:

1. Serial data **enters through RXD**
2. Serial data **exits through TXD**
3. 9th data bit (**TB8**) can be assign value 0 or 1
4. On receive, the 9th data bit goes into **RB8** in SCON
5. **11 bits** are transmitted / received
    1. Start bit (0)
    2. Data bits (9)
    3. Stop Bit (1)
6. **Baud rate** is programmable

8051 Serial Port – Mode 3

The Serial Port in Mode-3 has the following features:

1. Serial data **enters through RXD**
2. Serial data **exits through TXD**
3. 9th data bit (**TB8**) can be assign value 0 or 1
4. On receive, the 9th data bit goes into **RB8** in SCON
5. **11 bits** are transmitted / received
    1. Start bit (0)
    2. Data bits (9)
    3. Stop Bit (1)
6. **Baud rate** is determined by Timer 1 overflow rate.

Programming Serial Data Transmission

1. **TMOD register** is loaded with the value **20H**, indicating the use of timer 1 in mode 2 (8-bit auto-reload) **to set baud rate**.
2. The **TH1** is loaded with one of the values to set baud rate for serial data transfer.
3. The **SCON register** is loaded with the value **50H**, indicating serial mode 1, where an 8- bit data is framed with start and stop bits.
4. **TR1** is set to 1 to start timer 1
5. **TI** is cleared by **CLR TI** instruction
6. The character byte to be transferred serially is written into **SBUF register.**
7. The **TI flag bit** is monitored with the use of instruction **JNB TI, xx** to see if the character has been transferred completely.
8. To transfer the next byte, **go to step 5**

Programming Serial Data Reception

1. **TMOD register** is loaded with the value **20H**, indicating the use of timer 1 in mode 2 (8-bit auto-reload) **to set baud rate.**
2. **TH1** is loaded to set baud rate
3. The **SCON register** is loaded with the value **50H**, indicating serial mode 1, where an 8- bit data is framed with start and stop bits.
4. **TR1** is set to 1 to start timer 1
5. **RI** is cleared by **CLR RI** instruction
6. The **RI flag bit** is monitored with the use of instruction **JNB RI, xx** to see if an entire character has been received yet
7. **When RI is raised**, **SBUF** has the byte, its contents are moved into a safe place.
8. To receive the next character, **go to step 5.**

Doubling Baud Rate

- There are two ways to increase the baud rate of data transfer
  1. **By using a higher frequency crystal**
  2. **By changing a bit in the PCON register**

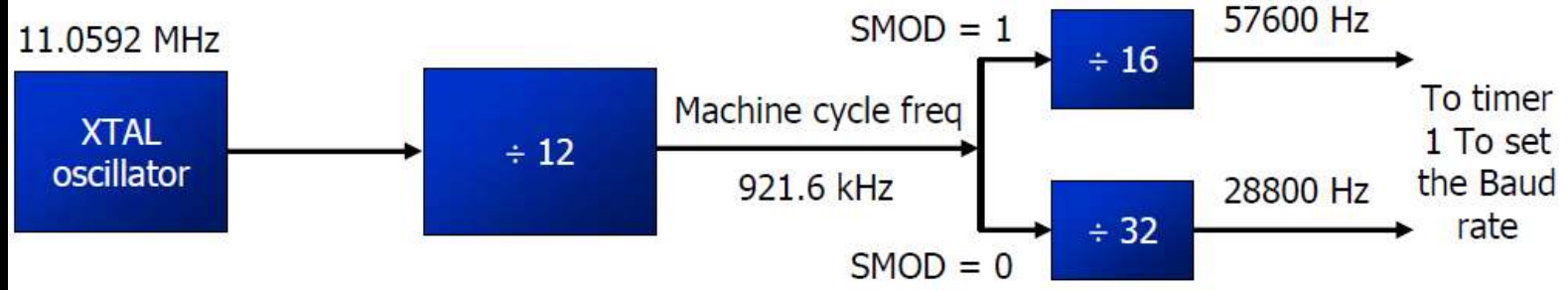
- **PCON register** is an 8-bit register.

SMOD	--	--	--	GF1	GF0	PD	IDL

- When 8051 is powered up, **SMOD** is zero
- **We can set it to high** by software and thereby **double** the baud rate.

Doubling Baud Rate (cont…)

```
MOV   A,PCON      ;place a copy of PCON in ACC
SETB  ACC.7       ;make D7=1
MOV   PCON,A      ;changing any other bits
```



Baud Rate comparison for SMOD=0 and SMOD=1

TH1	(Decimal)	(Hex)	SMOD=0	SMOD=1
	-3	FD	9600	19200
	-6	FA	4800	9600
	-12	F4	2400	4800
	-24	E8	1200	2400

8051 Interrupts

INTERRUPTS

- An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service

- A single microcontroller can serve several devices by two ways:

1. **Interrupt**
2. **Polling**

Interrupt Vs Polling

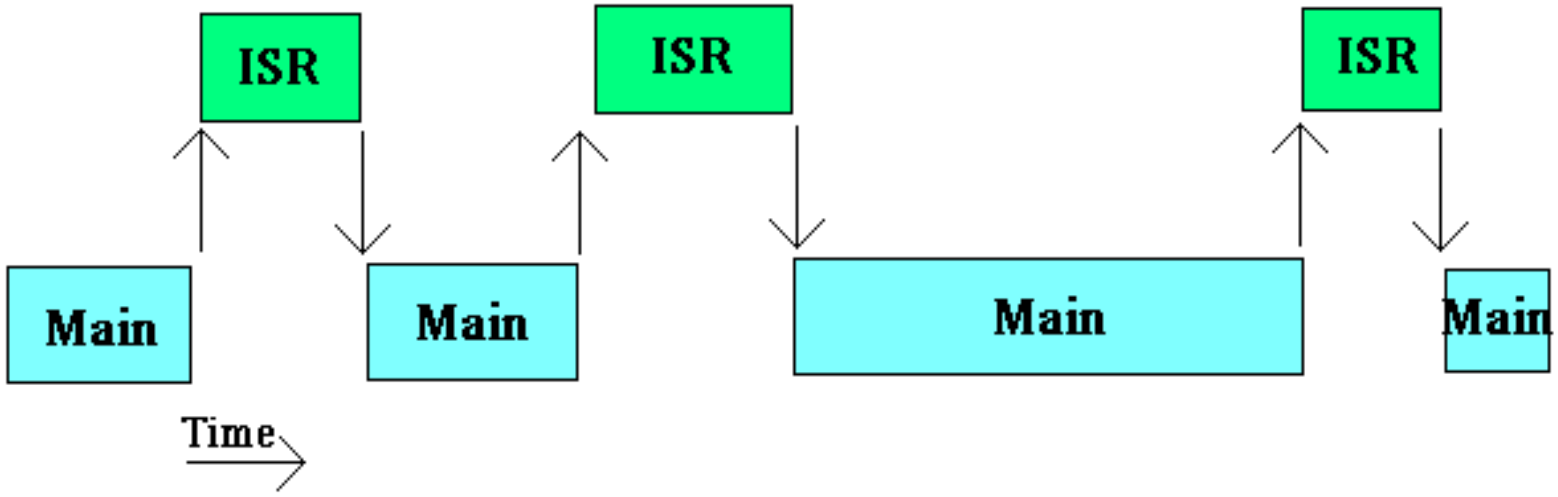
1. **Interrupts**
   - Whenever any device needs its service, the device notifies the microcontroller by sending it an **interrupt signal**.
   - Upon receiving an interrupt signal, the **microcontroller interrupts** whatever it is doing and serves the device.
   - The program which is associated with the interrupt is called the **interrupt service routine (ISR)** or interrupt handler.
2. **Polling**
   - The microcontroller **continuously monitors** the status of a given device.
   - When the **conditions** met, it performs the service.
   - After that, it moves on to monitor the **next device** until every one is serviced.

Interrupt Vs Polling

- The **polling method is not efficient**, since it wastes much of the microcontroller's time by polling devices that do not need service.
- The **advantage of interrupts** is that the microcontroller can serve many devices (not all at the same time).
- Each devices can get the attention of the microcontroller based on the **assigned priority**.
- For the polling method, it is **not possible** to assign priority since it checks all devices in a round-robin fashion.
- The microcontroller can also **ignore (mask)** a device request for service in Interrupt.

Program execution without intrrupts :

Program execution with intrrupts :



ISR : Intrrupt Service Routin

Six Interrupts in 8051

Six interrupts are allocated as follows:

1. **Reset – power-up reset.**

2. **Two interrupts are set aside for the timers.**
   - one for timer 0 and one for timer 1

3. **Two interrupts are set aside for hardware external interrupts.**
   - P3.2 and P3.3 are for the external hardware interrupts INT0 (or EX1), and INT1 (or EX2)

4. **Serial communication has a single interrupt that belongs to both receive and transfer.**

What events can trigger Interrupts?

- We can configure the 8051 so that any of the following events will cause an interrupt:
  - Timer 0 Overflow.
  - Timer 1 Overflow.
  - Reception/Transmission of Serial Character.
  - External Event 0.
  - External Event 1.
- We can configure the 8051 so that when Timer 0 Overflows or when a character is sent/received, the appropriate interrupt handler routines are called.

8051 Interrupt Vectors

INTERRUPT VECTORS

When the original 8051 and 8031 were introduced, only 5 interrupts were provided.

Interrupt Number	Interrupt Vector Address	Description
0	0003h	EXTERNAL 0
1	000Bh	TIMER/COUNTER 0
2	0013h	EXTERNAL 1
3	001Bh	TIMER/COUNTER 1
4	0023h	SERIAL PORT

8051 Interrupt related Registers

- The various registers associated with the use of interrupts are:
  - TCON - Edge and Type bits for External Interrupts 0/1
  - SCON - RI and TI interrupt flags for RS232
  - IE - Enable interrupt sources
  - IP - Specify priority of interrupts

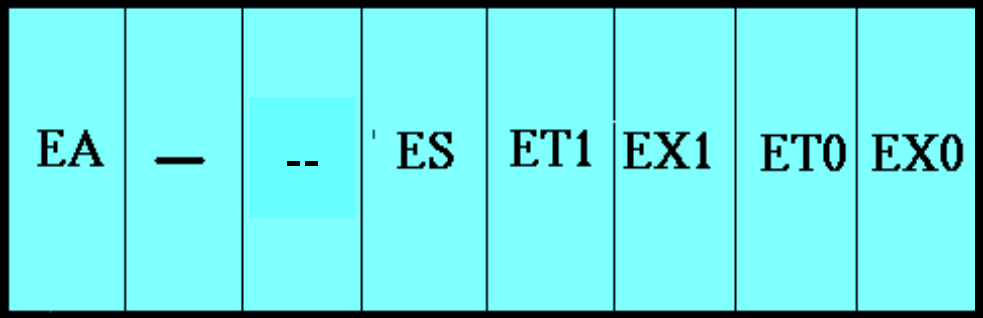
Enabling and Disabling an Interrupt

- Upon **reset**, all interrupts are **disabled (masked),** meaning that none will be responded to by the microcontroller if they are activated.

- The interrupts must be **enabled** by software in order for the microcontroller to respond to them.

- There is a register called **IE (interrupt enable)** that is responsible for enabling (unmasking) and disabling (masking) the interrupts.

Interrupt Enable (IE) Register



- EA : Global enable/disable.
- --- : Reserved for additional interrupt hardware.
- ES : Enable Serial port interrupt.
- ET1 : Enable Timer 1 control bit.
- EX1 : Enable External 1 interrupt.
- ET0 : Enable Timer 0 control bit.
- EX0 : Enable External 0 interrupt.

MOV IE,#08h
or
SETB ET1

Enabling and Disabling an Interrupt

- **Example:** Show the instructions to **(a)** enable the serial interrupt, timer 0 interrupt, and external hardware interrupt 1 and **(b)** disable (mask) the timer 0 interrupt, then **(c)** show how to disable all the interrupts with a single instruction.
- **Solution:**
  - **(a) MOV IE,#10010110B** ;enable serial, timer 0, EX1
    - Another way to perform the same manipulation is:
      - **SETB IE.7** ;EA=1, global enable
      - **SETB IE.4** ;enable serial interrupt
      - **SETB IE.1** ;enable Timer 0 interrupt
      - **SETB IE.2** ;enable EX1
  - **(b) CLR IE.1** ;mask (disable) timer 0 interrupt only
  - **(c) CLR IE.7** ;disable all interrupts

Interrupt Priority

- When the 8051 is powered up, the priorities are assigned according to the following.

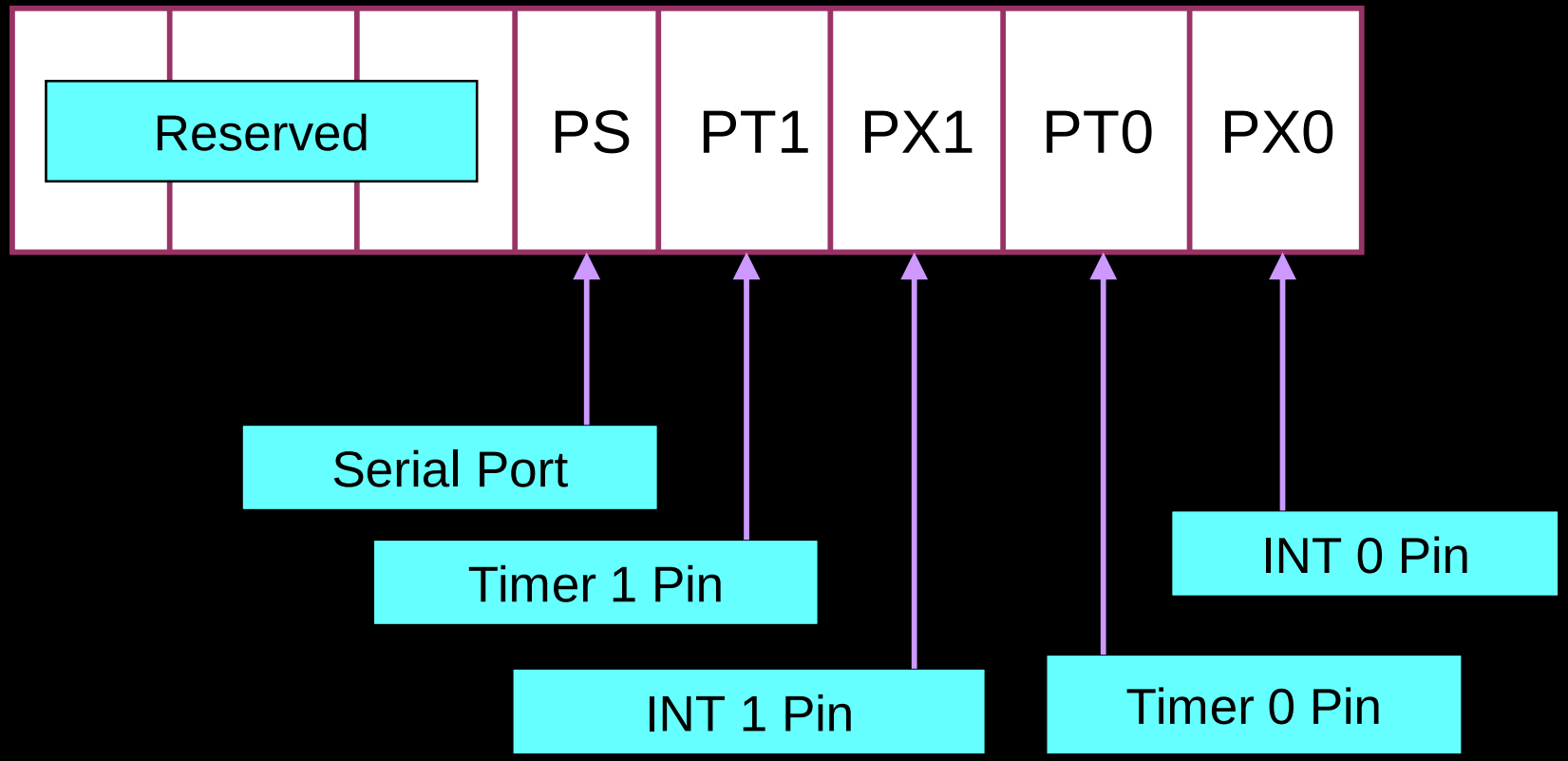
- In reality, the priority scheme is nothing but an internal polling sequence in which the 8051 polls the interrupts in the sequence listed and responds accordingly.

Highest To Lowest Priority	
External Interrupt 0	(INT0)
Timer Interrupt 0	(TF0)
External Interrupt 1	(INT1)
Timer Interrupt 1	(TF1)
Serial Communication	(RI + TI)

Interrupt Priority

- **We can alter** the sequence of interrupt priority by assigning a higher priority to any one of the interrupts by programming a register called **IP (interrupt priority).**
- To give a higher priority to any of the interrupts, we make the **corresponding bit in the IP register high.**

Interrupt Priority (IP) Register



Priority bit=1 assigns high priority
Priority bit=0 assigns low priority