# Microcontroller

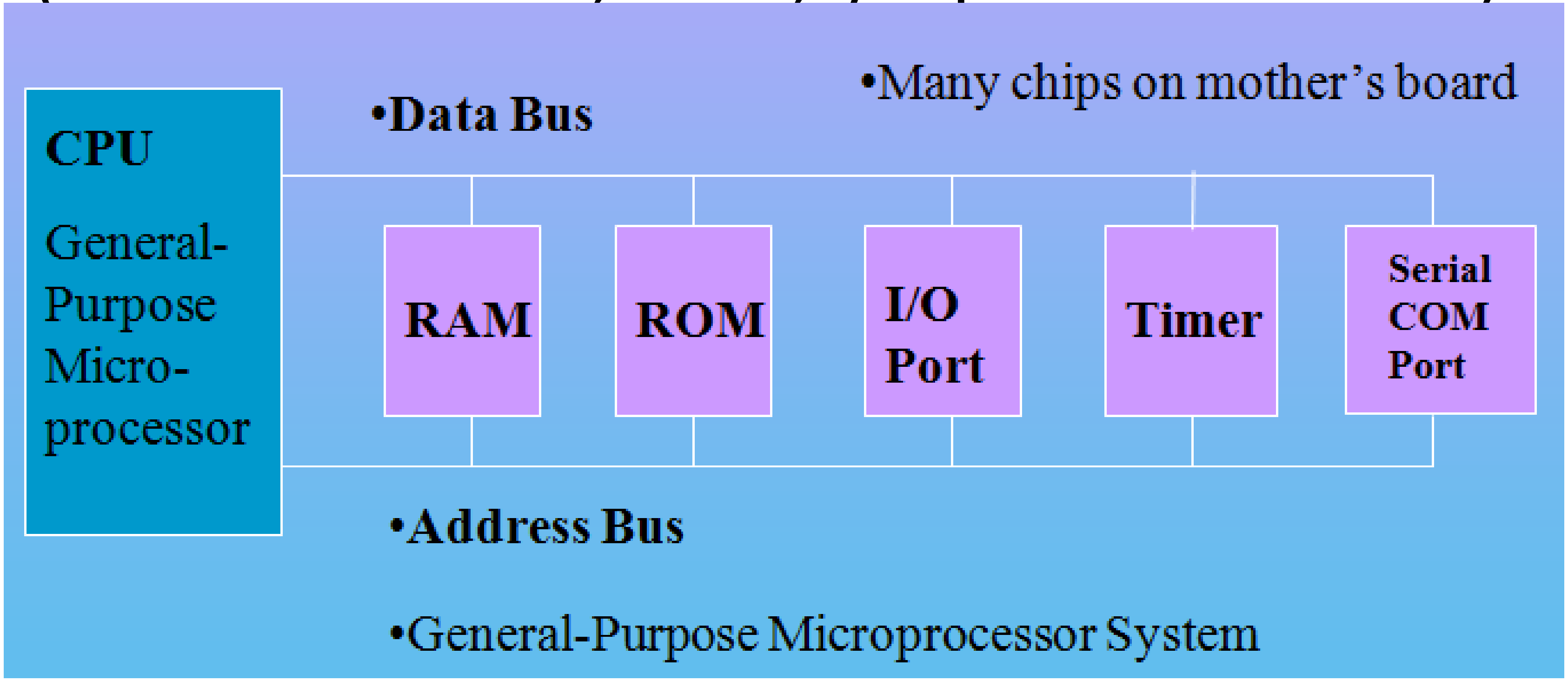
8051

# Microprocessor Based System

CPU

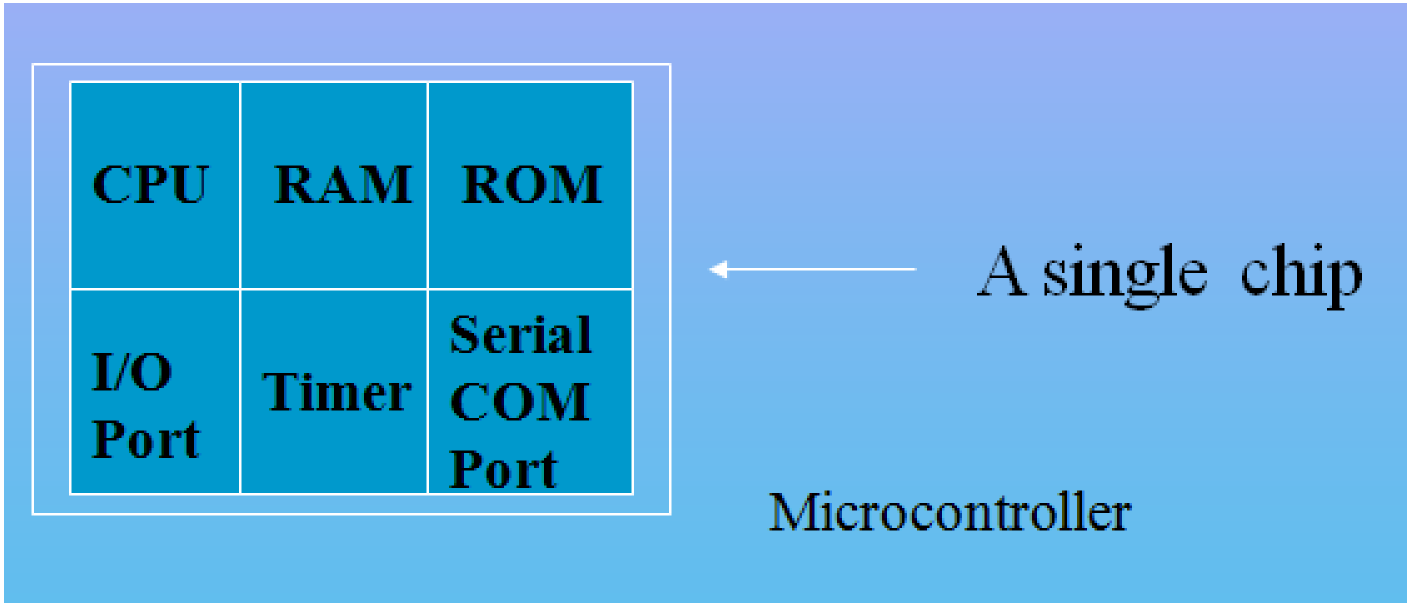
External RAM, ROM, I/O

(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.
- Expensive
- Versatility
- General-purpose

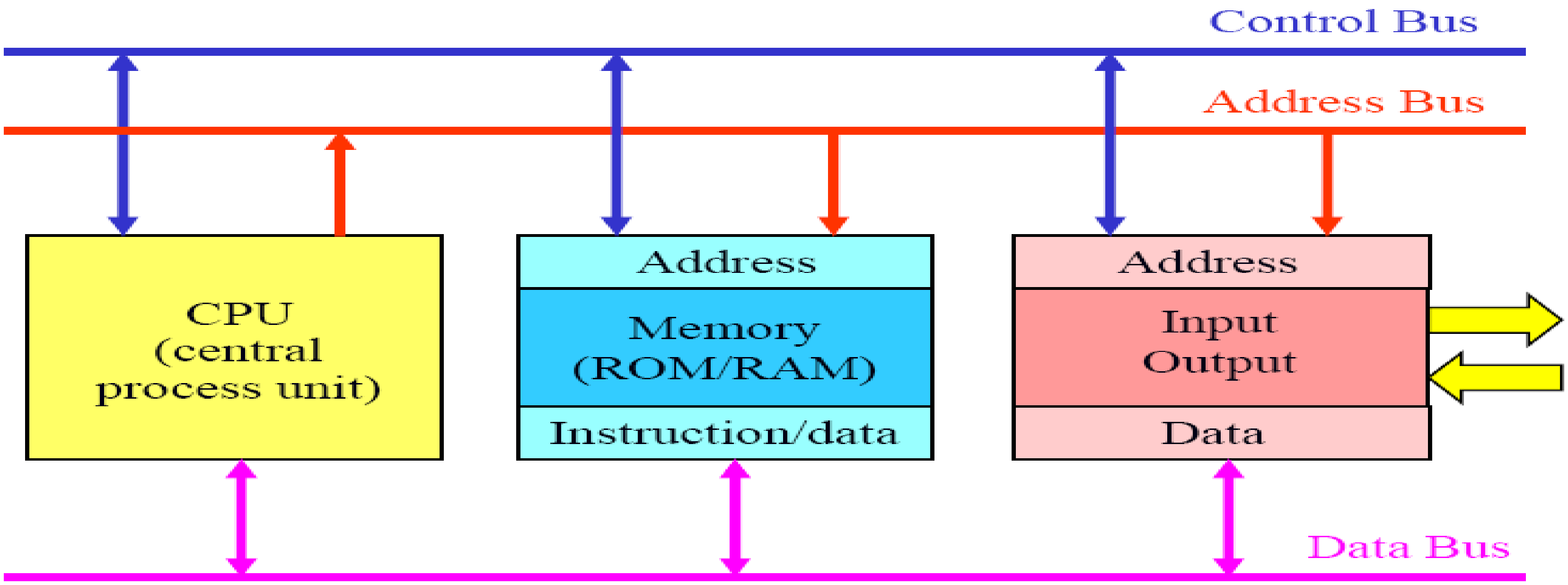
## Microcontroller

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

# Von Newmann Architecture

Von Neumann Architecture—another type of computer architecture where the instructions and data are stored in the same memory space

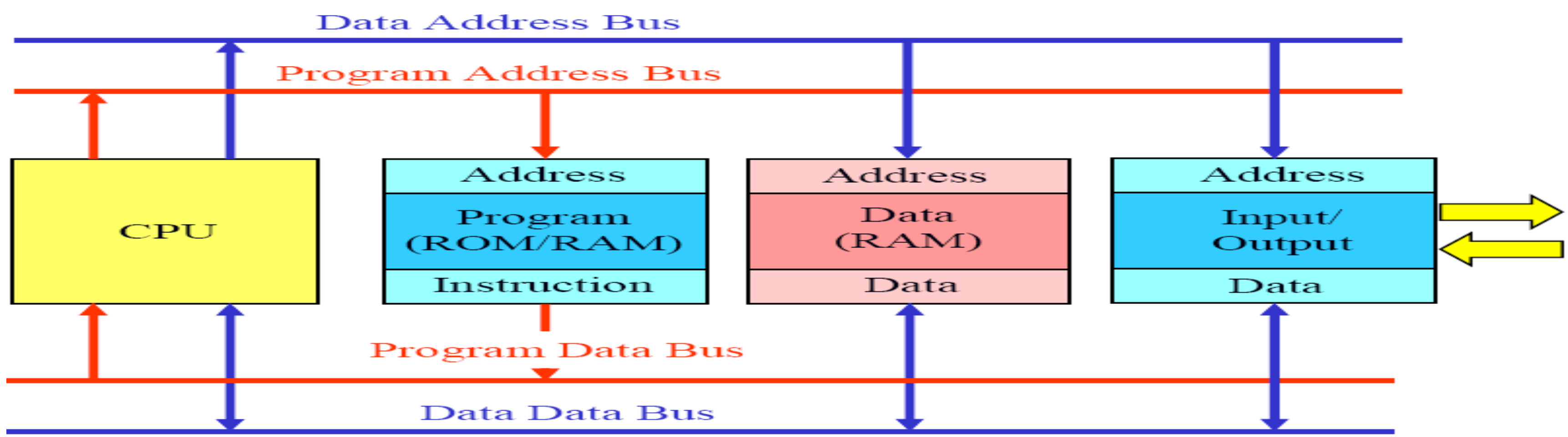
Example: Intel x86 architecture (Intel Pentium, AMD Athlon, etc.)



# Harvard Architecture

Harvard Architecture—a type of computer architecture where the instructions (program code) and data are stored in separate memory spaces

Example: Intel 8051 architecture




Often separate buses are used to access memory and inputs/outputs

# 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

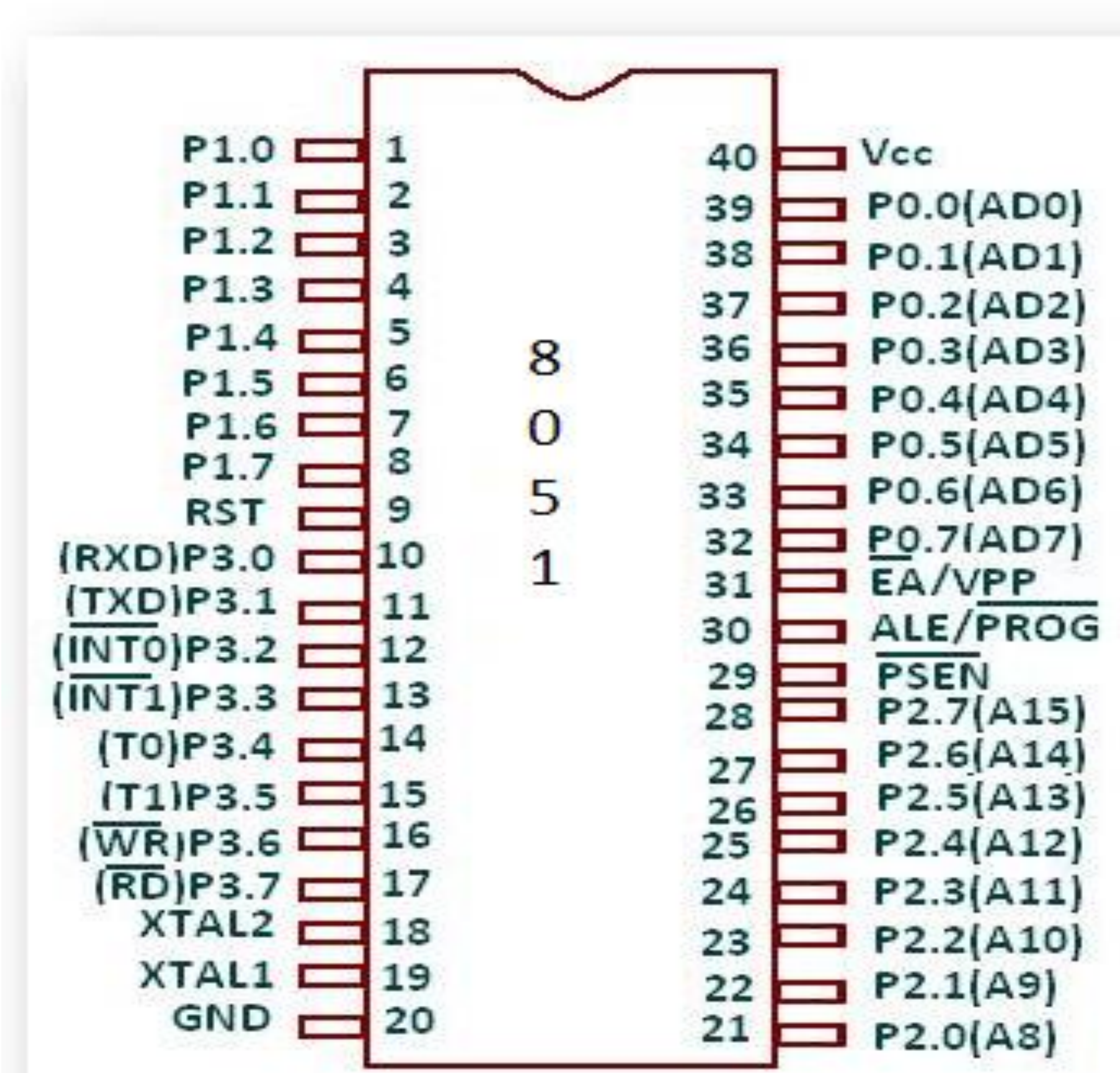
32 General Purpose Registers each of 8 bits

# 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

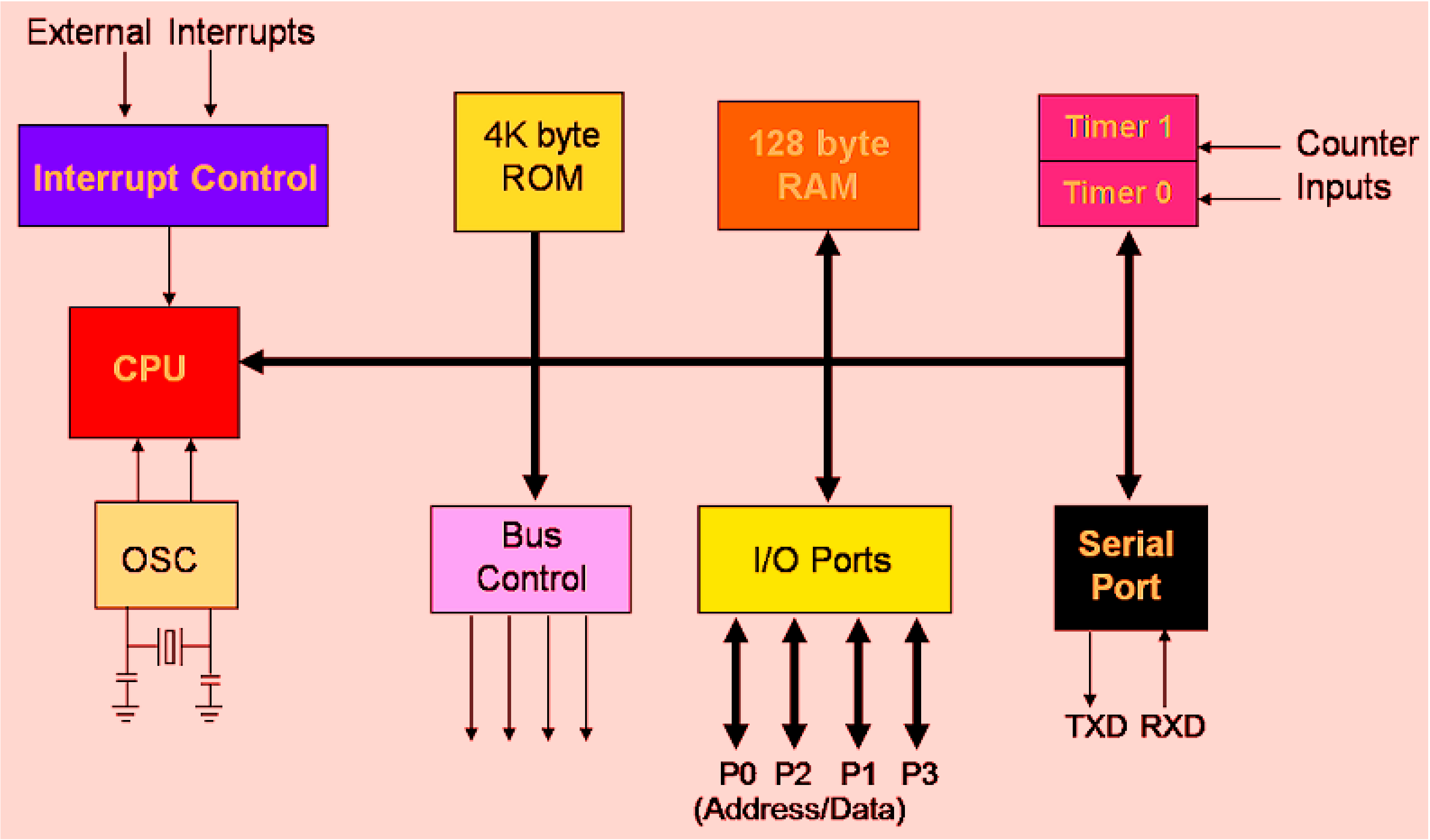
| Feature | 8051 | 8052 | 8031 |
|---|---|---|---|
| ROM (on-chip program space in bytes) | 4K | 8K | 0K |
| RAM (bytes) | 128 | 256 | 128 |
| Timers | 2 | 3 | 2 |
| I/O pins | 32 | 32 | 32 |
| Serial port | 1 | 1 | 1 |
| Interrupt sources | 6 | 8 | 6 |

# Pin Diagram

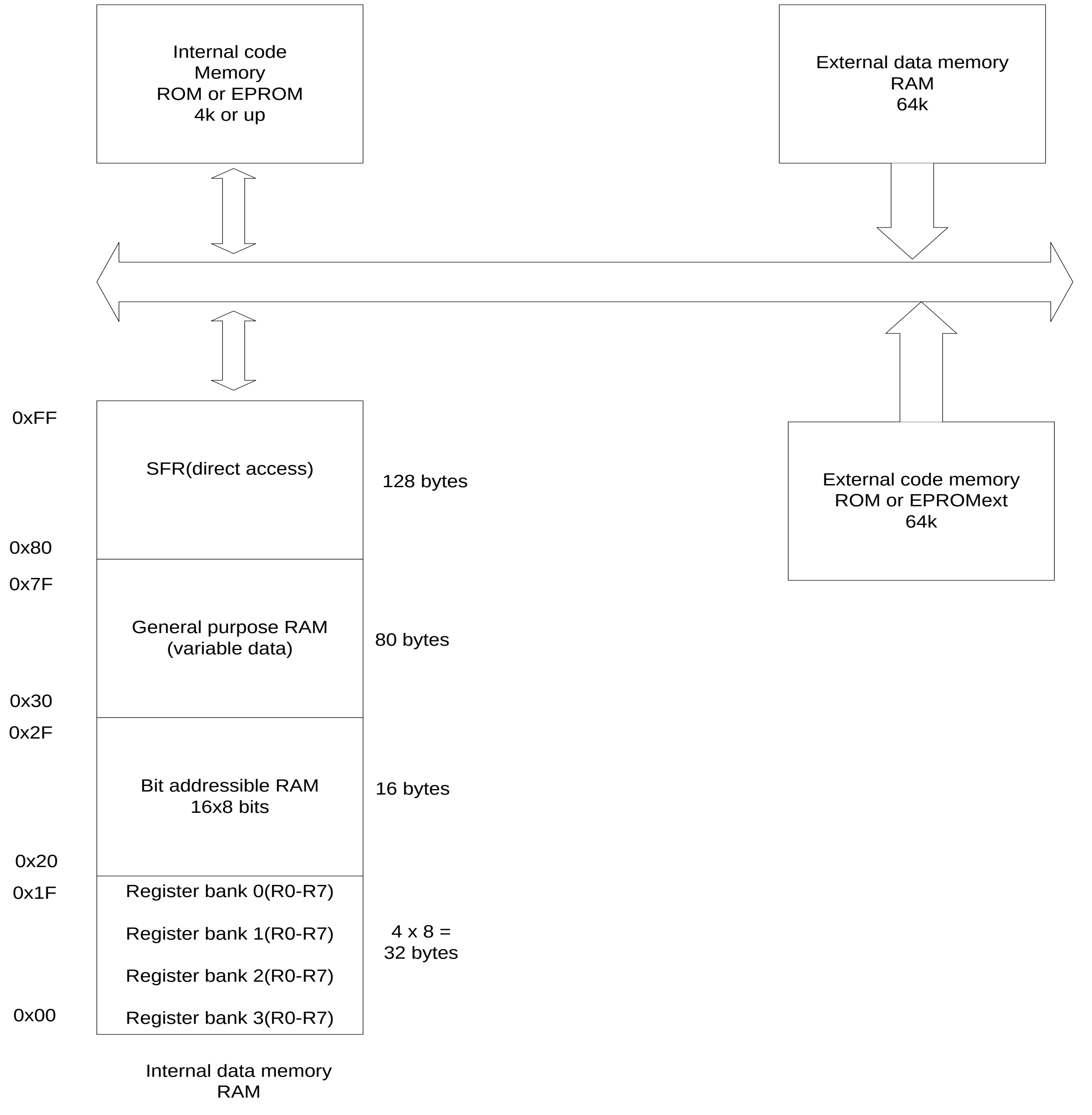


P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
(RXD)P3.0 10
(TXD)P3.1 11
(INT0)P3.2 12
(INT1)P3.3 13
(T0)P3.4 14
(T1)P3.5 15
(WR)P3.6 16
(RD)P3.7 17
XTAL2 18
XTAL1 19
GND 20
8051
40 Vcc
39 P0.0(AD0)
38 P0.1(AD1)
37 P0.2(AD2)
36 P0.3(AD3)
35 P0.4(AD4)
34 P0.5(AD5)
33 P0.6(AD6)
32 P0.7(AD7)
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



# Separate read instructions for external data and code memory.

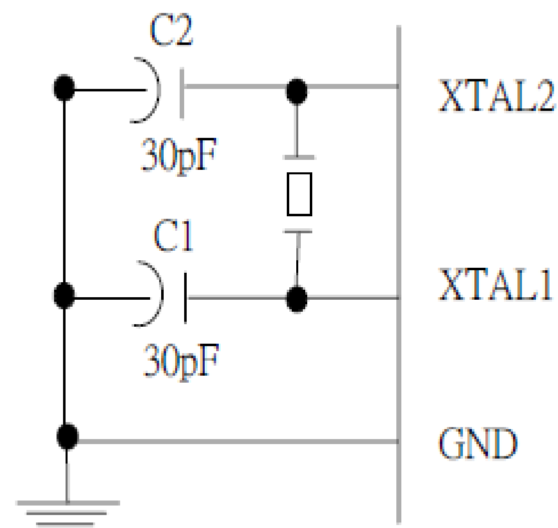


# Pin Description of the 8051

- 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

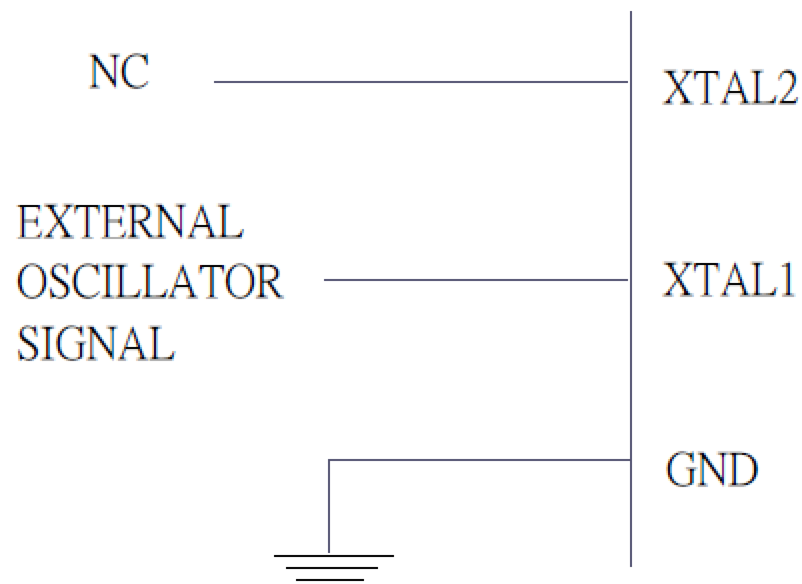
# XTAL1 and XTAL2

- 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.

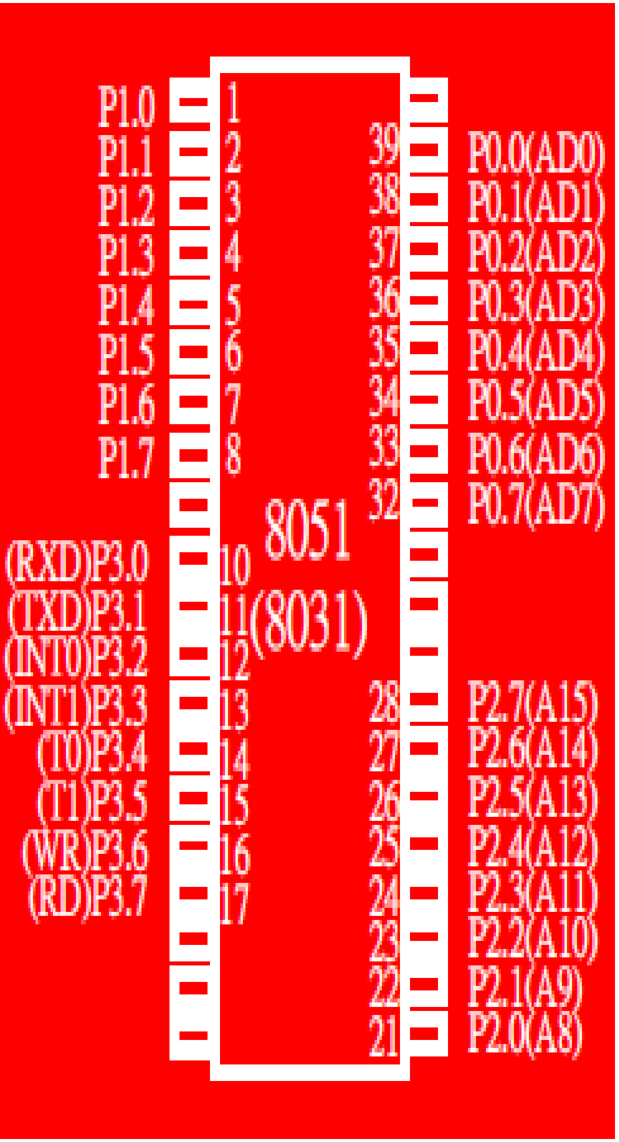
- Normally EA pin is connected to Vcc (Internal Access)

- EA pin must be connected to GND to indicate that the code or data is stored externally.

# PSEN’ and ALE

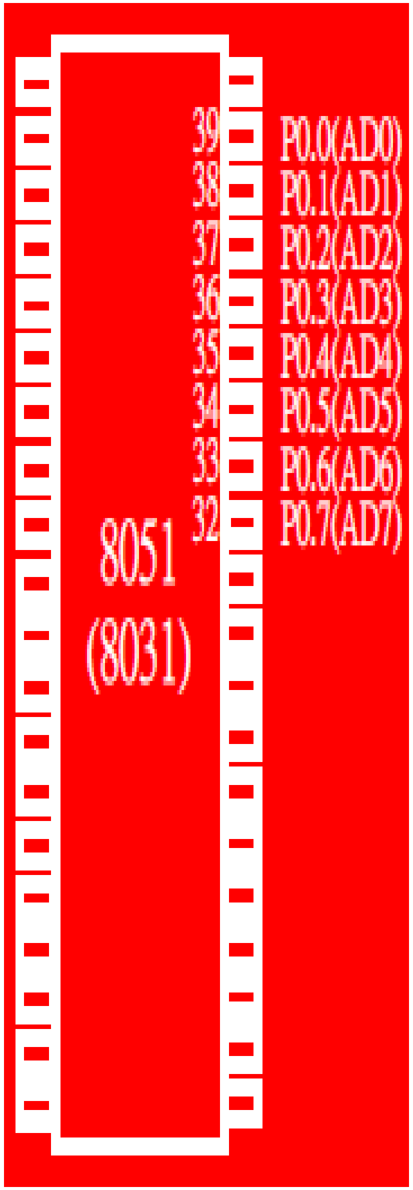
- PSEN, **“program store enable’’**, is an output pin
- This pin is connected to the OE pin of the external memory.
- For External Code Memory, PSEN’ = 0
- For External Data Memory, PSEN’ = 1
- ALE pin is used for demultiplexing the address and data.

# I/O Port Pins



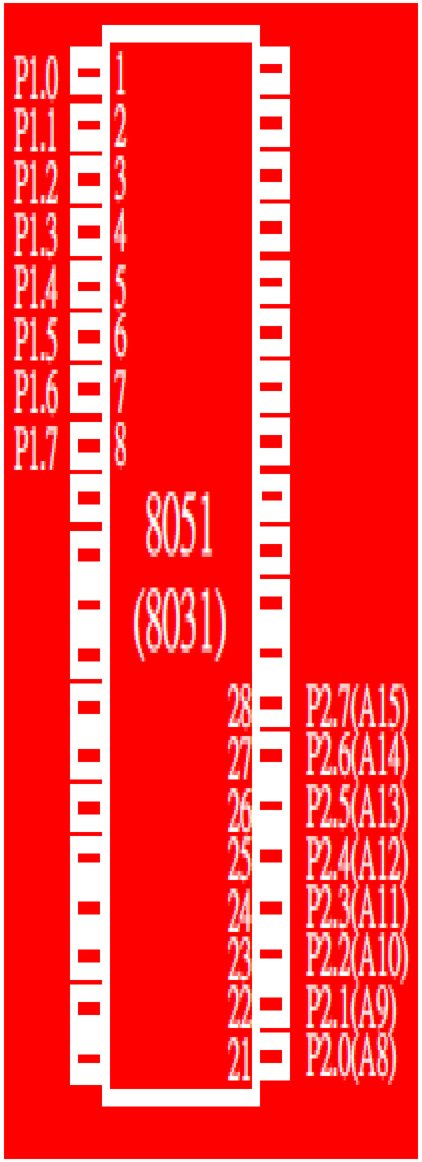

The four 8-bit I/O ports **P0, P1, P2 and P3** each uses 8 pins.

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 |

Serial communications

External interrupts

Timers

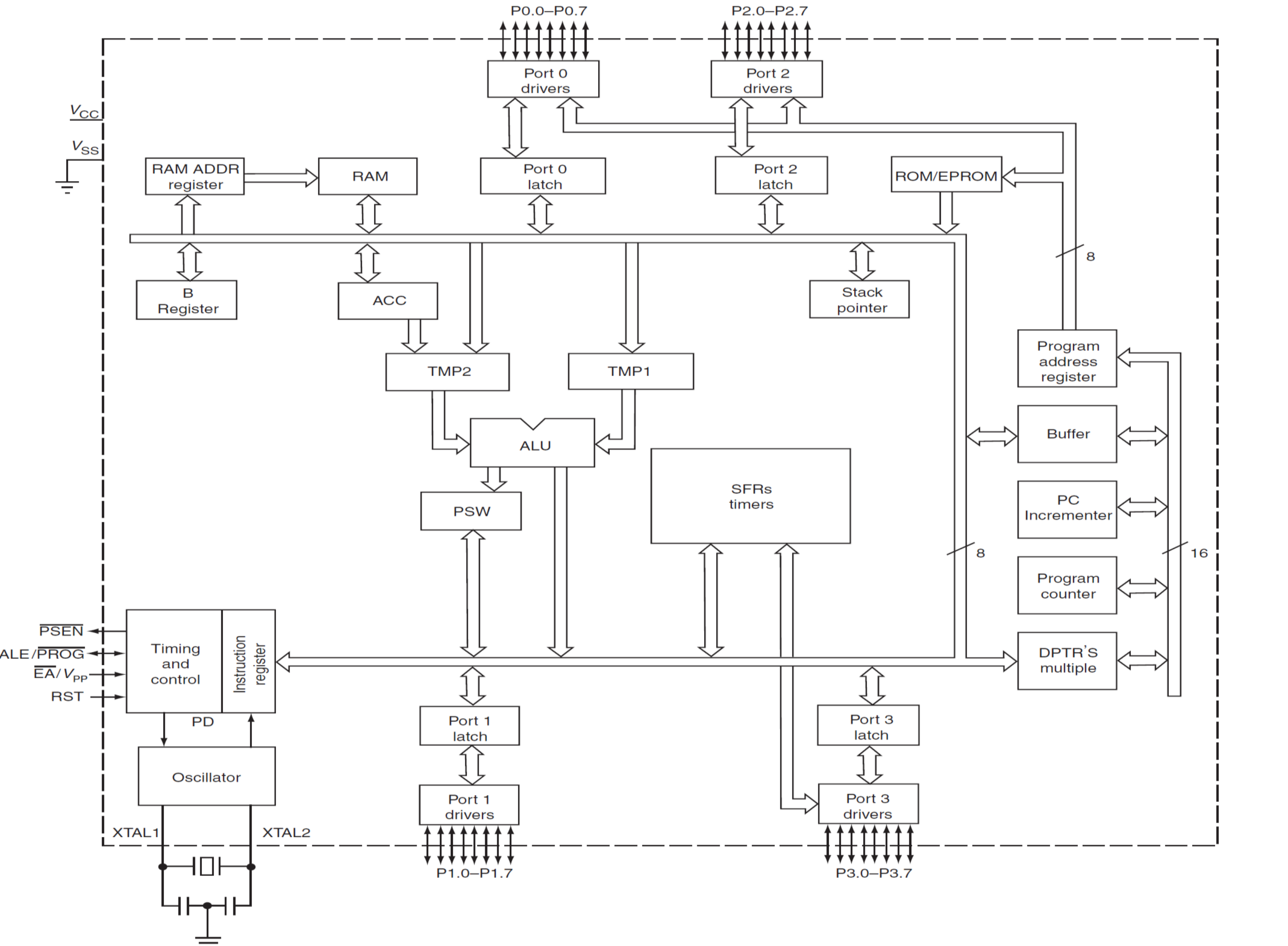
Read/Write signals of external memories

# 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. |

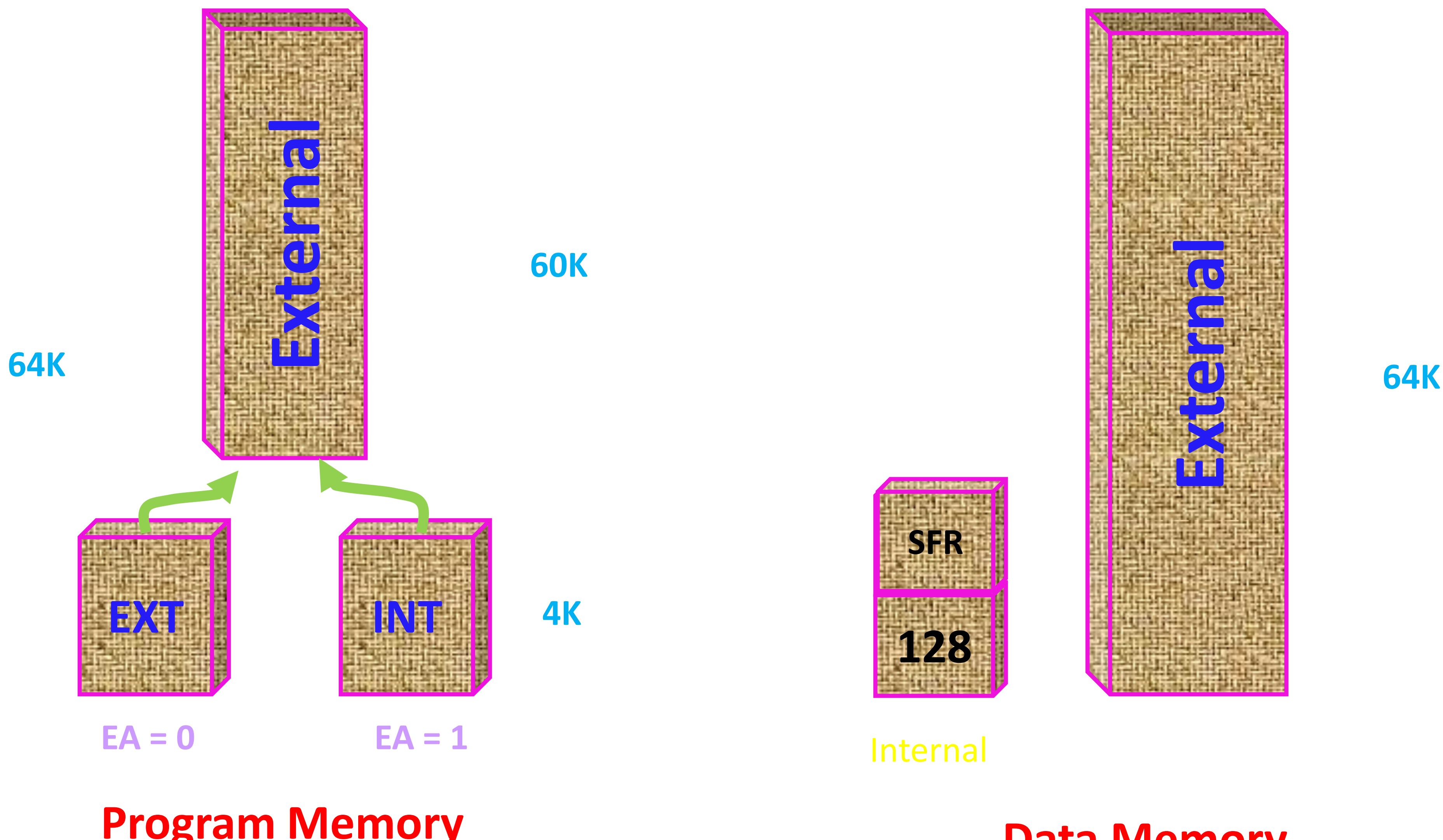
# Pin Description Summary

| 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) |

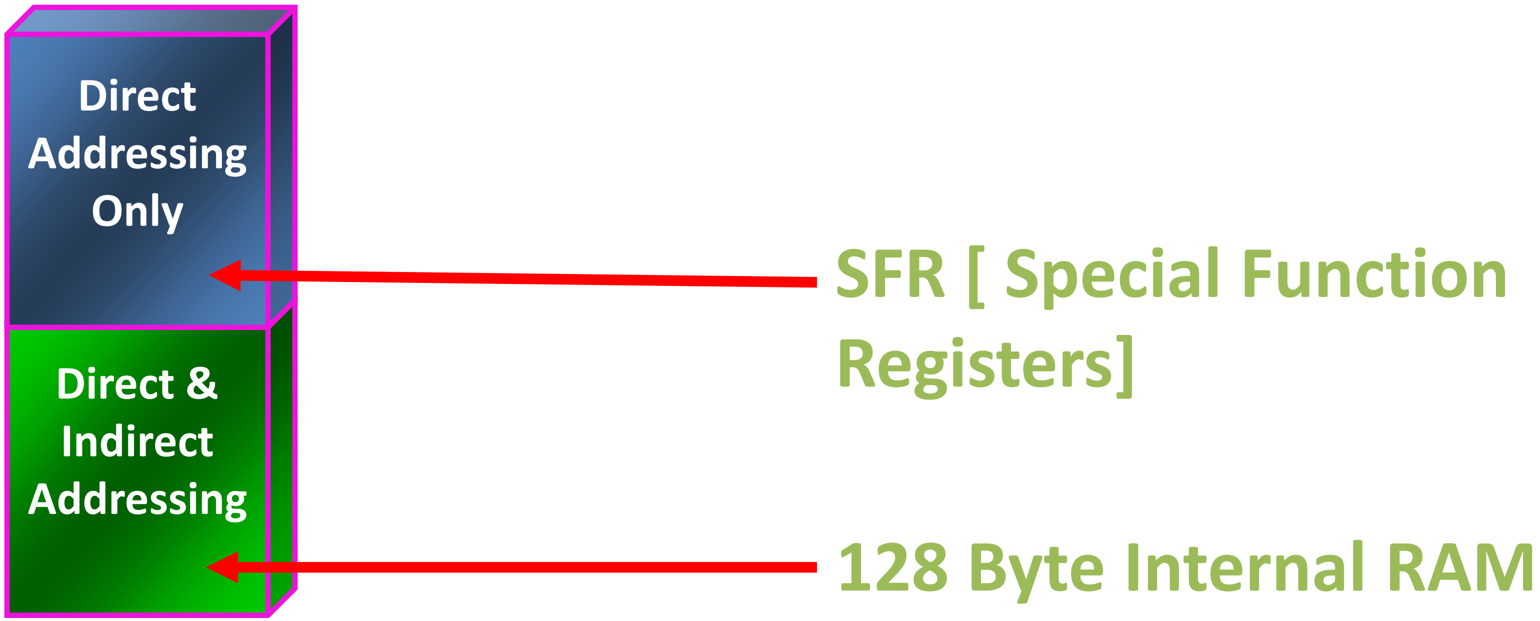

P0.0–P0.7
P2.0–P2.7
Port 0 drivers
Port 2 drivers
$V_{CC}$
$V_{SS}$
RAM ADDR register
RAM
Port 0 latch
Port 2 latch
ROM/EPROM
8
B Register
ACC
Stack pointer
TMP2
TMP1
Program address register
ALU
Buffer
SFRs timers
PSW
PC Incrementer
8
16
Program counter
PSEN
ALE/PROG
EA/$V_{PP}$
RST
Timing and control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

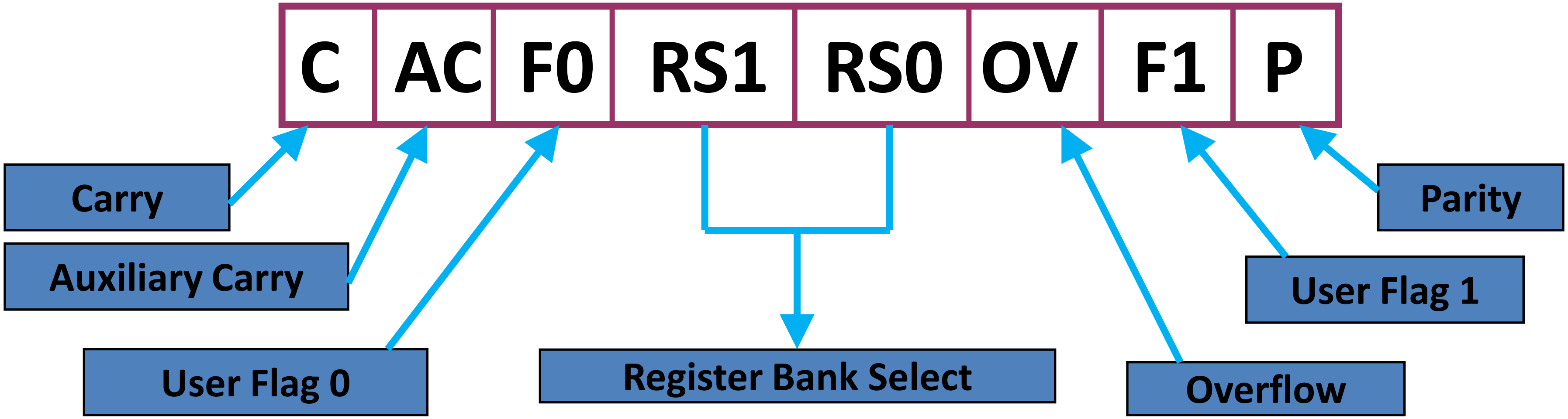


# Special function register

| | |
|---|---|
| 80 | PO |
| 81 | SP |
| 82 | DPL |
| 83 | DPH |
| 87 | PCON |
| 88 | TCON |
| 89 | TMOD |
| 8A | TL0 |
| 8B | TL1 |
| 8C | TH0 |
| 8D | TH1 |

| | |
|---|---|
| 90 | P1 |
| 98 | SCON |
| 99 | SBUF |
| A0 | P2 |
| A8 | IE |
| B0 | P3 |
| B8 | IP |
| D0 | PSW |
| E0 | ACC |
| F0 | B |

# Program Status Word [PSW]



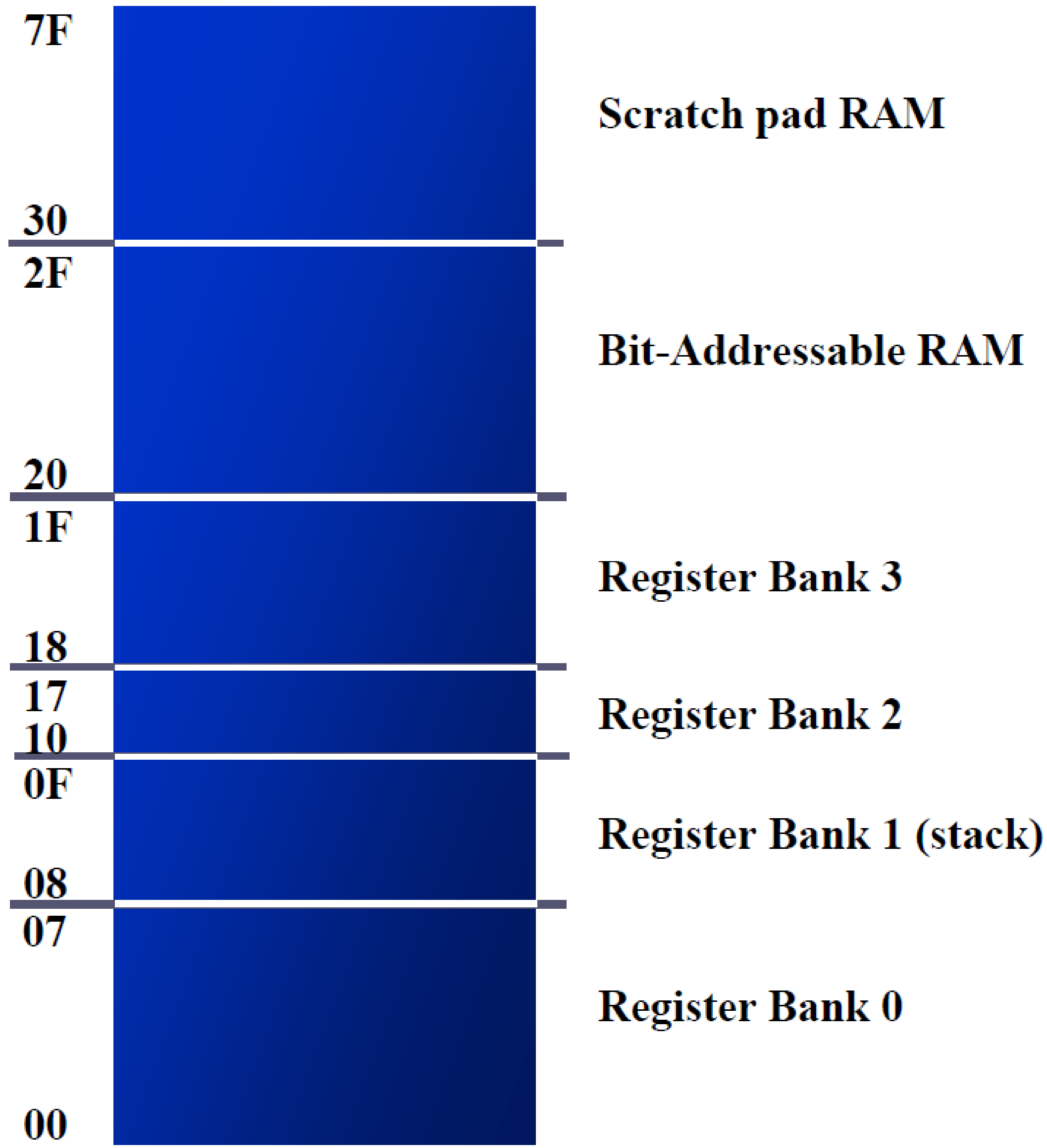
# 8051 instructions that affects flag

| 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

| | Register bank 0 | | Register bank 1 | | Register bank 2 | | Register bank 3 |
|---|---|---|---|---|---|---|---|
| 00 | R0 | 08 | R0 | 10 | R0 | 18 | R0 |
| 01 | R1 | 09 | R1 | 11 | R1 | 19 | R1 |
| 02 | R2 | 0A | R2 | 12 | R2 | 1A | R2 |
| 03 | R3 | 0B | R3 | 13 | R3 | 1B | R3 |
| 04 | R4 | 0C | R4 | 14 | R4 | 1C | R4 |
| 05 | R5 | 0D | R5 | 15 | R5 | 1D | R5 |
| 06 | R6 | 0E | R6 | 16 | R6 | 1E | R6 |
| 07 | R7 | 0F | R7 | 17 | R7 | 1F | R7 |

# 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

Bit-addressable locations

Byte address

| Byte address | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 7F – 30 | General purpose RAM | | | | | | | |
| 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 |
| 1F – 18 | Bank 3 | | | | | | | |
| 17 – 10 | Bank 2 | | | | | | | |
| 0F – 08 | Bank 1 | | | | | | | |
| 07 – 00 | Default register bank for R0-R7 | | | | | | | |

# 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
```

# Example

Write code to send 55H to port P1 and P2 using their names and their addresses.

1. MOV A,#55H
   MOV P1,A
   MOV P2,A

2. MOV A,#55H
   MOV 80H,A
   MOV 0A0,A

# 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 Memory RAM

- External Memory is accessed.

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

# External Memory ROM

- External ROM can be accessed by MOVC instruction.
- This instruction moves a byte of data located in program ROM into register A.
- MOVC A,@A+DPTR
- MOVC A,@A+PC
- This allows us to put strings of data, such as lookup table elements.

# 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
DATA2:  DB    00110101B
DATA3:  DB    39H
        ORG   510H
DATA4:  DB    “2591”
        ORG   518H
DATA6:  DB    “My name is Joe”
```

# Multiplication of Unsigned Numbers

MUL AB ; A $\times$ 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,<br>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 |

# List of bit addressable registers

- Following are the bit addressable registers:
  - Acc
  - B
  - PSW
  - P0,P1,P2,P3
  - IP
  - IE
  - TCON
  - SCON

# Logical Instructions

- ANL indicates AND operation
  - ANL A,Rn
  - ANL A,direct
  - ANL A,@Ri
  - ANL Mask,A
  - ANL direct,#data
  - ANL C,bit
  - ANL C,/bit
- Similarly ORL/XRL instructions are meant for logical OR and XOR operation.

# Logical Instructions

- CLR A
- CPL A
- RL A
- RR A
- RLC A
- RRC A
- SWAP A =>D7-D4<-> D3-D0

# Checking an input bit

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

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 low to high pulse to port P1.5 to turn on buzzer.

**Soln:**

```
OVEN_HOT BIT P2.3
BUZZER       BIT P1.5
Here: JNB OVEN_HOT,Here
           CLR BUZZER
            SETB BUZZER
```

- A switch is connected to pin P1.7. Write a program to check the status of SW and perform the following:

  If SW=0, send letter ‘N’ to P2

  If SW=1, send letter ‘Y’ to P2

Soln:

```
        SETB P1.7
Again:  JB P1.2,over
        MOV P2,#'N'
        SJMP Again
Over:   MOV P2,#'Y'
        SJMP Again
```

# 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 and register 0 is selected.

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

**Solution:**

| | After PUSH 6 | After PUSH 1 | After PUSH 4 |
|---|---|---|---|
| 0B | 0B | 0B | 0B |
| 0A | 0A | 0A | 0A F3 |
| 09 | 09 | 09 12 | 09 12 |
| 08 | 08 25 | 08 25 | 08 25 |
| Start SP = 07 | SP = 08 | SP = 09 | SP = 0A |

# 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

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)

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
        MOV  R3,#10
NEXT:   MOV  R2,#70
AGAIN:  CPL  A
        DJNZ R2,AGAIN
        DJNZ R3,NEXT
```

```
;A=55H
;R3=10, the outer loop count
;R2=70, the inner loop count
;complement A register
;repeat it 70 times (inner loop)
```

# 8051 Conditional Jump Instructions

| Instruction | Action |
|---|---|
| JZ | Jump if $A = 0$ |
| JNZ | Jump if $A \neq 0$ |
| DJNZ | Decrement and jump if register $\neq 0$ |
| CJNE A, data | Jump if $A \neq$ data |
| CJNE reg, #data | Jump if byte $\neq$ #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

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:   ...
```

Example 2: Write a program to bring in data in serial form from P0.0 and send it out in parallel form to P1.

```
        MOV R0,#08
        SETB P0.0
Back:   MOV C,P0.0
        RRC A
        DJNZ R0, Back
        MOV P1,A
        END
```

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

```
        MOV A,#0      ;A=0
        MOV R5,A      ;clear R5
        ADD A,#79H    ;A=0+79H=79H ;
        JNC N_1       ;if CY=0, 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,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

- **Write an assembly language program to transfer N = 05 bytes of data from location 30h to location 40h.**

```
     mov  r0,#30h            //source address

     mov  r1,#40h            //destination address

     mov  r7,#05h            //Number of bytes to be moved
back: mov  a,@r0
     mov  @r1,a
     inc  r0
     inc  r1
     djnz  r7,back           //repeat till all data transferred end
```

- **Write an assembly language program to find the largest element in a given array of N =** 06h **bytes at location 4000h. Store the largest element at location 4062h.**

```
          mov   r3,#6                 //length of the array
          mov   dptr,#4000H           //starting address of array
          movx  a,@dptr
          mov   r1,a
nextbyte: inc   dptr
          movx  a,@dptr
          clr   c                     //reset borrow flag
          mov   r2,a                  //next number in the array
          subb  a,r1                  //other Num-Prev largest no.
          jc    skip                  // JNC FOR SMALLEST ELEMENT
          mov   a,r2                  //update larger number in r1
          mov   r1,a
skip:     djnz  r3,nextbyte
          mov   dptr, #4062H          //location of the result-4062h
          mov   a,r1                  //largest number
          movx  @dptr,a               //store at #4062H
end
```

- Add two 32 bit numbers starting at location 40H to 43H and 50H to 53H. Store the result at address 60H to 63H.

```
        CLR C
        MOV R0,#40H
        MOV R1,#50H
        MOV R2,#04
        SETB PSW.3
        MOV R0,#60H
        CLR PSW.3
Back:   MOV A,@R0
        ADDC A,@R1
        INC R0
        INC R1
        SETB PSW.3
        MOV @R0,A
        INC R0
        CLR PSW.3
        DJNZ R2,Back
```

. Write an assembly language program to find whether given eight bit number is odd or even. If odd store 00h in accumulator. If even store FFh in accumulator.

```
    mov    a,20h              // 20h=given number, to find
                                 is it even or odd
    jb  acc.0,odd       //jump if direct bit is set i.e., if lower
                              bit is 1 then number is odd
    mov   a,#0FFh
    sjmp   ext
odd:  mov   a,#00h
Ext:end
```

- Compute the logical AND of the input signals on bits 0 and 1 of port 1 and output the result to bit 2 of port 1.
- Soln

```
        SETB P1.0
        SETB P1.1
  Loop: MOV C,P1.0
        ANL C,P1.1
        MOV P1.2,C
        SJMP Loop
```

- Write a program to perform the following:

i. Keep monitoring pin P0.1 until becomes high

Ii. When P0.1 becomes high, read in the data from Port 1.

Iii. Send a low to high pulse on P0.2 to indicate that the data has been read.

```
        SETB P0.1
        MOV P1,#0FFH
Here: JNB P0.1, Here
        MOV A,P1
        CLR P0.2
        SETB P0.2
        END
```

- Assuming that ROM space starting at 250H contains “DDU”, write a program to transfer the bytes into RAM location starting at 40H.

```
        ORG 0000
        MOV DPTR,#Mydata
        MOV R0,#40H
        MOV R2,#3
Back:   CLR A
        MOVC A,@A+DPTR
        MOV @R0,A
        INC DPTR
        INC R0
        DJNZ R2,Back
Here:   SJMP Here

        ORG 250H
        Mydata DB “DDU”
        END
```

. **Write an assembly language program to implement (display) an eight bit UP/DOWN binary (hex) counter on watch window.**

```
          mov   a,#00                        //mov a, #0ffh for down counter
back:    acall   delay
           inc   a                           //dec a for binary down counter
          jnz   back
here:    sjmp   here
delay:     mov   r1,.#0ffh
decr1:    mov   r2,#0ffh
decr:    mov   r3,#0ffh
Here1: djnz   r3,here1
        djnz   r2,decr
        djnz   r1,decr1
        ret
end
```