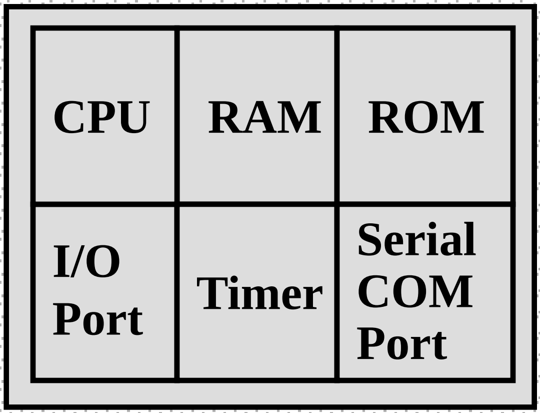
The 8051 Microcontroller

8051 Basic Component

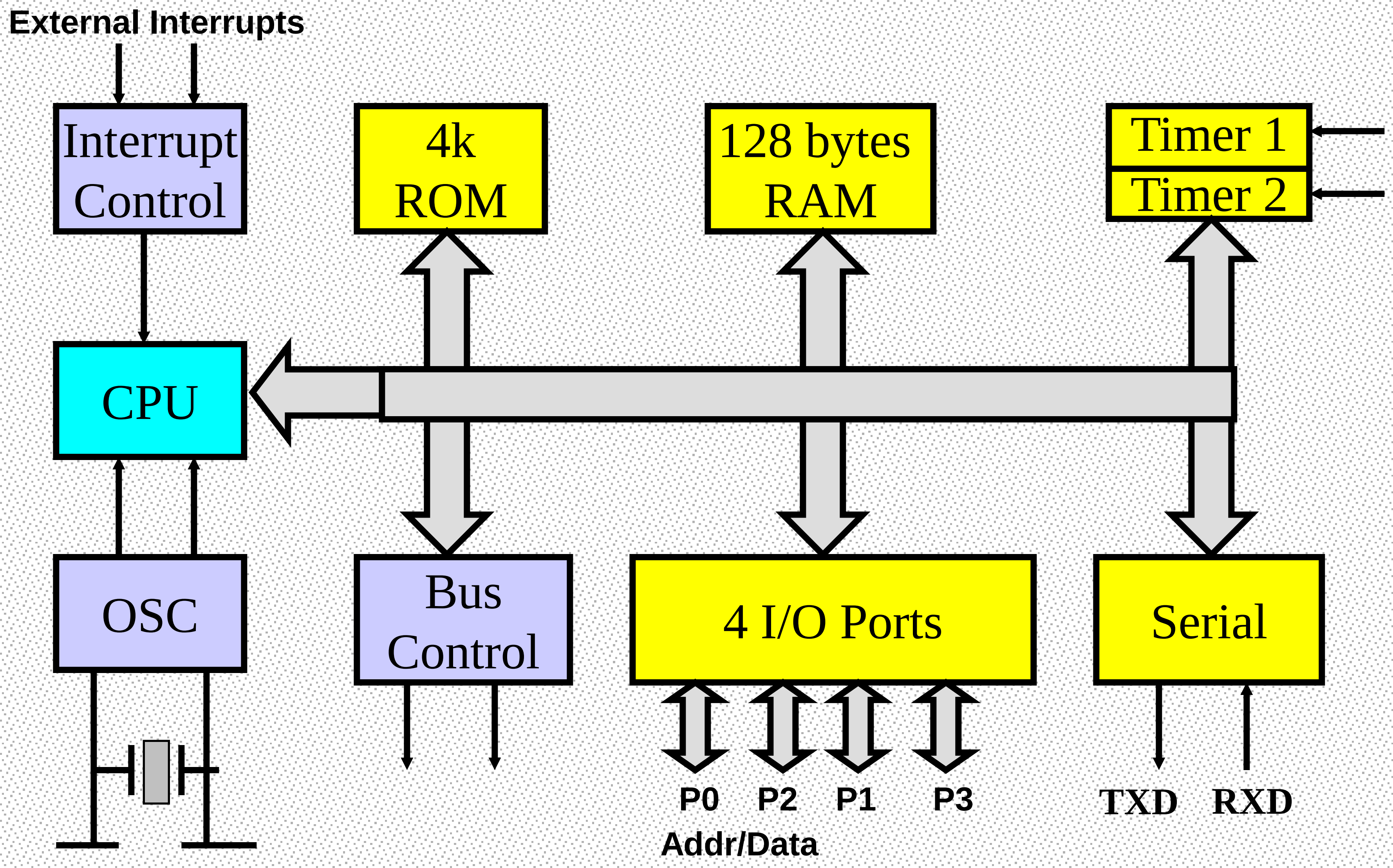
- 4K bytes internal ROM
- 128 bytes internal RAM
- Four 8-bit I/O ports (P0 - P3).
- Two 16-bit timers/counters
- One serial interface



CPU	RAM	ROM
I/O Port	Timer	Serial COM Port

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 P2 P1 P3

Addr/Data

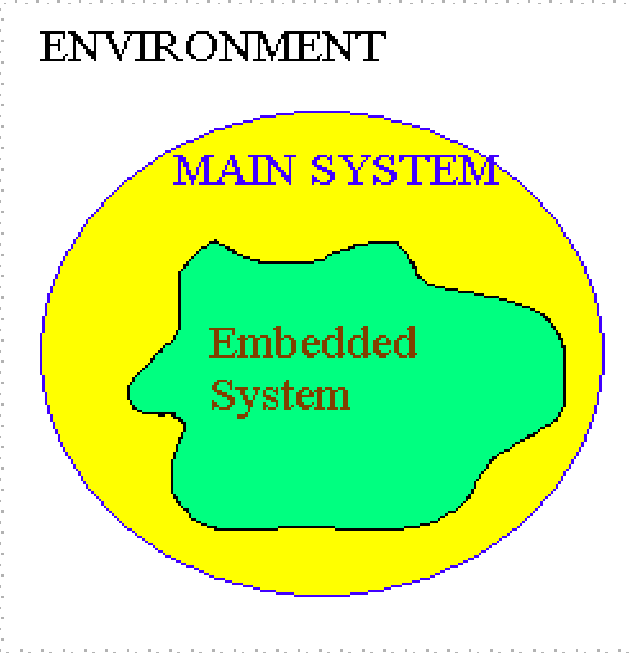
TXD RXD

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

Embedded System
(8051 Application)

- What is Embedded System?
  - An embedded system is closely integrated with the main system
  - It may not interact directly with the environment
  - For example – A microcomputer in a car ignition control



- An embedded product uses a microprocessor or microcontroller to do one task only
- There is only one application software that is typically burned into ROM

Examples of Embedded Systems

- ❑ Keyboard
- ❑ Printer
- ❑ video game player
- ❑ MP3 music players
- ❑ Embedded memories to keep configuration information
- ❑ Mobile phone units
- ❑ Domestic (home) appliances
- ❑ Data switches
- ❑ Automotive controls

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

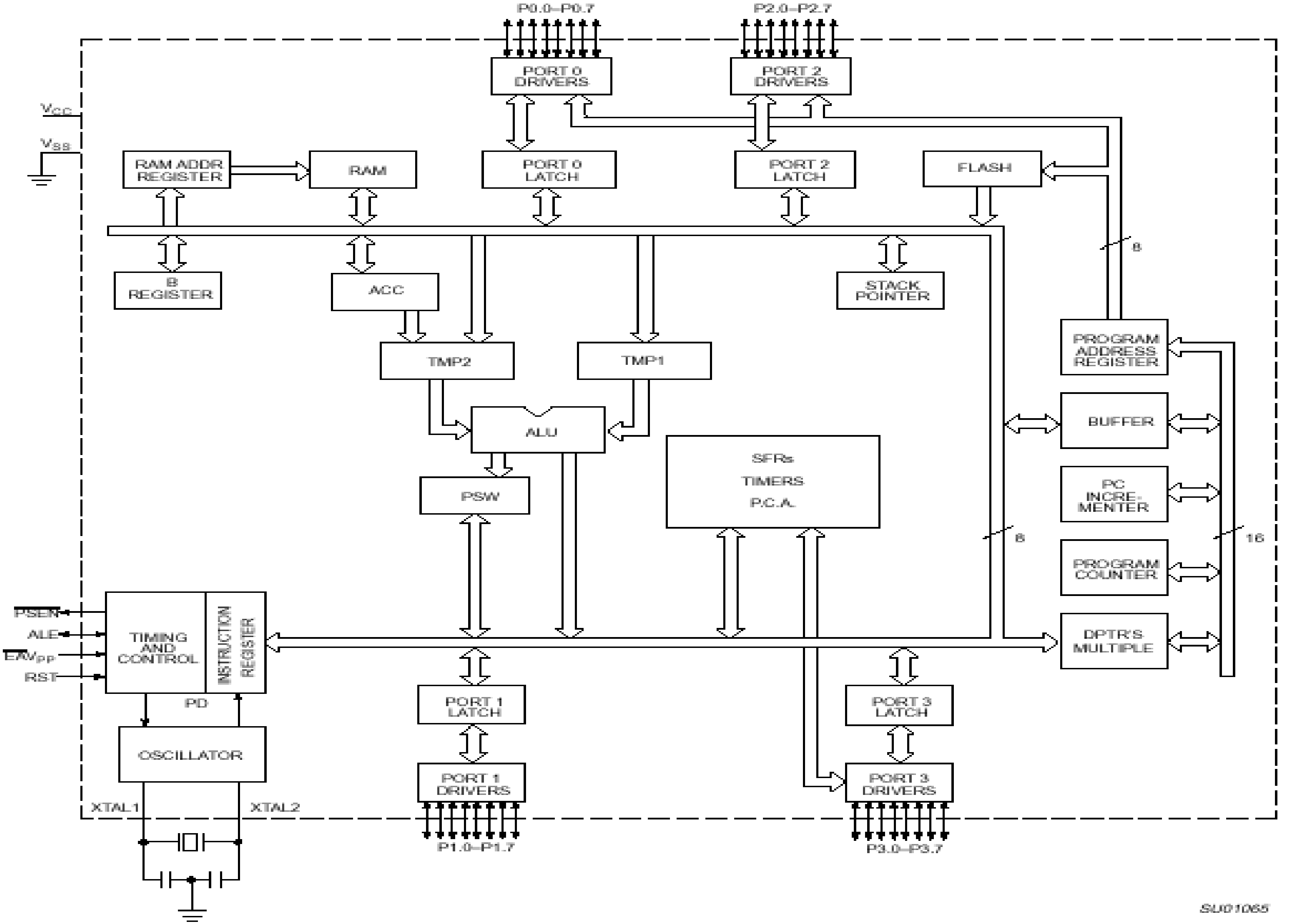
- ROM type
  - 8031 no ROM
  - 80xx mask ROM
  - 87xx EPROM
  - 89xx Flash EEPROM
- 89xx
  - 8951
  - 8952
  - 8953
  - 8955
  - 898252
  - 891051
  - 892051
- Example (AT89C51,AT89LV51,AT89S51)
  - AT= ATMEL(Manufacture)
  - C = CMOS technology
  - LV= Low Power(3.0v)

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 Internal Block Diagram



8051 Schematic Pin out

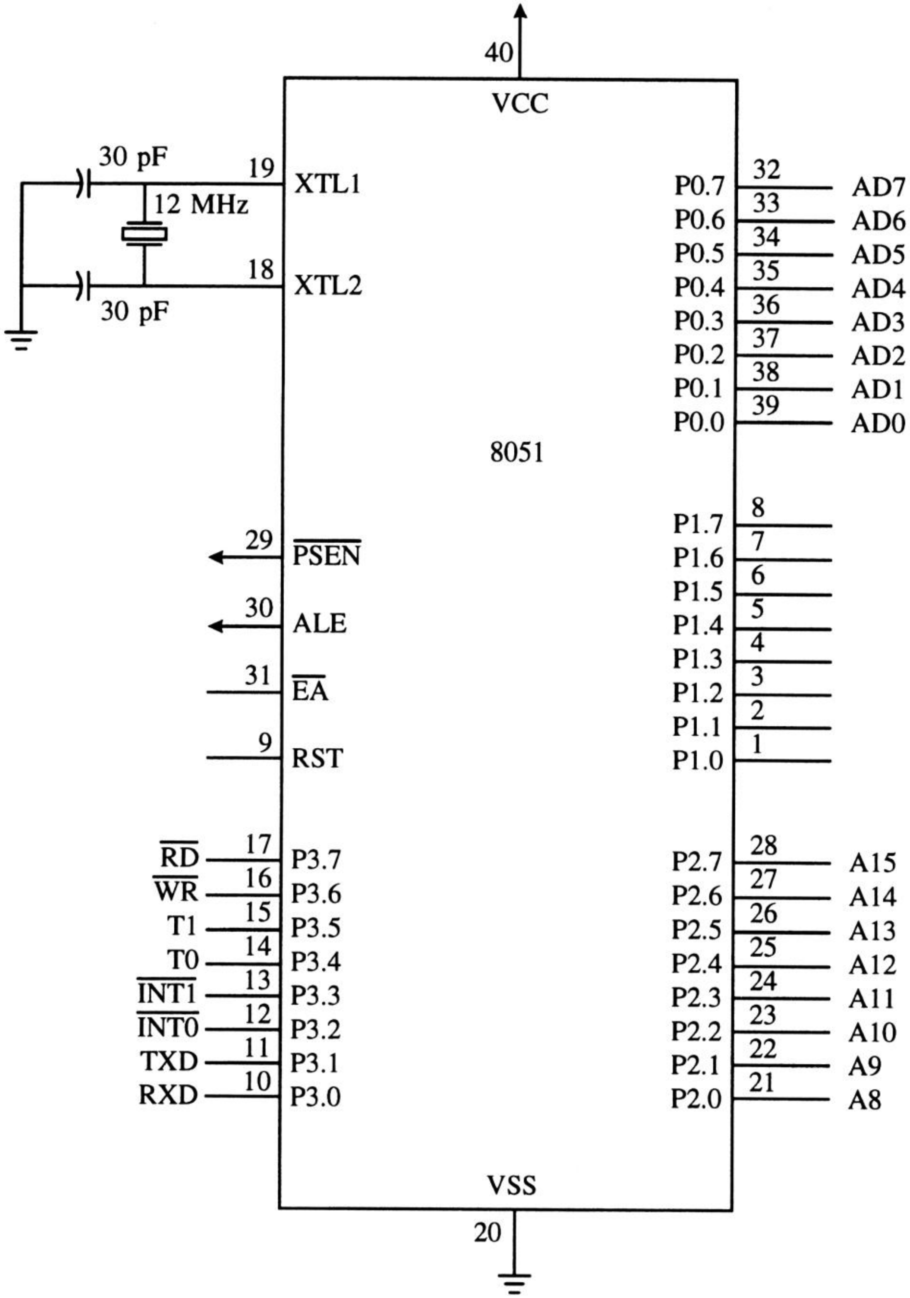
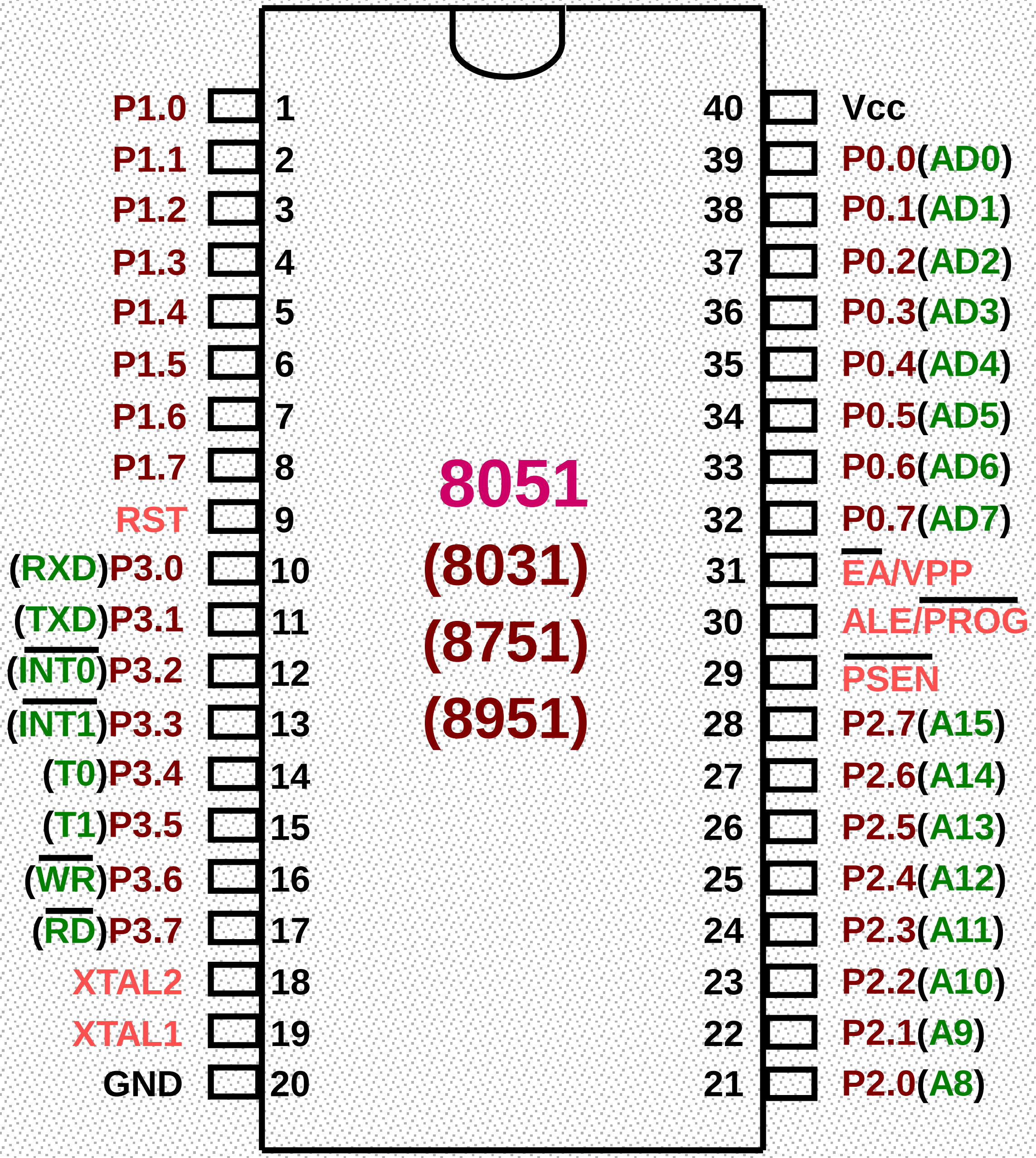


FIGURE 2–2
8051 pinouts

8051 Foot Print



8051

(8031)

(8751)

(8951)

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

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

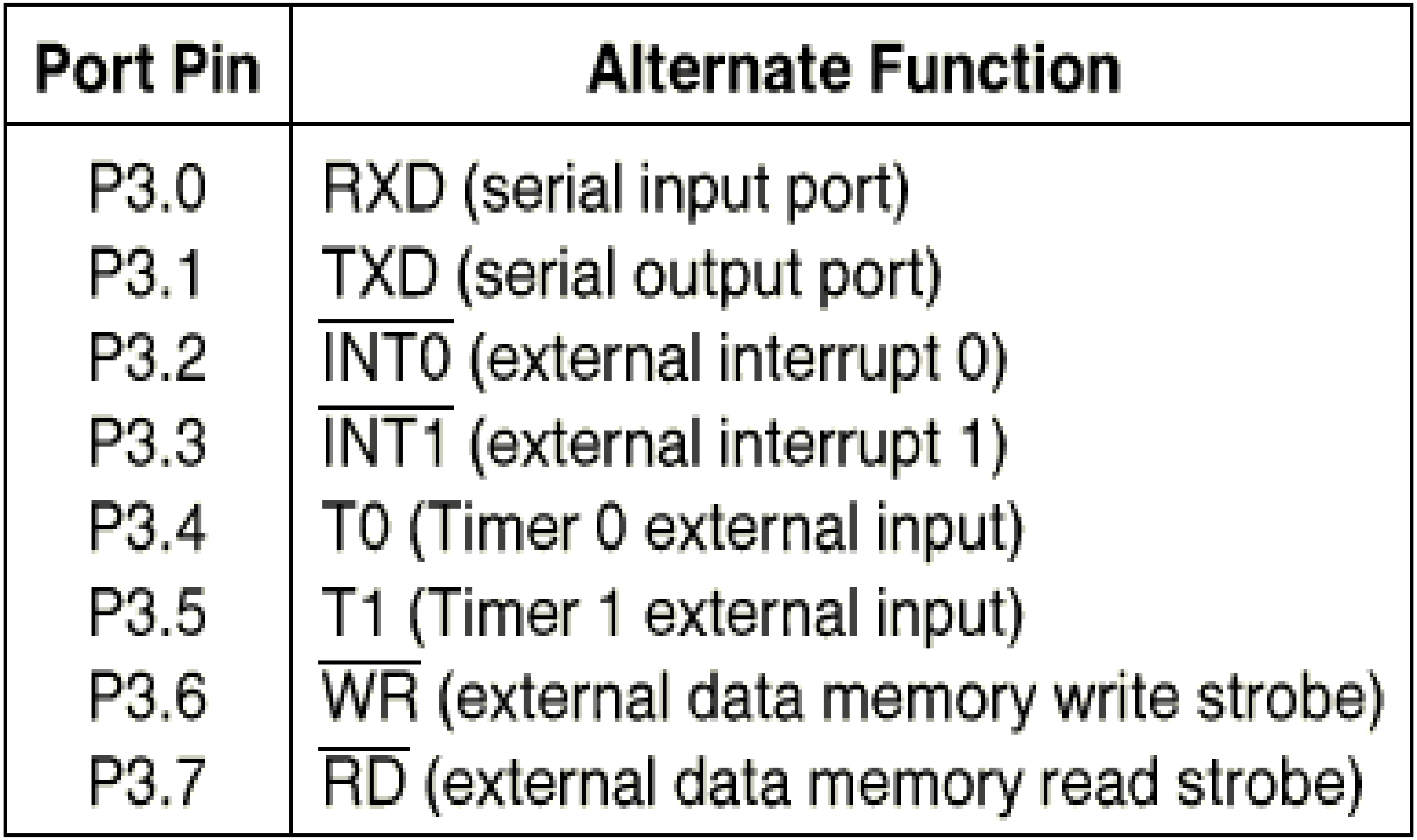
- Port 1 （pins 1-8） ：P1（P1.0～P1.7）
  - Only 8-bit R/W - General Purpose I/O

- 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.

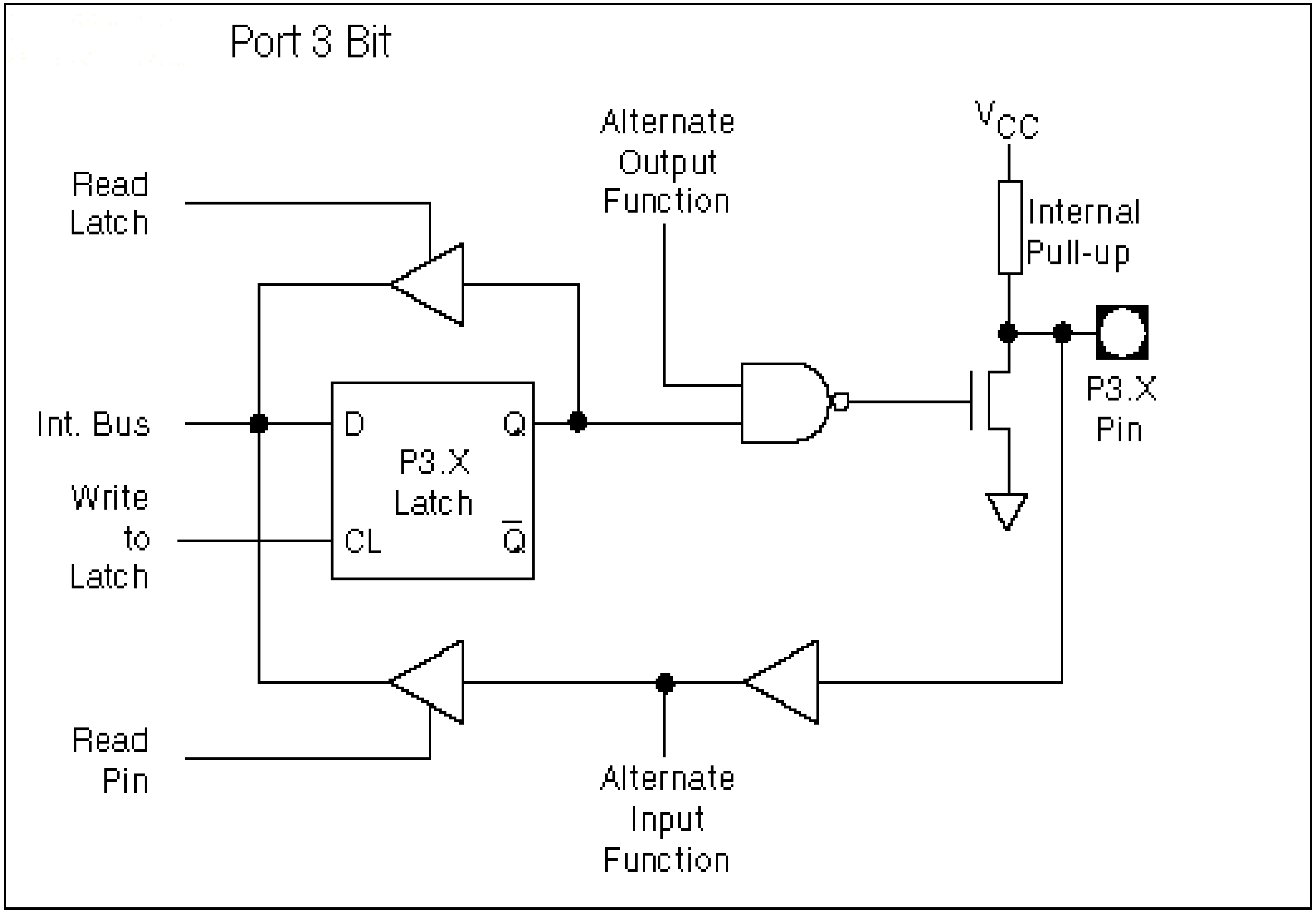
- **Each port can be used as input or output (bi-direction)**

Port 3 Alternate Functions

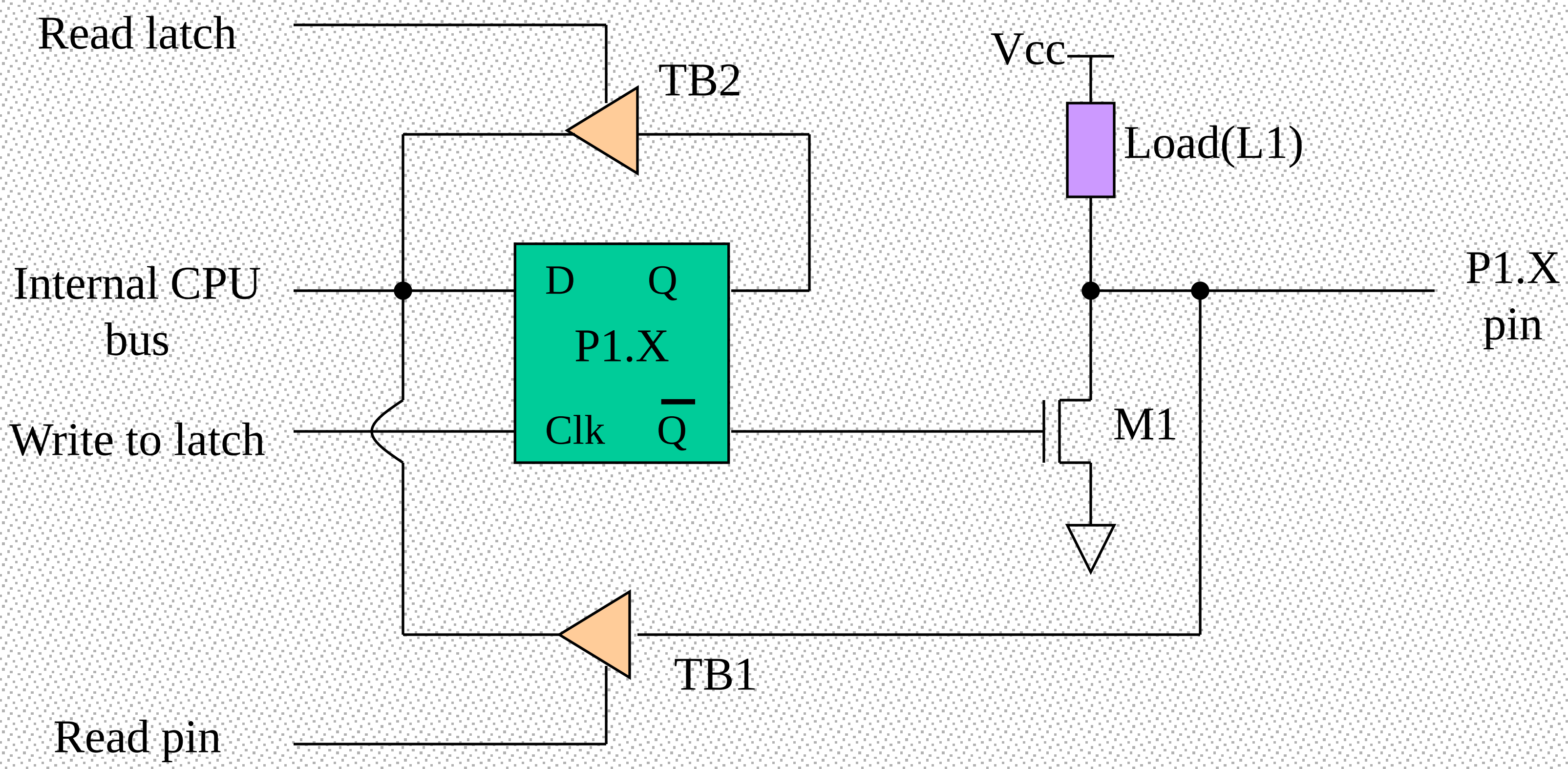


Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

8051 Port 3 Bit Latches and I/O Buffers



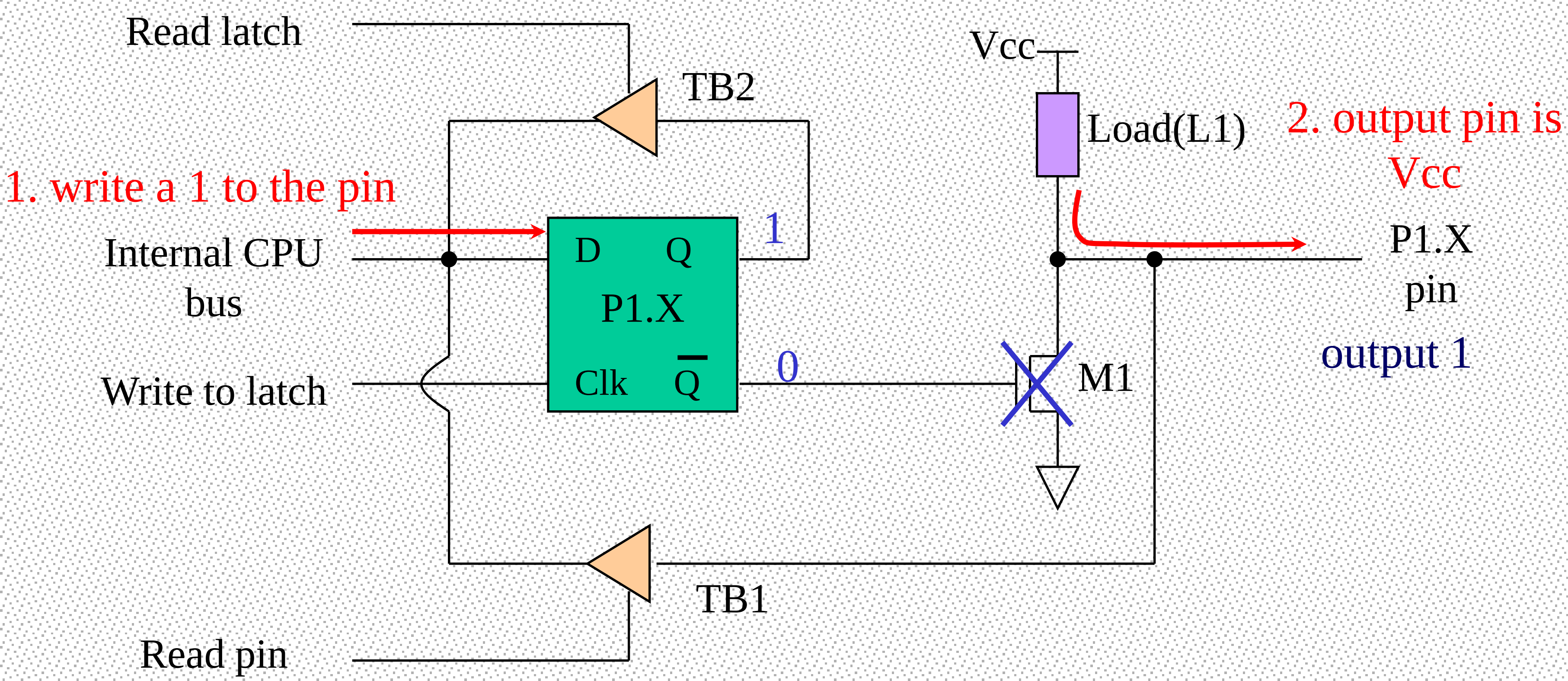
Hardware Structure of I/O Pin



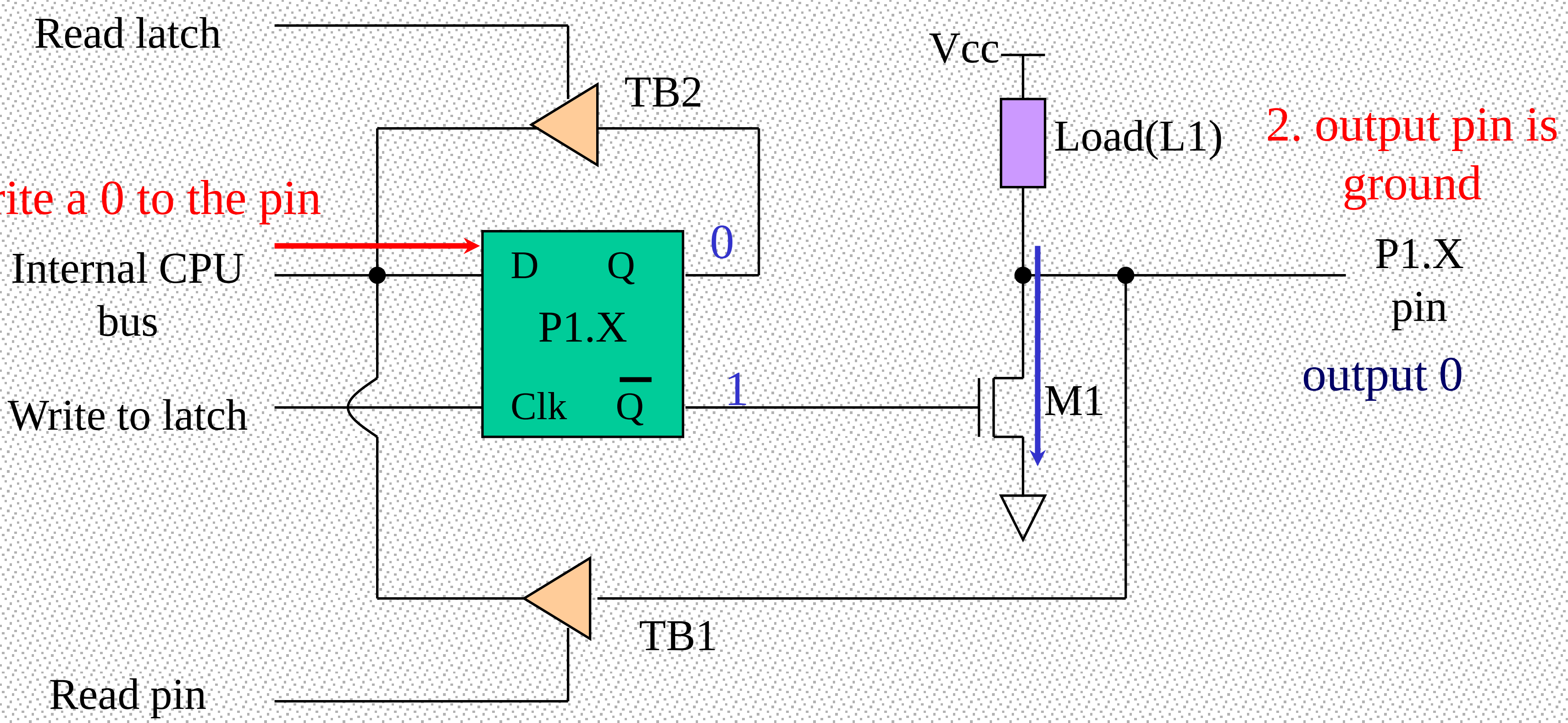
Hardware Structure of I/O Pin

- 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

Writing "1" to Output Pin P1.X



Writing “0” to Output Pin P1.X

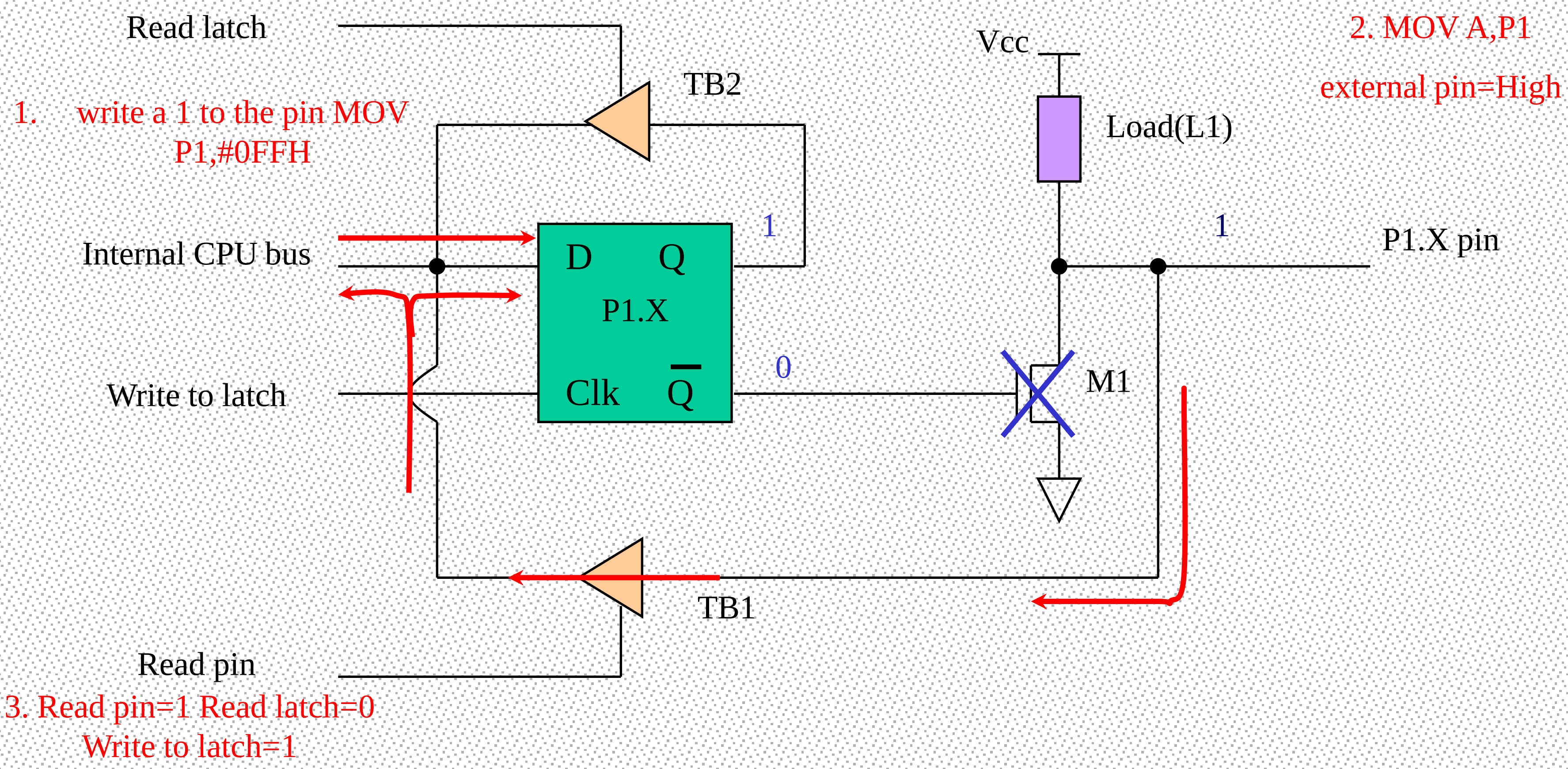


1. write a 0 to the pin

2. output pin is ground

output 0

Reading “High” at Input Pin



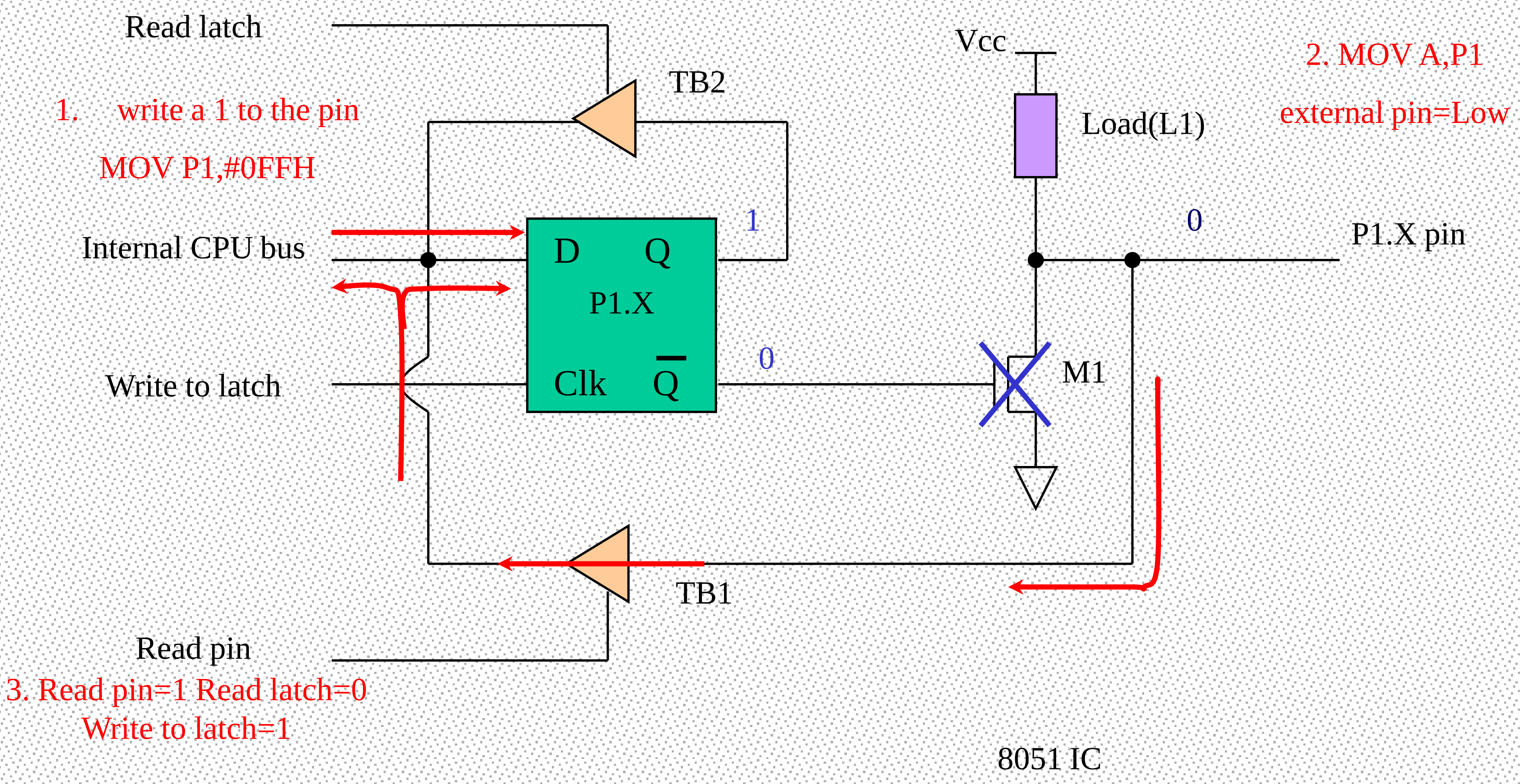
1. write a 1 to the pin MOV P1,#0FFH

2. MOV A,P1

external pin=High

3. Read pin=1 Read latch=0 Write to latch=1

Reading "Low" at Input Pin



1. write a 1 to the pin

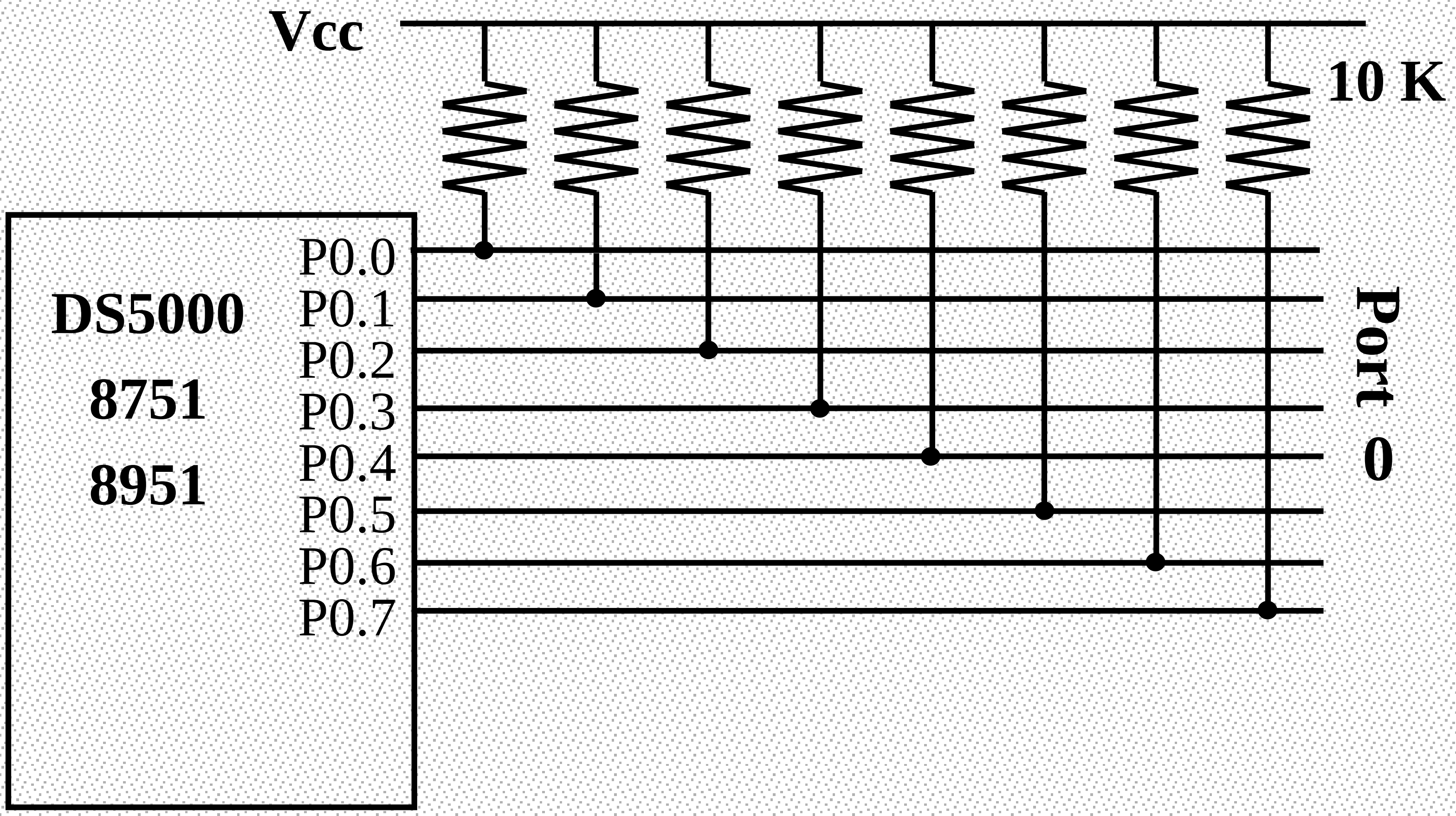
MOV P1,#0FFH

2. MOV A,P1

external pin=Low

3. Read pin=1 Read latch=0
Write to latch=1

Port 0 with Pull-Up Resistors



IMPORTANT PINS

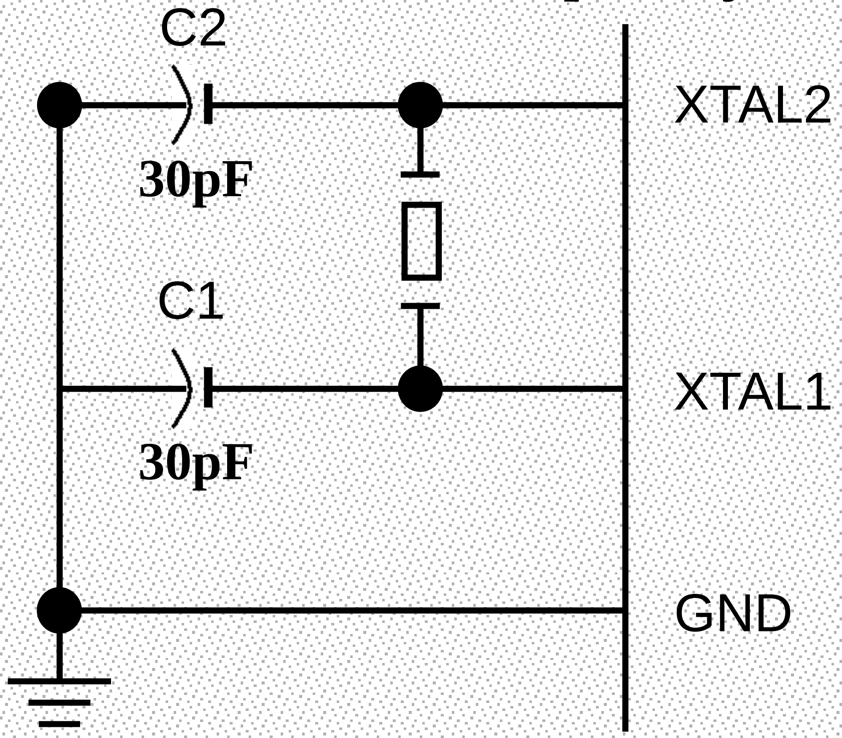
- PSEN (out): Program Store Enable, the read signal for external program memory (active low).
- ALE (out): Address Latch Enable, to latch address outputs at Port0 and Port2
- EA (in): External Access Enable, active low to access external program memory locations 0 to 4K
- RXD,TXD: UART pins for serial 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.

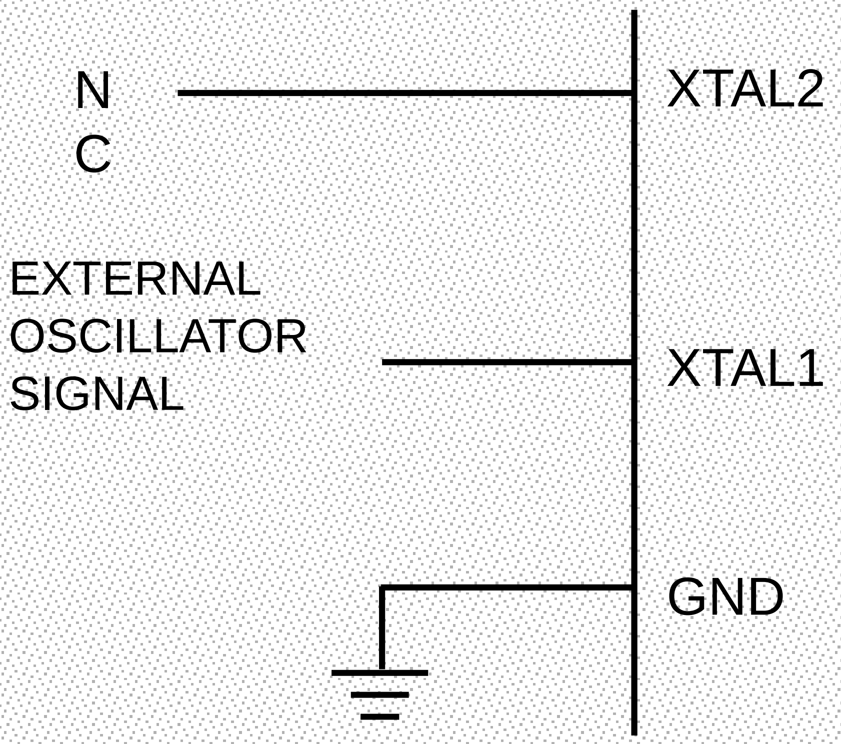
XTAL Connection to 8051

- 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.



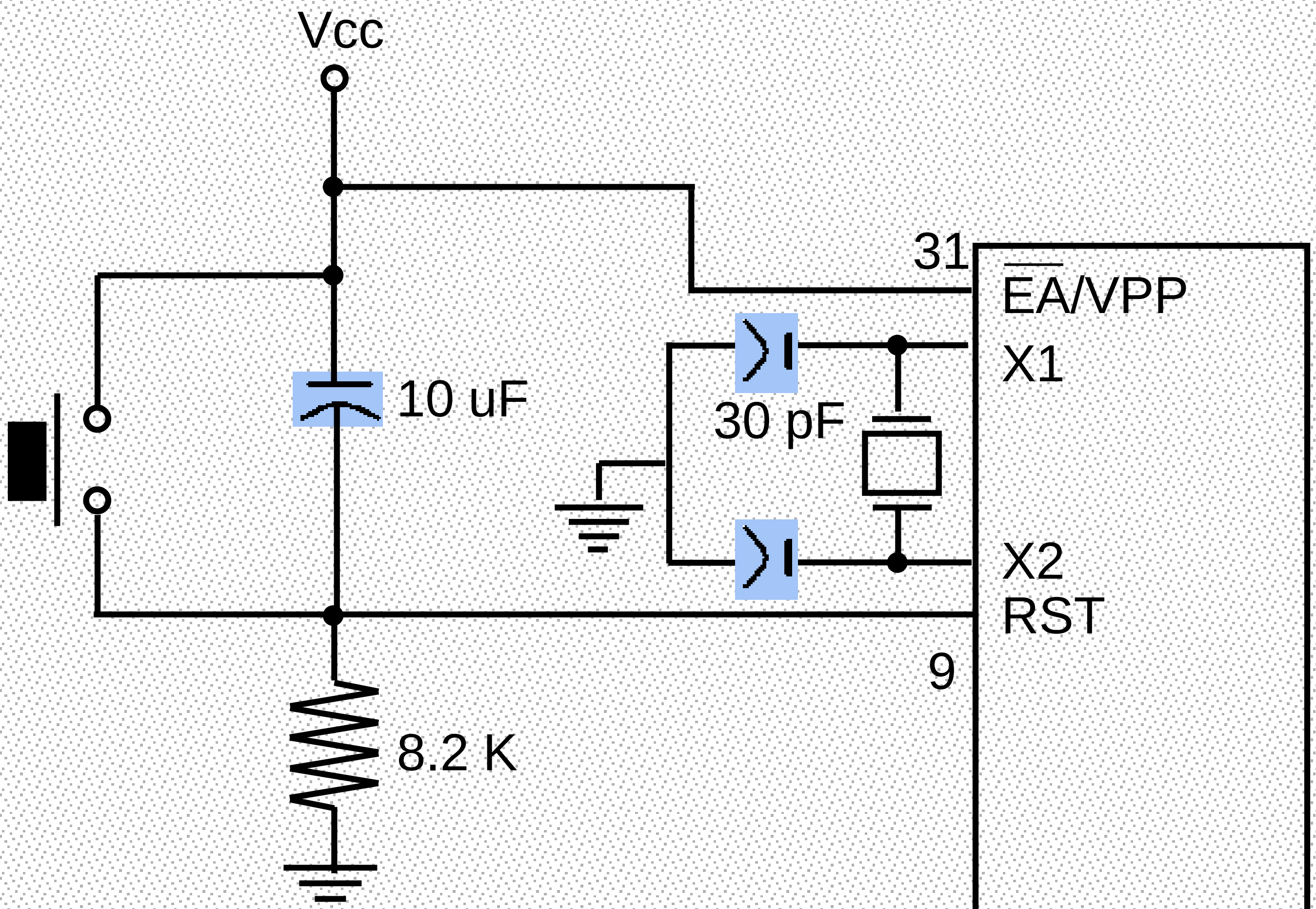
Machine cycle

- 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

Pins of 8051

- RST（pin 9） : reset
  - input pin and active high（normally low）.
    - The high pulse must be high at least 2 machine cycles.
  - power-on reset.
    - Upon applying a high pulse to RST, the microcontroller will reset and all values in registers will be lost.
    - Reset values of some 8051 registers
  - power-on reset circuit

Power-On RESET



RESET Value of Some 8051 Registers:

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0007
DPTR	0000

RAM are all zero

Pins of 8051

- /EA（pin 31）：external access
  - There is no on-chip ROM in 8031 and 8032 .
  - The /EA pin is connected to GND to indicate the code is stored externally.
  - /PSEN ＆ ALE are used for external ROM.
  - For 8051, /EA pin is connected to Vcc.
  - “/” means active low.
- /PSEN（pin 29）：program store enable
  - This is an output pin and is connected to the OE pin of the ROM.
  - See Chapter 14.

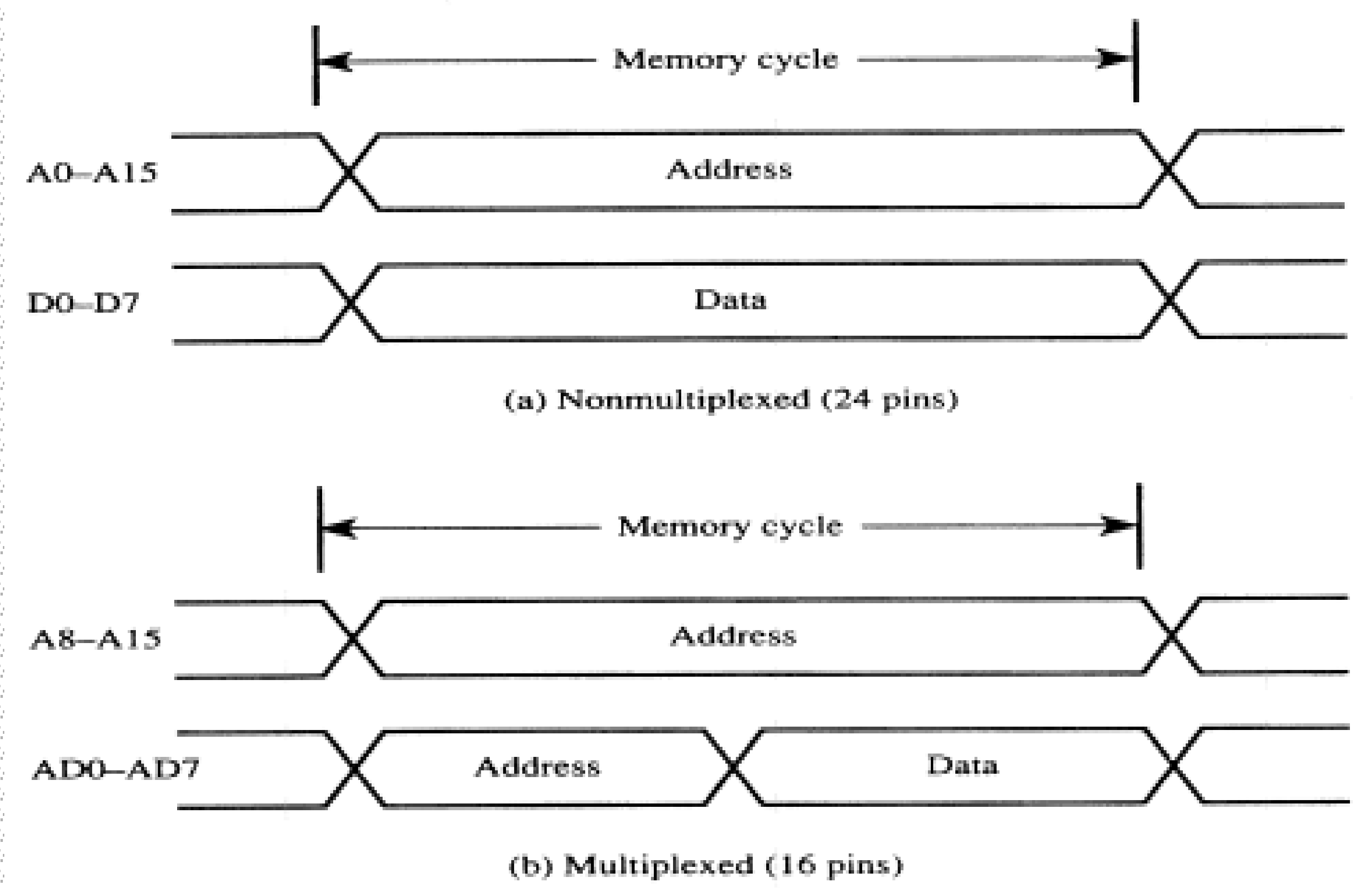
Pins of 8051

- ALE（pin 30）：address latch enable
  - It is an output pin and is active high.
  - 8051 port 0 provides both address and data.
  - The ALE pin is used for de-multiplexing the address and data by connecting to the G pin of the 74LS373 latch.

Address Multiplexing for External Memory

Figure 2-7

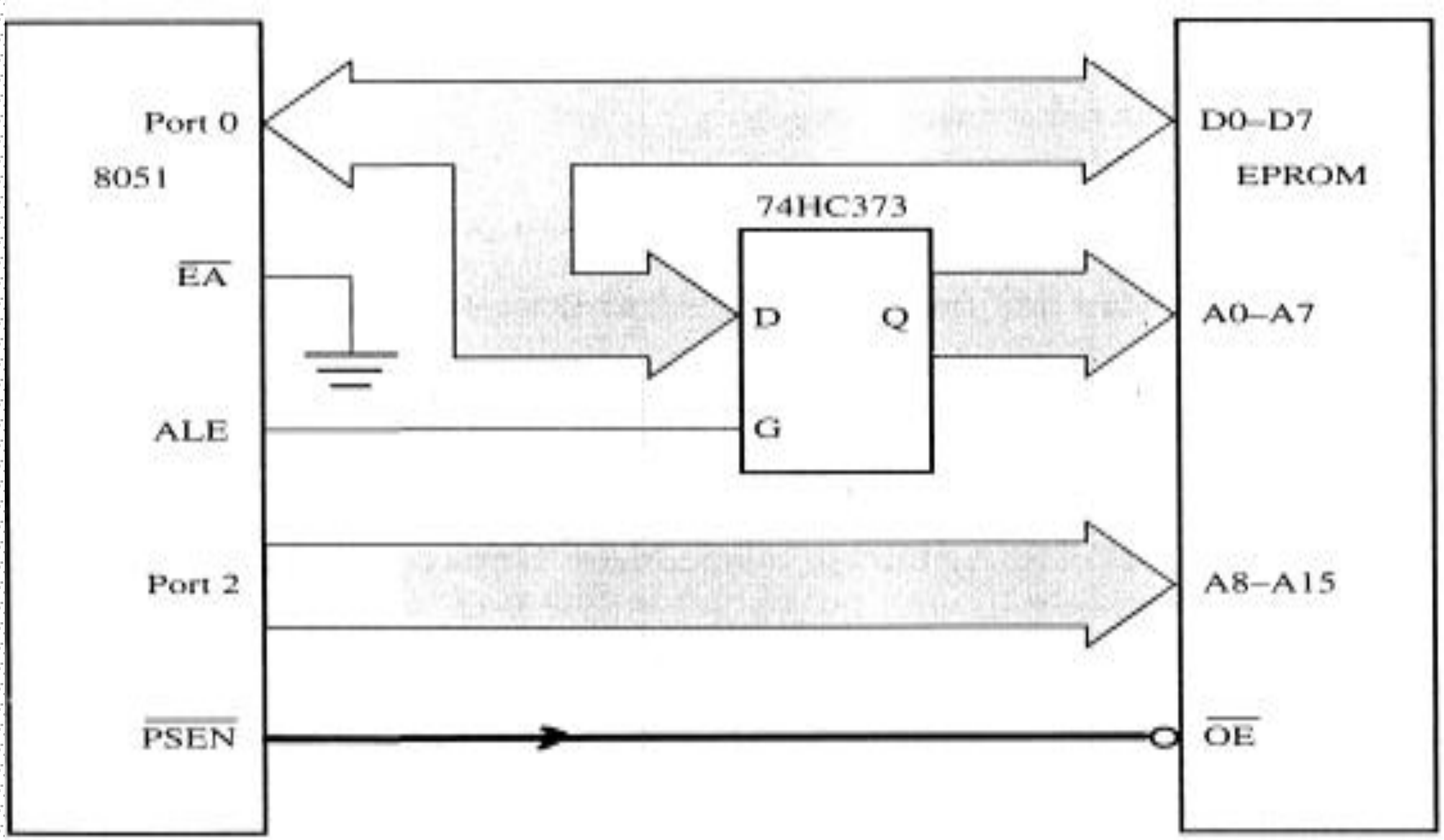
Multiplexing the address (low-byte) and data bus



Address Multiplexing for External Memory

Figure 2-8

Accessing external code memory



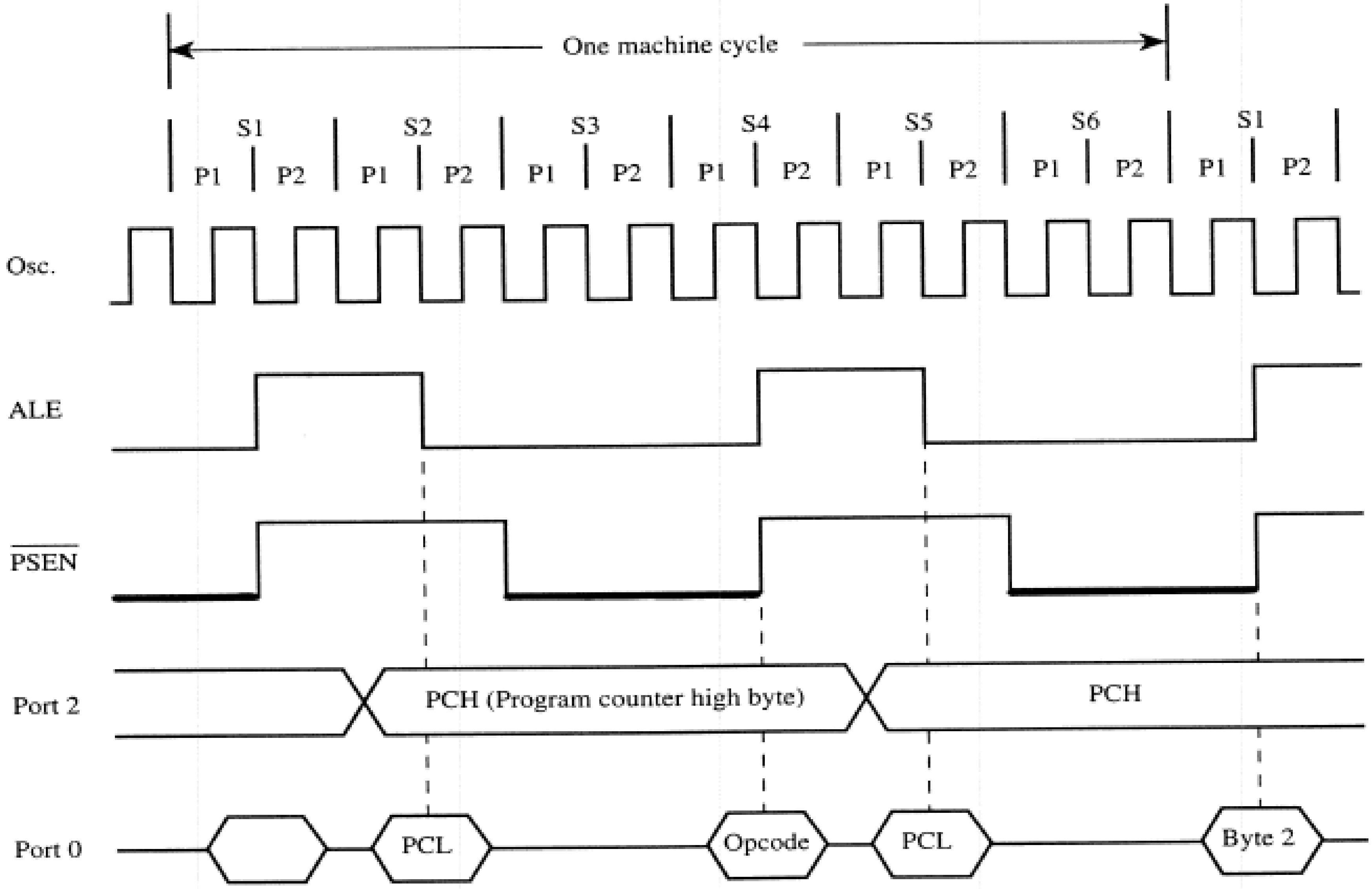
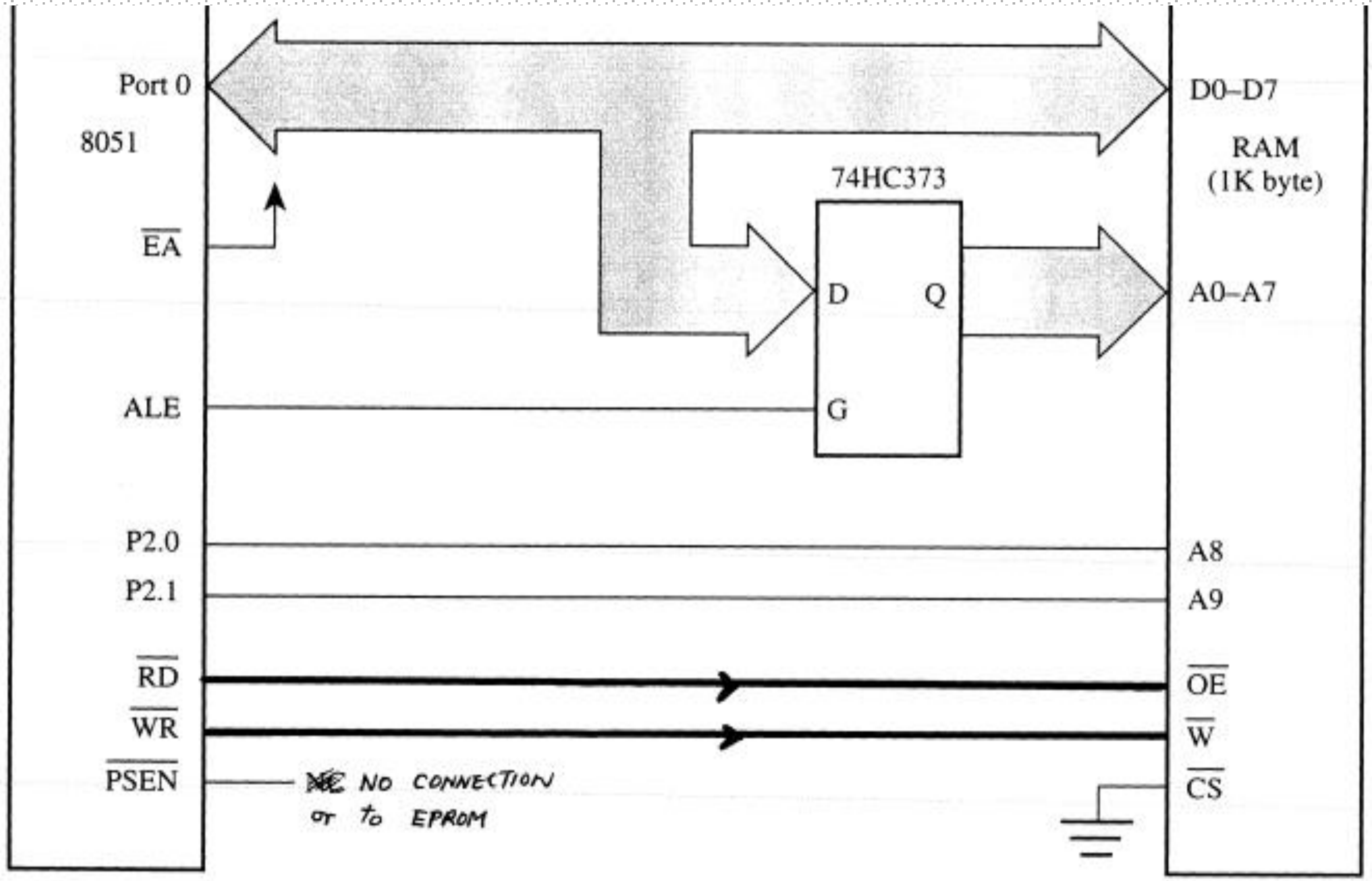


FIGURE 2–9
Read timing for external code memory

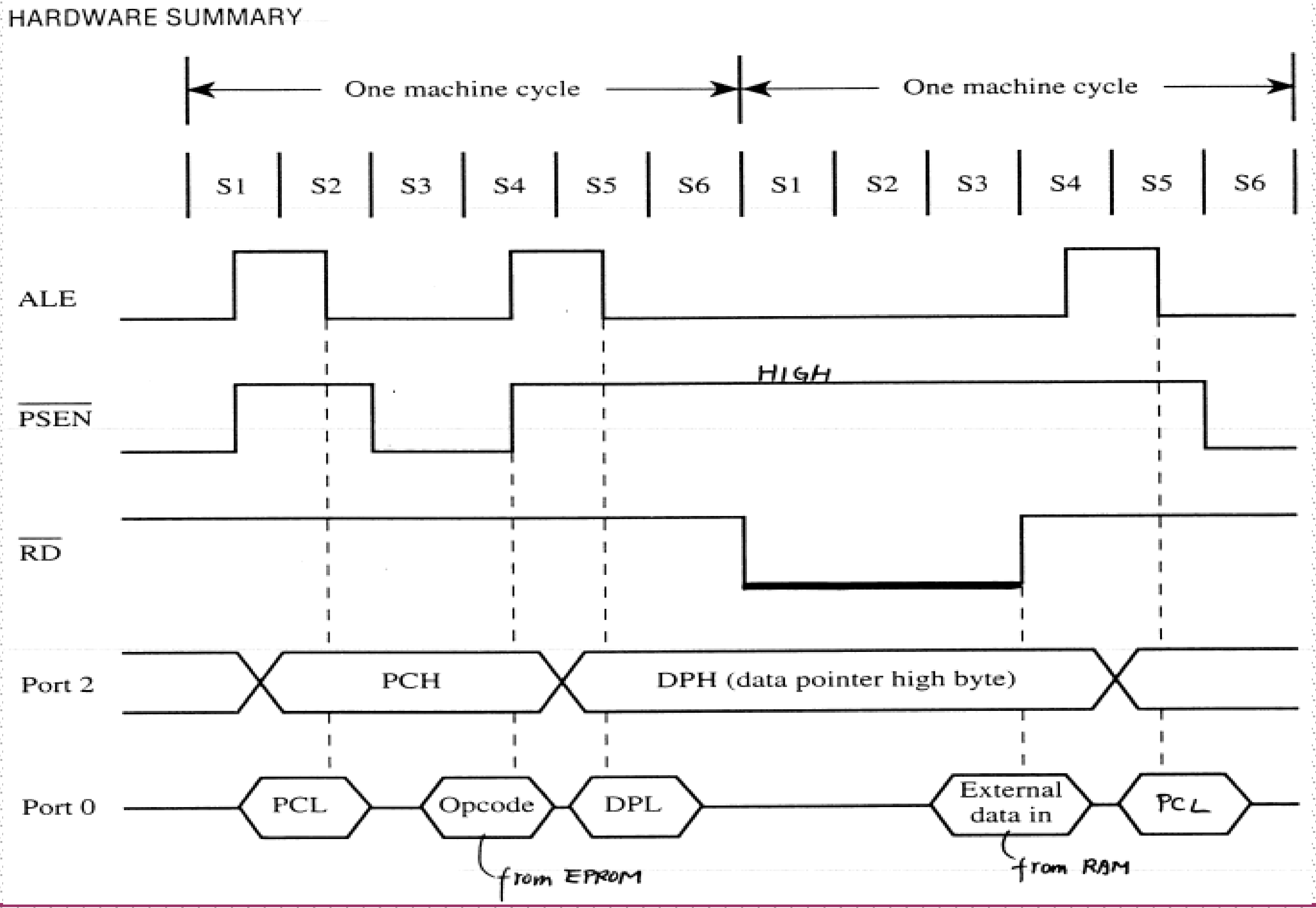
Accessing External Data Memory

Figure 2-11

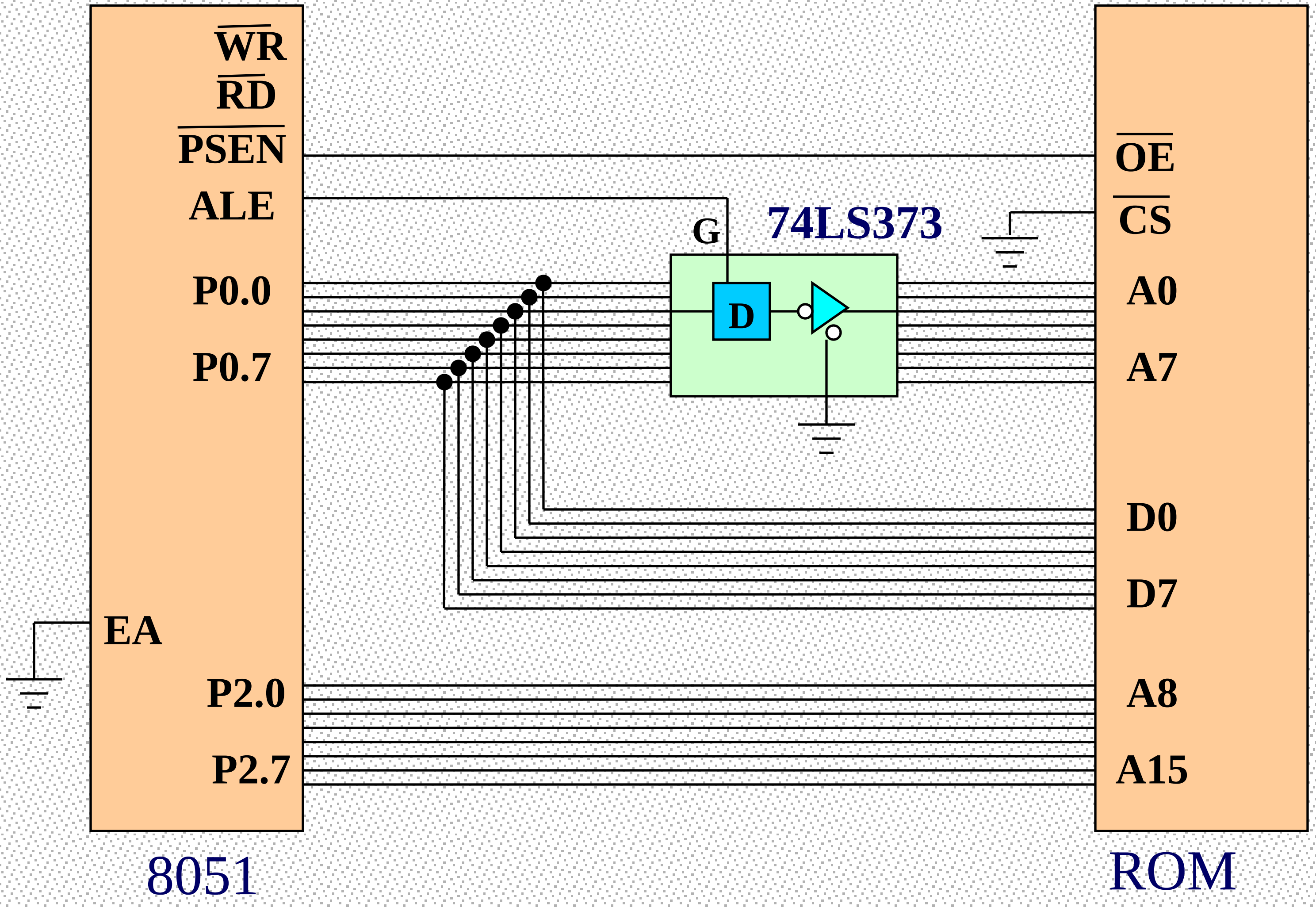
Interface to 1K RAM



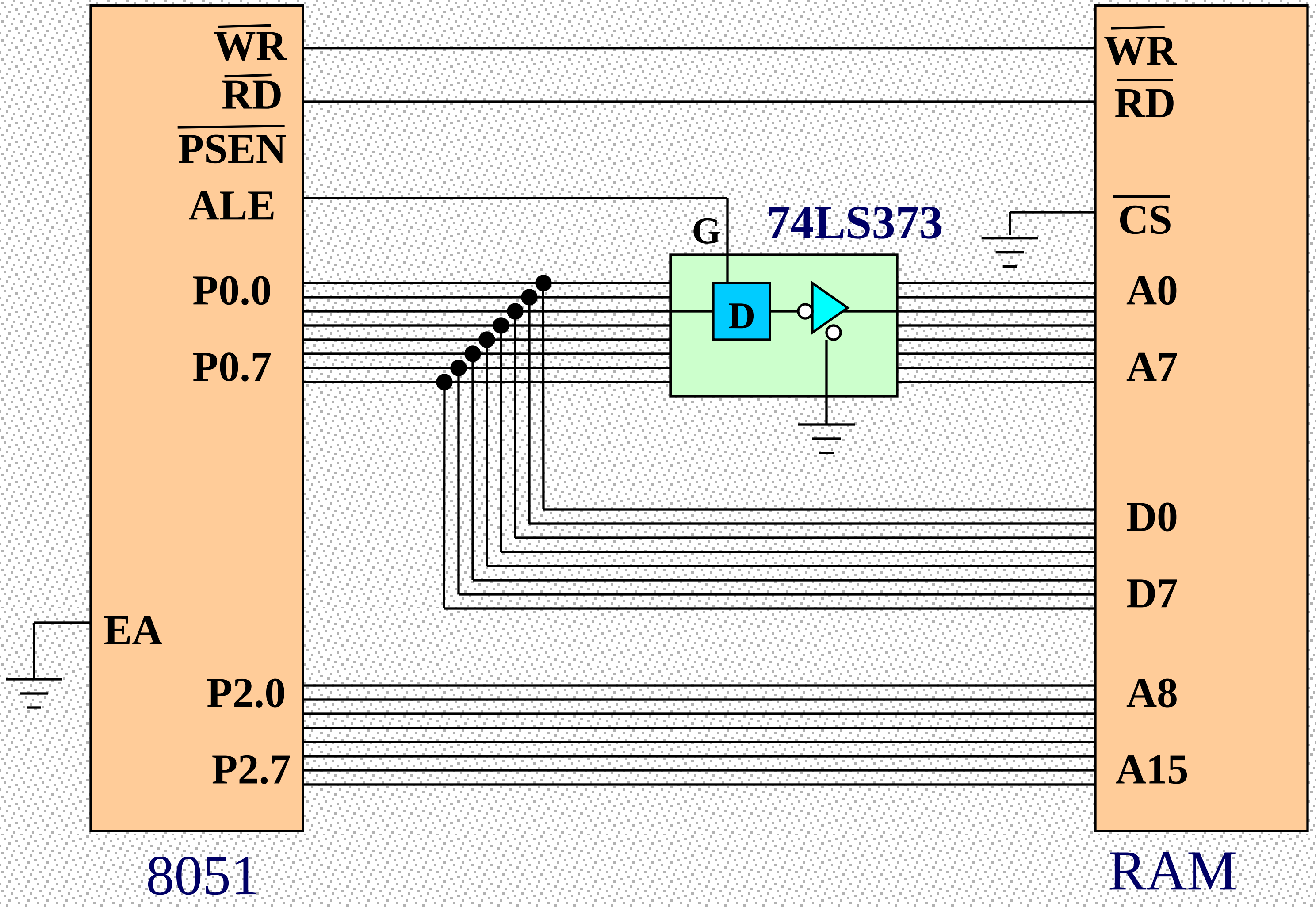
Timing for MOVX instruction



External code memory



External data memory



Overlapping External Code and Data Spaces

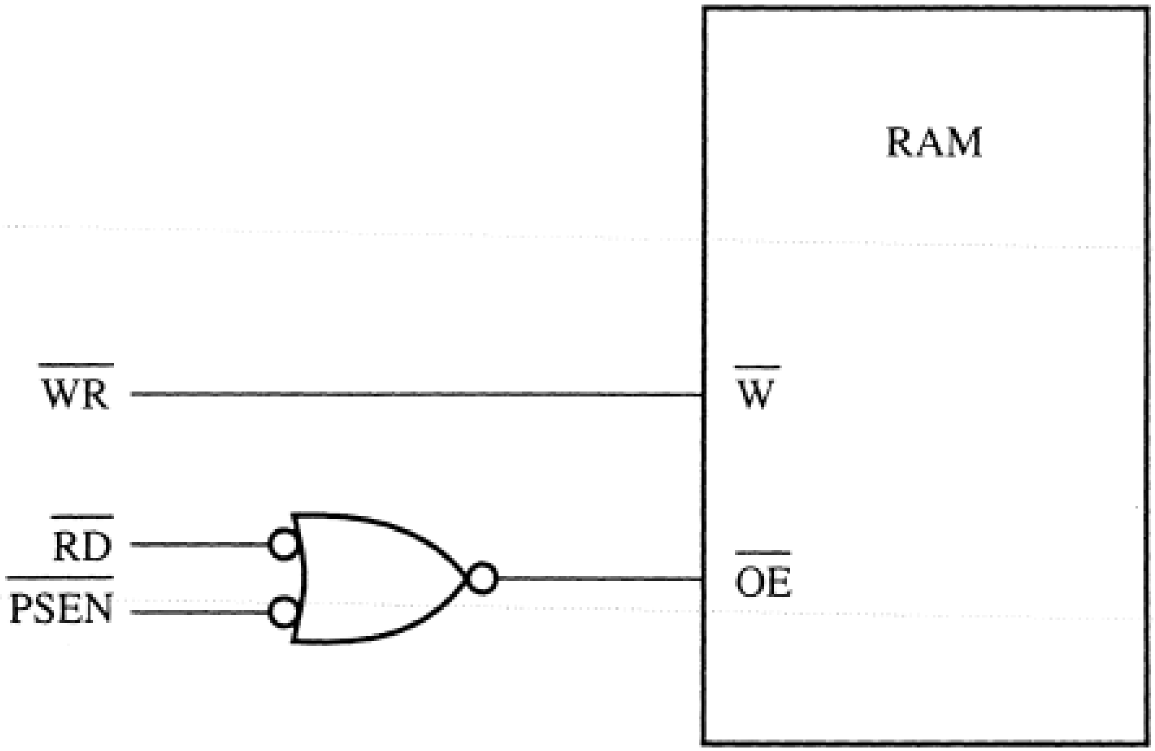
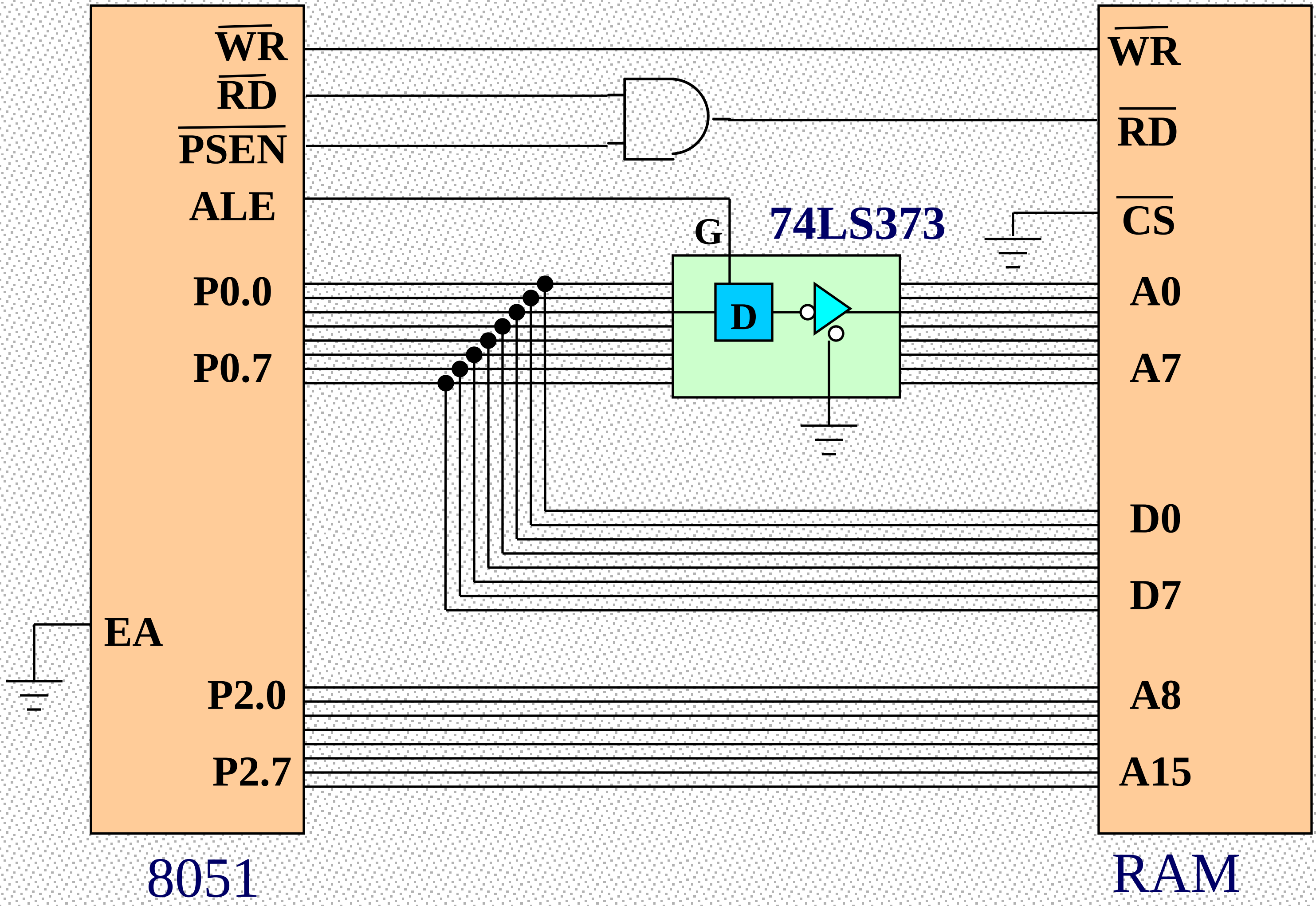


FIGURE 2–13
Overlapping the external code and data spaces

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.

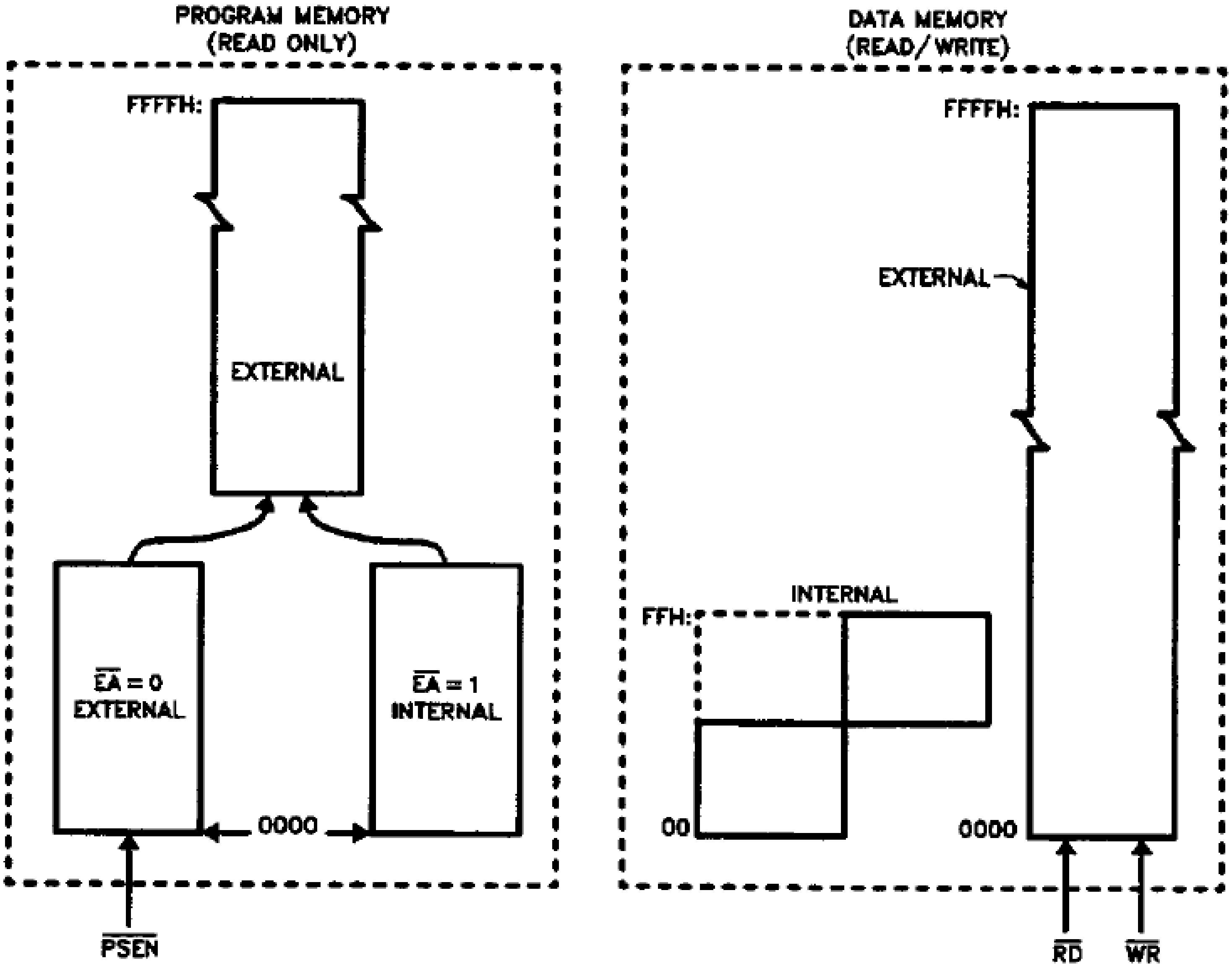
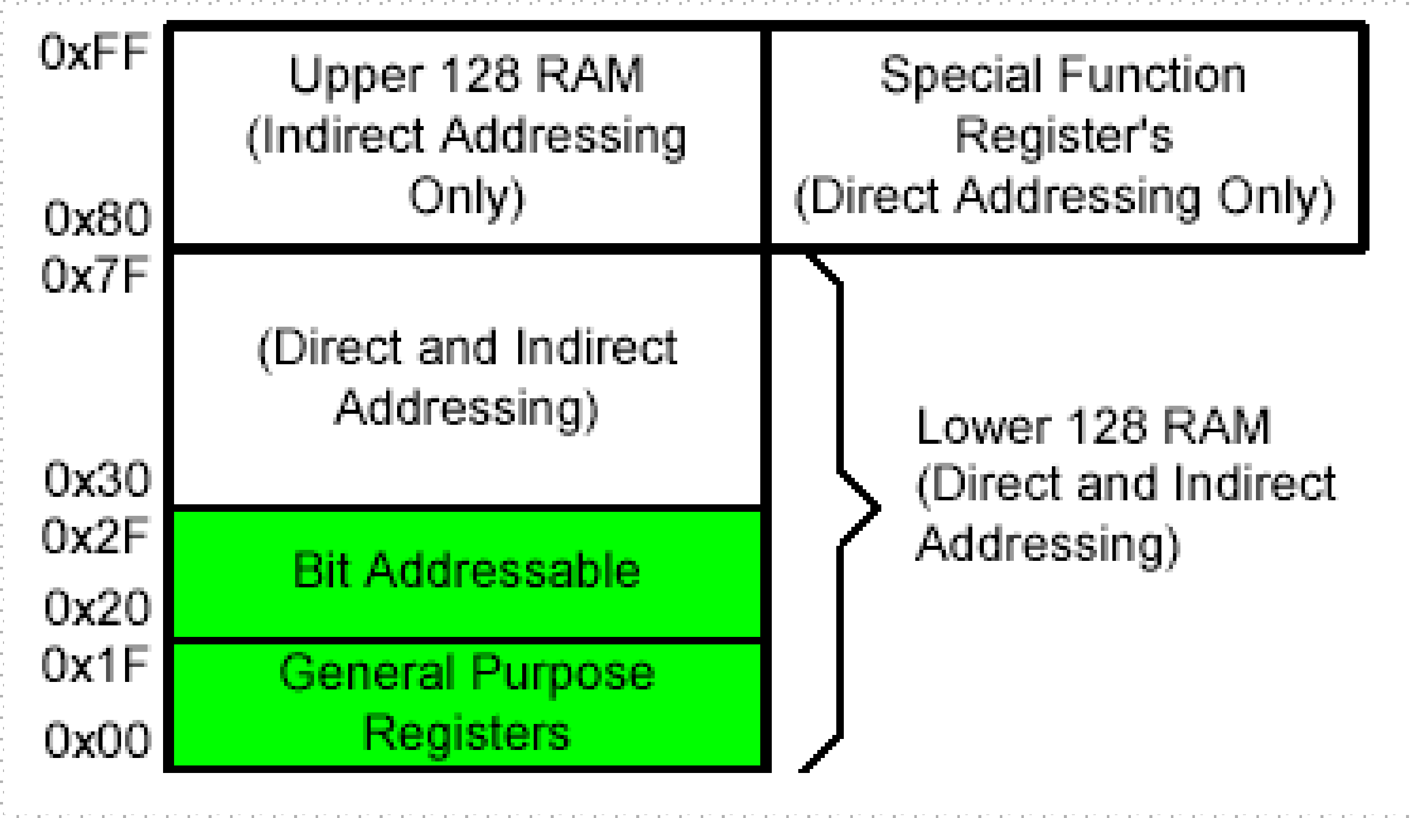


Figure 2. MCS®-51 Memory Structure

On-Chip Memory
Internal RAM



0xFF

Upper 128 RAM (Indirect Addressing Only)

Special Function Register's (Direct Addressing Only)

0x80

0x7F

(Direct and Indirect Addressing)

0x30

0x2F

Bit Addressable

0x20

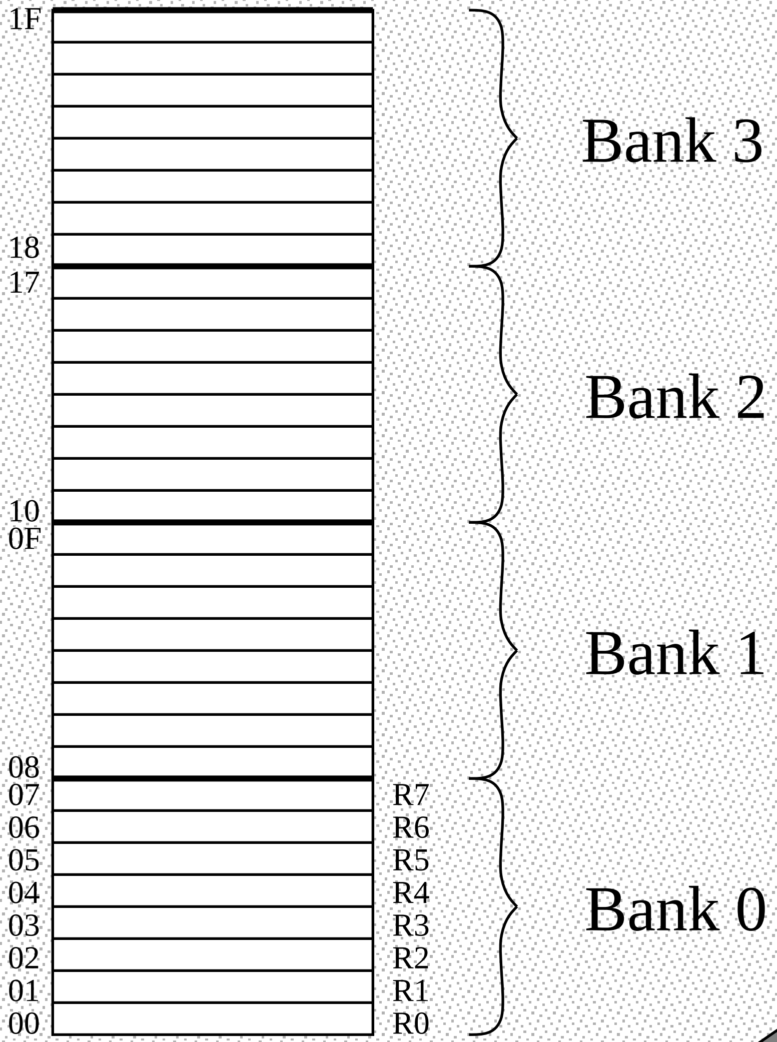
0x1F

General Purpose Registers

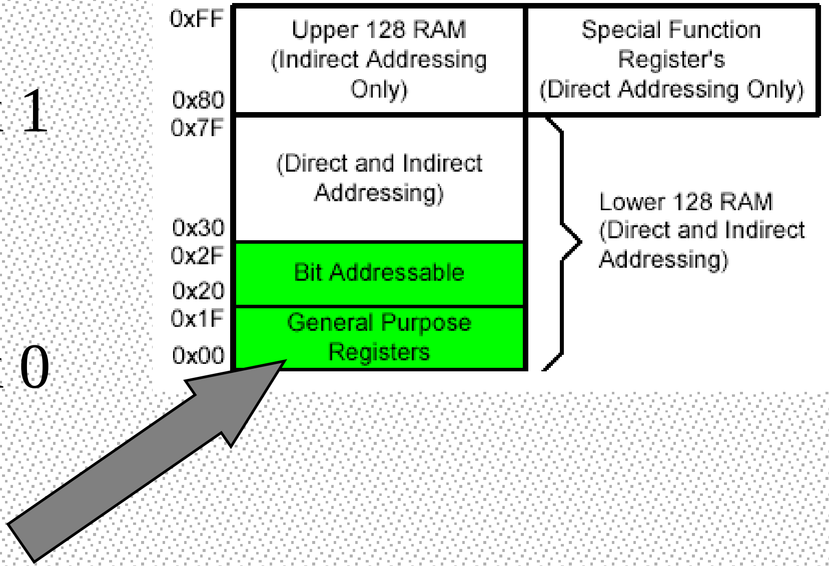
0x00

Lower 128 RAM (Direct and Indirect Addressing)

Registers



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



Bit Addressable Memory

2F	7F							78
2E								
2D								
2C								
2B								
2A								
29								
28								
27								
26								
25								
24								
23						1A		
22								10
21	0F							08
20	07	06	05	04	03	02	01	00

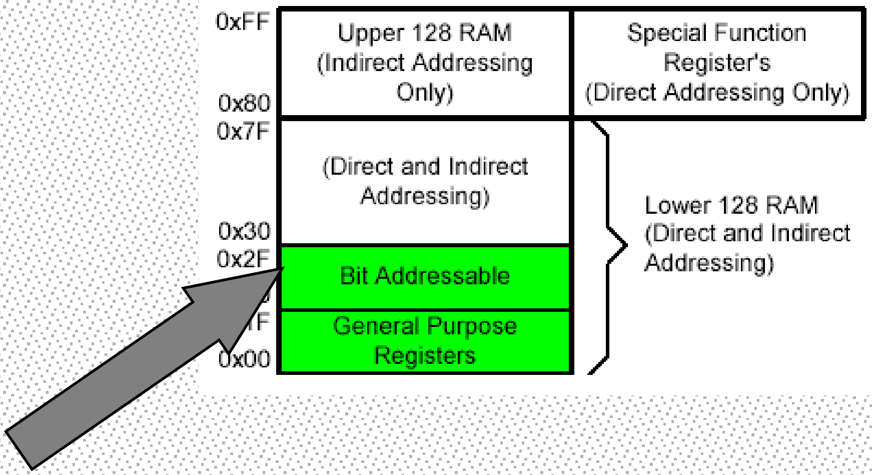
20h – 2Fh (16 locations X 8-bits = 128 bits)

Bit addressing:

mov C, 1Ah

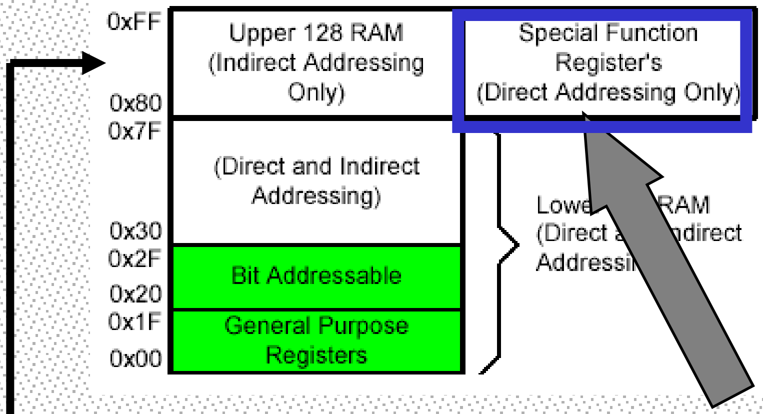
or

mov C, 23h.2


0xFF
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
0x80
0x7F
(Direct and Indirect Addressing)
Lower 128 RAM (Direct and Indirect Addressing)
0x30
0x2F
Bit Addressable
F
General Purpose Registers
0x00

Special Function Registers

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


0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lowe RAM
(Direct a ndirect
Addressi

Addresses 80h - FFh

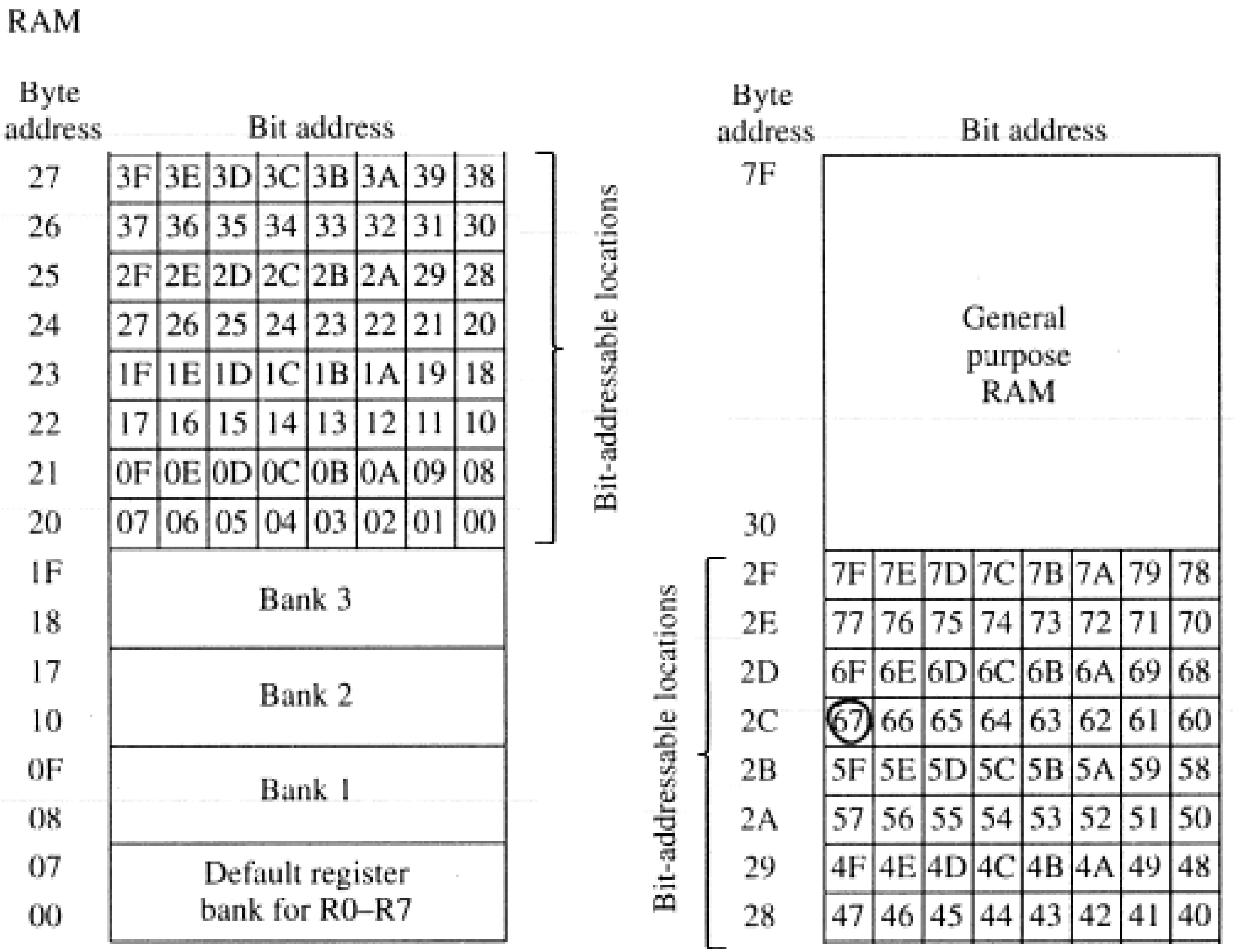
Direct Addressing used to access SPRs

Bit Addressable RAM

Figure 2-6

Summary of the 8051 on-chip data memory

(RAM)



RAM

Byte address	Bit address							
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							

Bit-addressable locations

Byte address	Bit 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

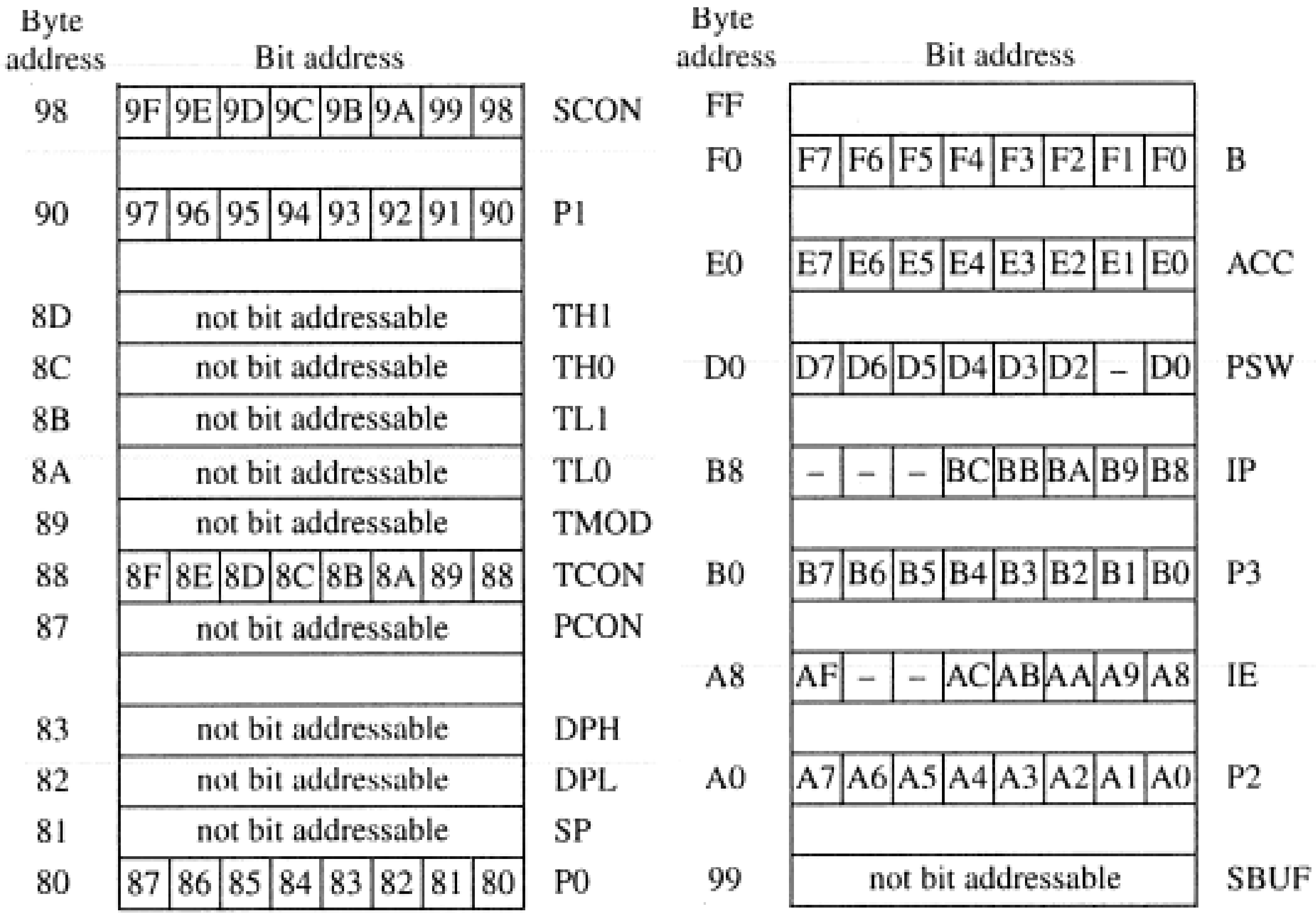
Bit-addressable locations

Bit Addressable RAM

Figure 2-6

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

SFR MEMORY MAP

8 Bytes

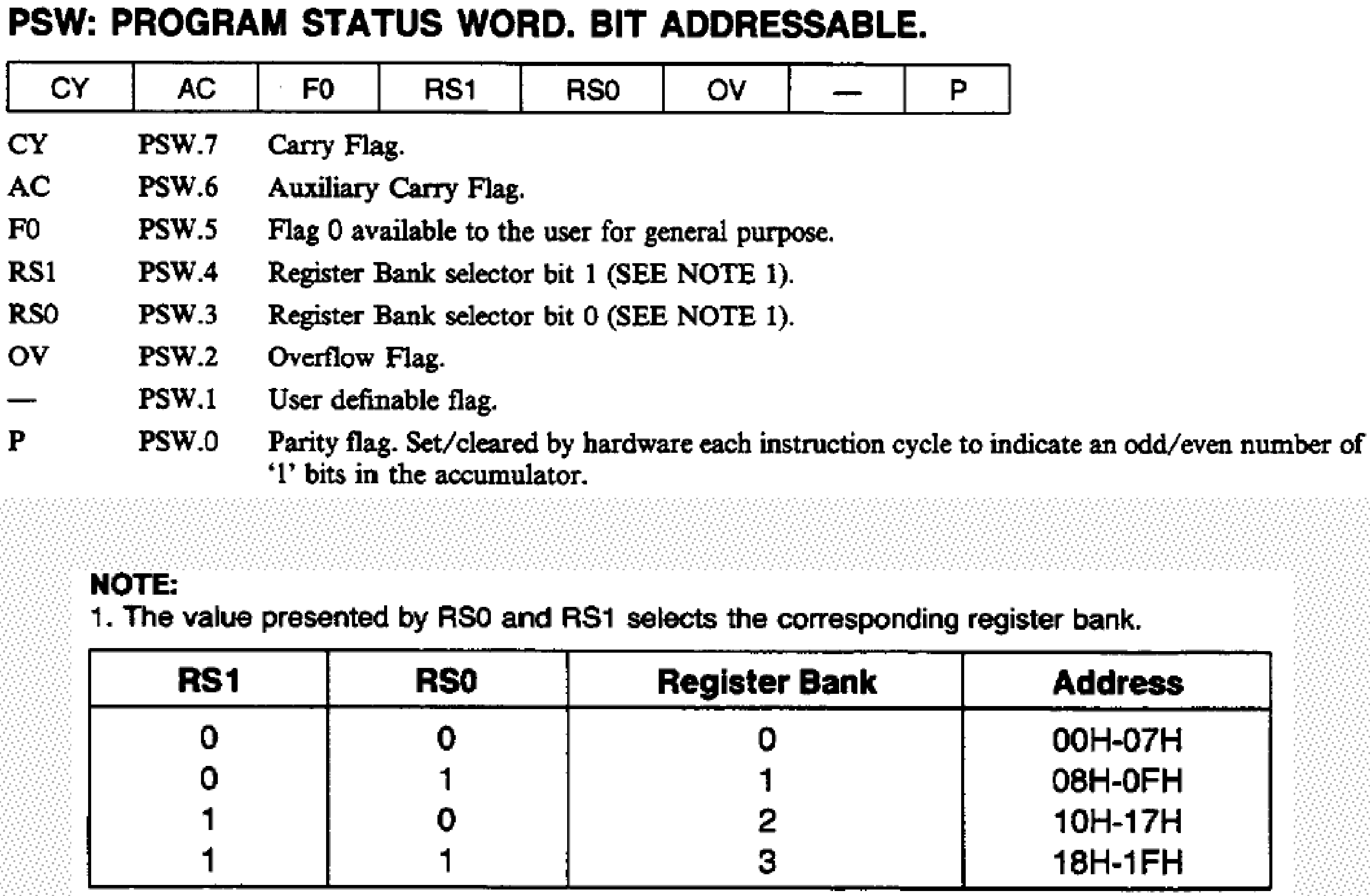
F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

↑
Bit
Addressable

Figure 5

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

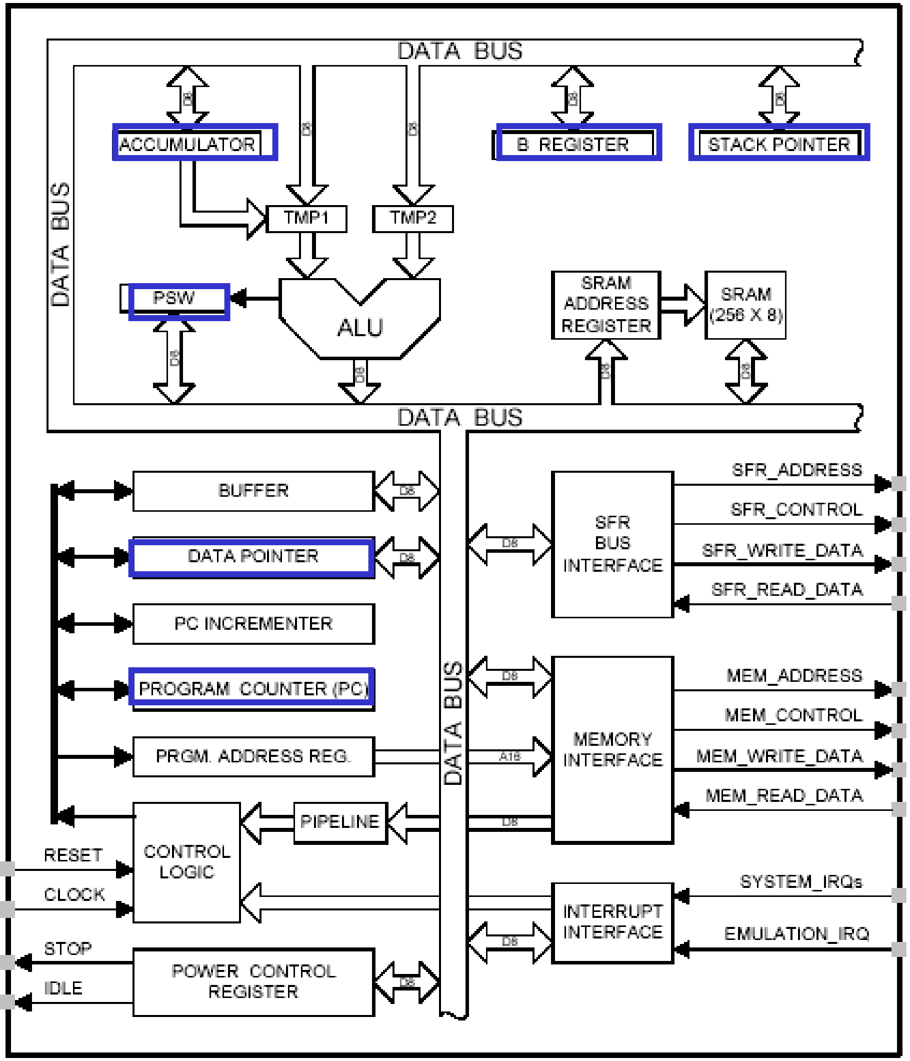
CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared 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

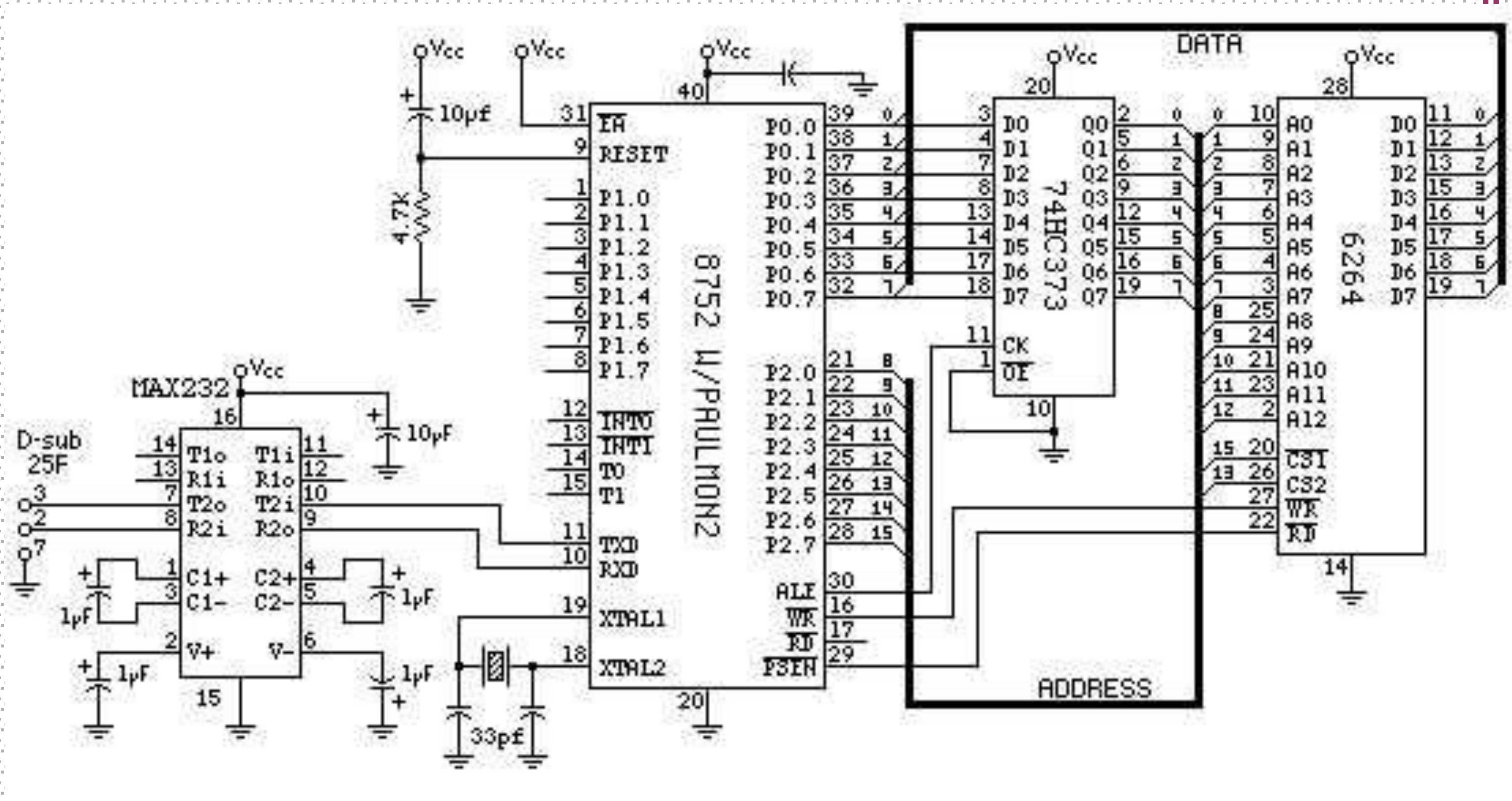
8051 CPU Registers



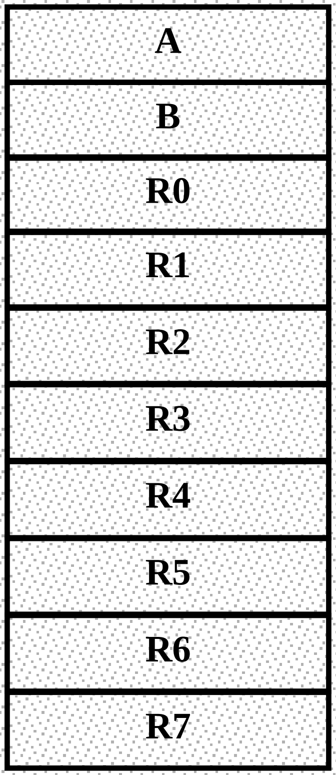
- ❑A (Accumulator)
- ❑B
- ❑PSW (Program Status Word)
- ❑SP (Stack Pointer)
- ❑PC (Program Counter)
- ❑DPTR (Data Pointer)

Used in assembler instructions

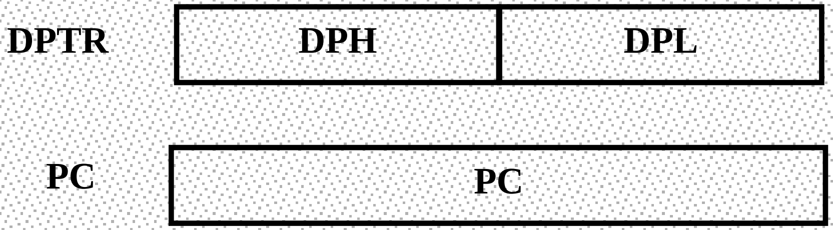
Registers


Vcc
10µf
4.7K
31 EA
9 RESET
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
INT0
INT1
T0
T1
TXD
RXD
XTAL1
XTAL2
33pf
40
20
8752 W/PAULMON2
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
ALE
WR
RD
PSEN
DATA
ADDRESS
74HC373
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
CK
OE
6264
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
CS1
CS2
WR
RD
D0
D1
D2
D3
D4
D5
D6
D7
MAX232
D-sub 25F
T1o
R1i
T2o
R2i
T1i
R1o
T2i
R2o
C1+
C1-
C2+
C2-
V+
V-
1µF
10µF
15
16

Registers



Some 8-bit Registers
of the 8051



Some 8051 16-bit Register

The 8051 Assembly Language

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
                     ;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

Immediate Mode - continue

```
         MOV DPTR,#7521h
         MOV DPL,#21H
         MOV DPH, #75
```

```
         COUNT EGU 30
          ~
          ~
         mov R4, #COUNT
```

```
         MOV DPTR,#MYDATA
         ~
         ~
         0RG 200H
MYDATA:DB "IRAN"
```

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
```

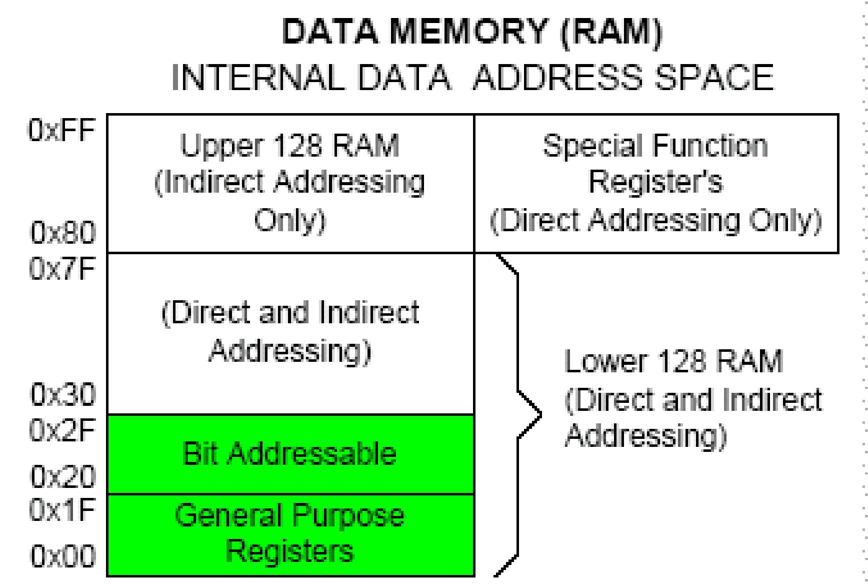
Note that MOV R4,R7 is incorrect

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


DATA MEMORY (RAM)
INTERNAL DATA ADDRESS SPACE
0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

Addressing Modes

Direct Mode – play with R0-R7 by direct address

MOV A,4 ≡ MOV A,R4

MOV A,7 ≡ MOV A,R7

MOV 7,2 ≡ ~~MOV R7,R6~~

```
MOV R2,#5    ;Put 5 in R2
MOV R2,5     ;Put content of RAM at 5 in R2
```

Addressing Modes

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

Uses registers R0 or R1 for 8-bit address:

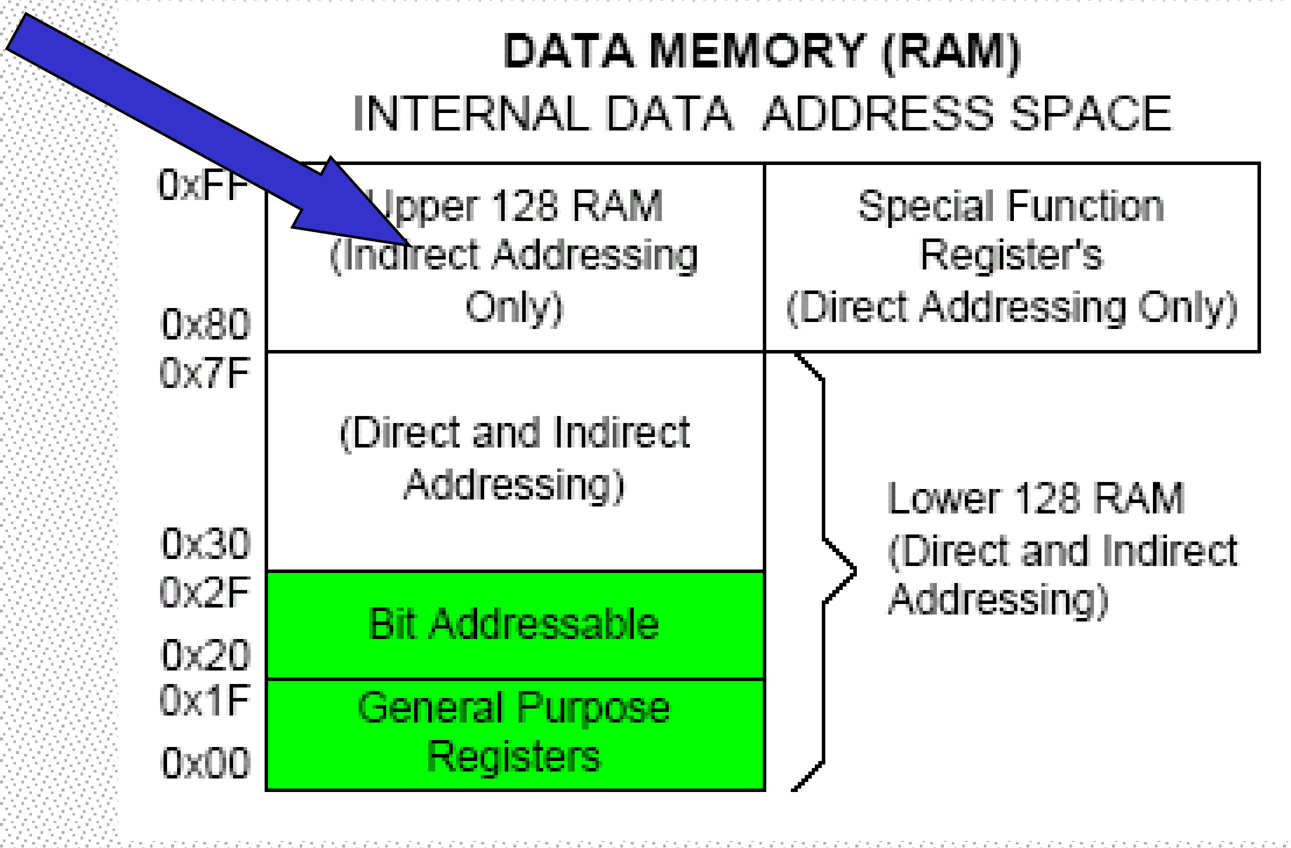
```
mov psw, #0              ; use register bank 0
mov r0, #0x3C
mov @r0, #3              ; memory at 3C gets #3
                         ; M[3C] ← 3
```

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

Use Register Indirect to access upper RAM block (+8052)


DATA MEMORY (RAM)
INTERNAL DATA ADDRESS SPACE
0xFF
0x80
0x7F
0x30
0x2F
0x20
0x1F
0x00
Upper 128 RAM (Indirect Addressing Only)
Special Function Register's (Direct Addressing Only)
(Direct and Indirect Addressing)
Bit Addressable
General Purpose Registers
Lower 128 RAM (Direct and Indirect Addressing)

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

Op code	Immediate data

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

❑ Direct addressing

Op code	Direct address

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

8051 Instruction Format

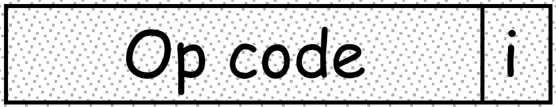
- Register addressing

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

- ❑ Register indirect addressing



mov a, @Ri ; i = 0 or 1

```
070D E7         mov   a,@r1
070D 93         movc  a,@a+dptr
070E 83         movc  a,@a+pc
070F E0         movx  a,@dptr
0710 F0         movx  @dptr,a
0711 F2         movx  @r0,a
0712 E3         movx  a,@r1
```

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

~~07FEh~~

```
0700                1              org 0700h
0700 E106           2              ajmp next    ;next=706h
0702  00            3              nop
0703  00            4              nop
0704  00            5              nop
0705  00            6              nop
                    7        next:
                    8              end
```

8051 Instruction Format

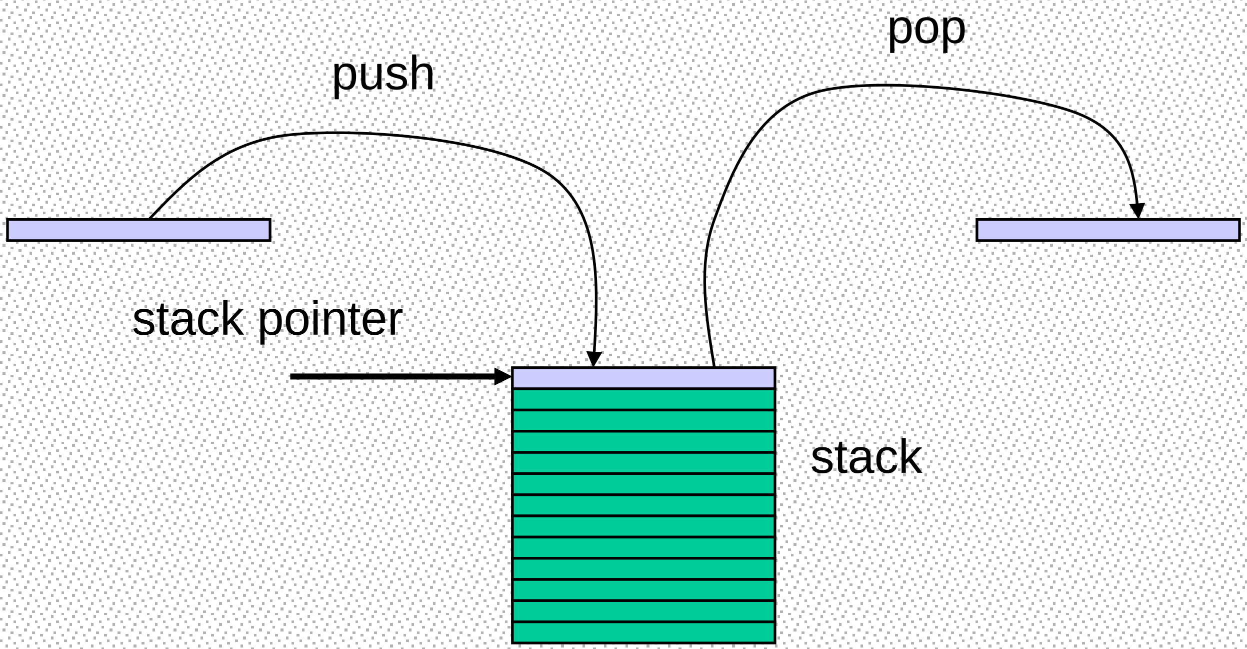
- Long distance address

Op code	A15-A8	A7-A0

Range = (0000h ~ FFFFh)

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

Stacks



Go do the stack exercise.....

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)

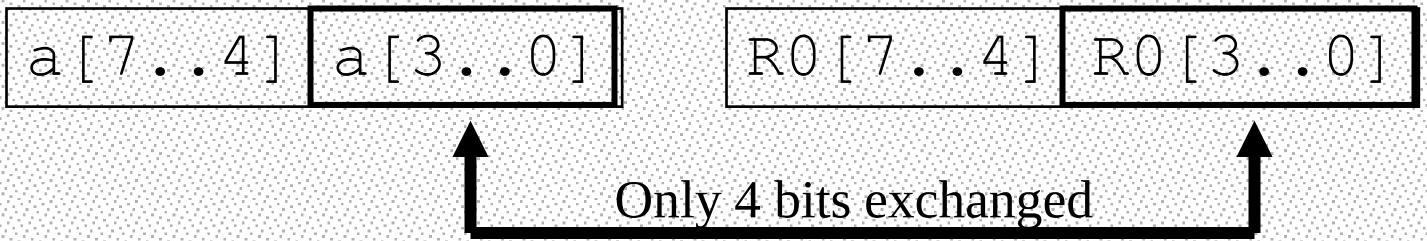
- Therefore

```
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”
```



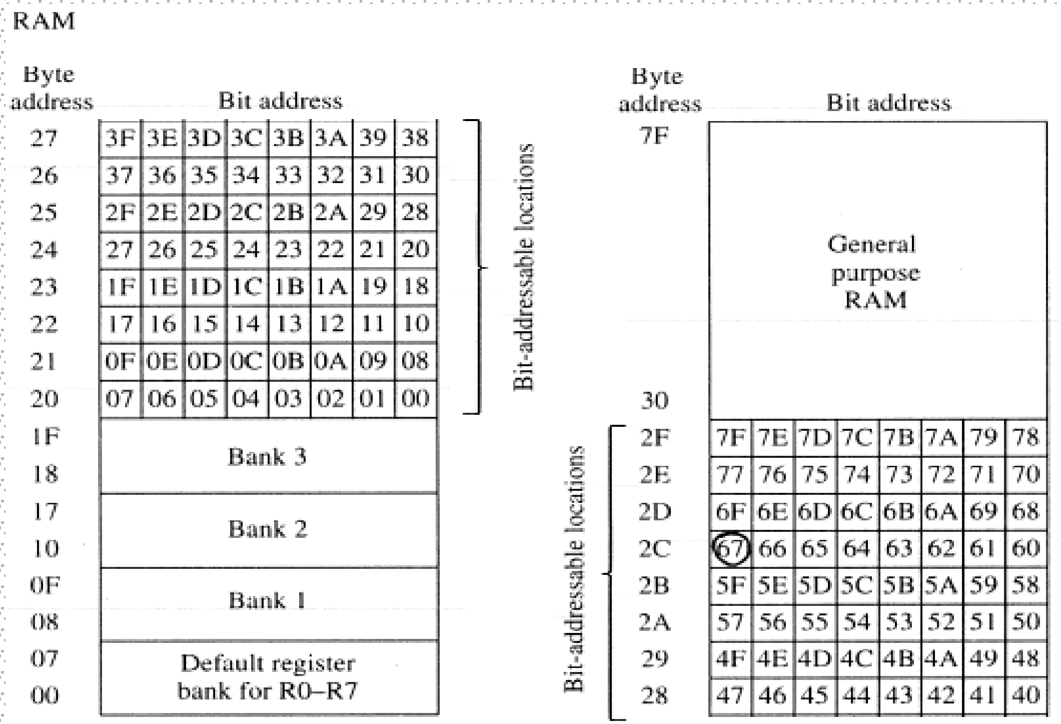
Bit-Oriented Data Transfer

- ❑ 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
```



RAM

Byte address	Bit address							
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	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Default register bank for R0–R7							
00								

Bit-addressable locations

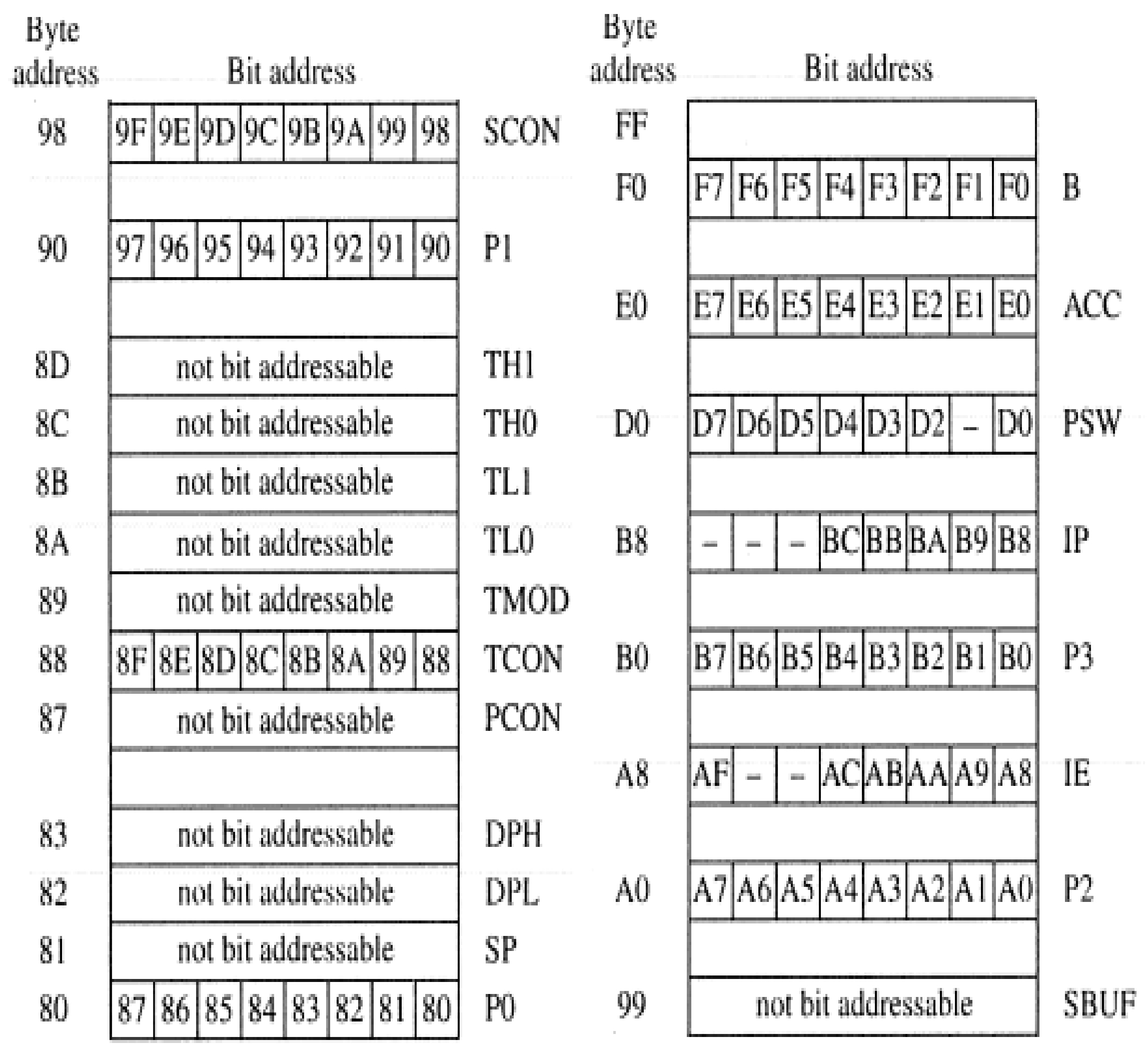
Byte address	Bit address							
7F	General purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40

Bit-addressable locations

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								
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

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

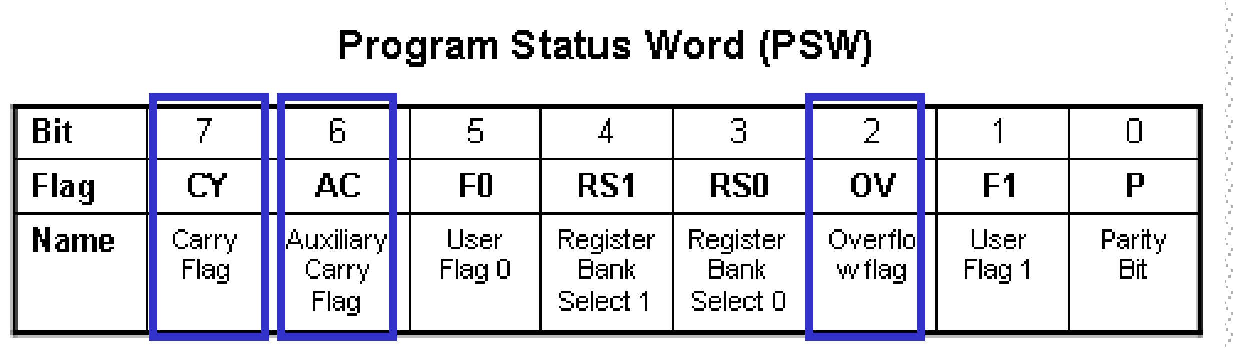
```
add a, byte          ; a ← a + byte
addc a, byte               ; a ← a + byte + C
```

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

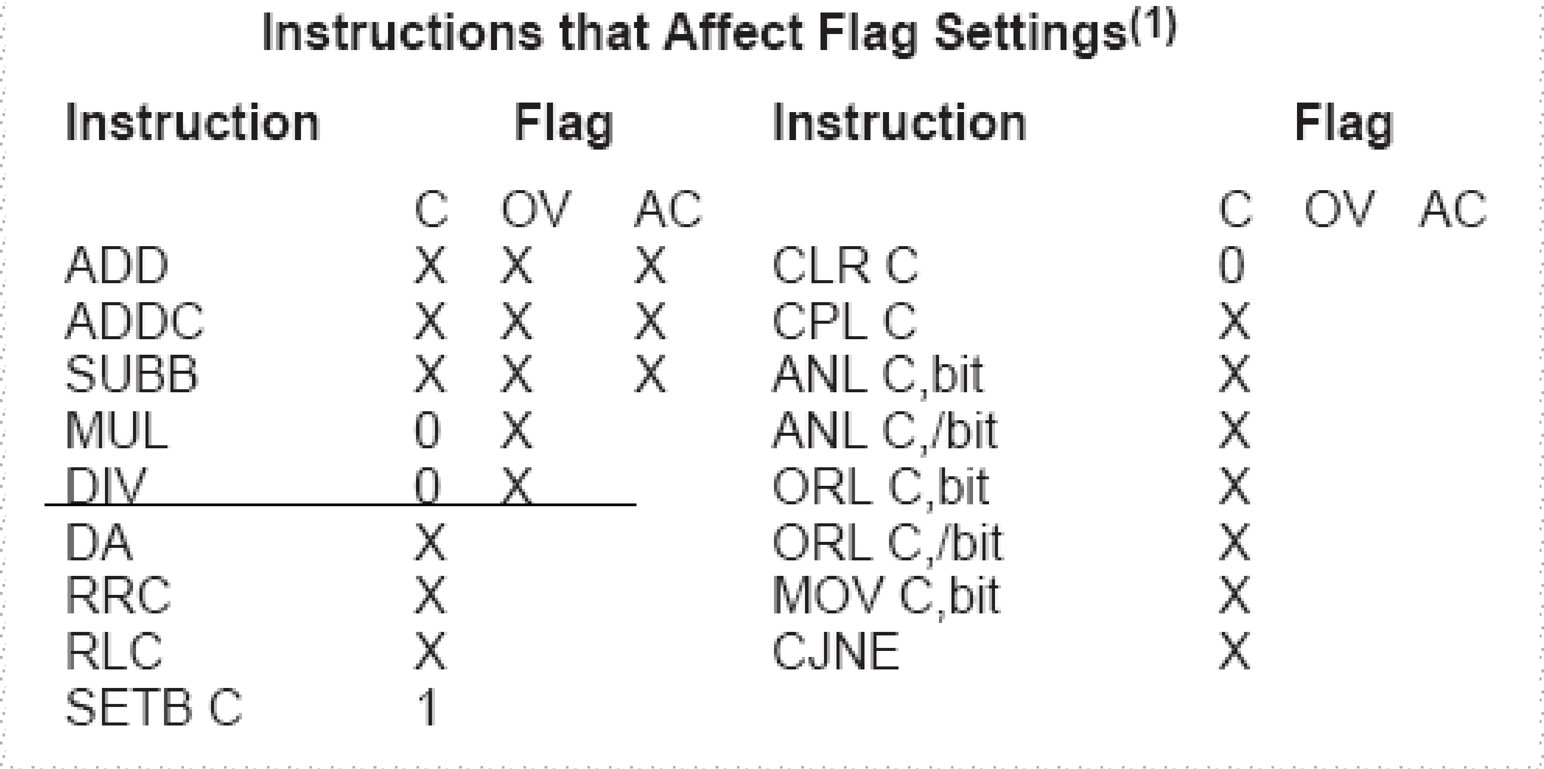
OV = 1 if there is a carry out of bit 7, but not from bit 6, or visa versa.



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	Overflo w 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						

ADD Examples

```
mov a, #3Fh
add a, #D3h
```

- What is the value of the C, AC, OV flags after the second instruction is executed?

```
0011 1111
1101 0011
---------
0001 0010
```

```
C  = 1
AC = 1
OV = 0
```

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           equu        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

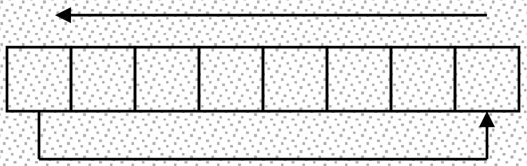
CLR (Set all bits to 0)

CLR A	
CLR byte	(direct mode)
CLR Ri	(register mode)
CLR @Ri	(register indirect mode)

Rotate

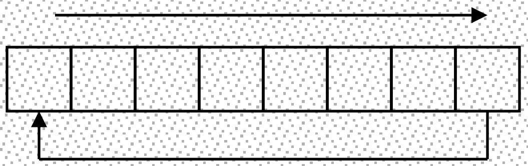
- ❑ Rotate instructions operate only on **a**

RL a



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

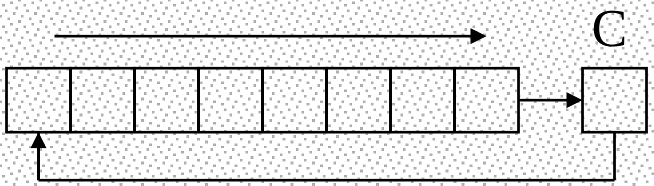
RR a



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

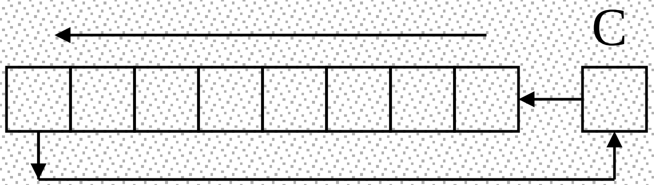
Rotate through Carry

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
```

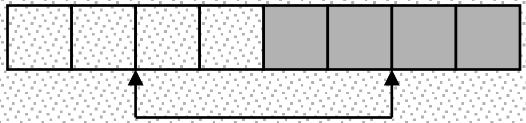
Rotate and Multiplication/Division

- 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 ← 27h
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.

Shift/Mutliply Example

- Program segment to multiply by 2 and add 1.

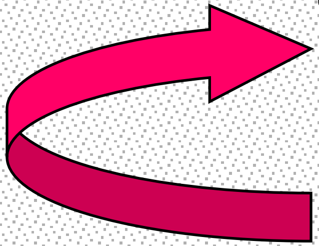
Program Flow Control

- ❑ 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
```

Iterative Loops(examples)

```
      mov a,#50h
      mov b,#00h
      cjne a,#50h,next
      mov  b,#01h
next: nop
      end
```

```
       mov a,#25h
       mov r0,#10h
       mov r2,#5
Again:  mov @ro,a
       inc r0
       djnz r2,again
       end
```

```
      mov a,#0aah
      mov b,#10h
Back1:mov r6,#50
Back2:cpl a
      djnz r6,back2
      djnz b,back1
      end
```

```
       mov a,#0h
       mov r4,#12h
Back:  add a,#05
       djnz r4,back
       mov  r5,a
       end
```

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

call to the subroutine

```
Main:       ...
            acall sublabel
            ...
            ...
sublabel:   ...
            ...
            ret
```

the subroutine

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

Subroutine - another example

```
; Program to compute square root of value on Port 3
; (bits 3-0) and output on Port 1.
        org 0
        ljmp Main                                    }  reset service
Main:   mov P3, #0xFF    ; Port 3 is an input
loop:   mov a, P3
        anl a, #0x0F     ; Clear bits 7..4 of A
        lcall sqrt                                   }  main program
        mov P1, a
        sjmp loop

sqrt:   inc a
        movc a, @a + PC                              }  subroutine
        ret

Sqrs:  db  0,1,1,1,2,2,2,2,2,3,3,3,3,3,3,3           }  data
       end
```

Why Subroutines?

- ❑ Subroutines allow us to have "structured" assembly language programs.
- ❑ This is useful for breaking a large design into manageable parts.
- ❑ It saves code space when subroutines can be called many times in the same program.

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

Long delay Example

```
GREEN_LED:      equ   P1.6
                org   ooh
                ljmp Main                     reset service

                org   100h
Main:           clr    GREEN_LED
Again:          acall Delay                   main program
                cpl    GREEN_LED
                sjmp   Again


Delay:          mov    R7, #02
Loop1:          mov    R6, #00h
Loop0:          mov    R5, #00h
                djnz   R5, $                  subroutine
                djnz   R6, Loop0
                djnz   R7, Loop1
                ret
                END
```

Example

```
; Move string from code memory to RAM
            org 0
            mov dptr,#string
            mov r0,#10h
Loop1:      clr a
            movc a,@a+dptr
            jz stop
            mov @r0,a
            inc dptr
            inc r0
            sjmp loop1
Stop:       sjmp stop

; on-chip code memory used for string
            org 18h
String:     db ‘this is a string’,0
            end
```

Example

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

```
          setb p1.2
          mov a,#45h       ;data
Again:    jnb p1.2,again ;wait for data
  request
          mov p0,a         ;enable strobe
          setb p2.3
          clr p2.3
```

Example

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

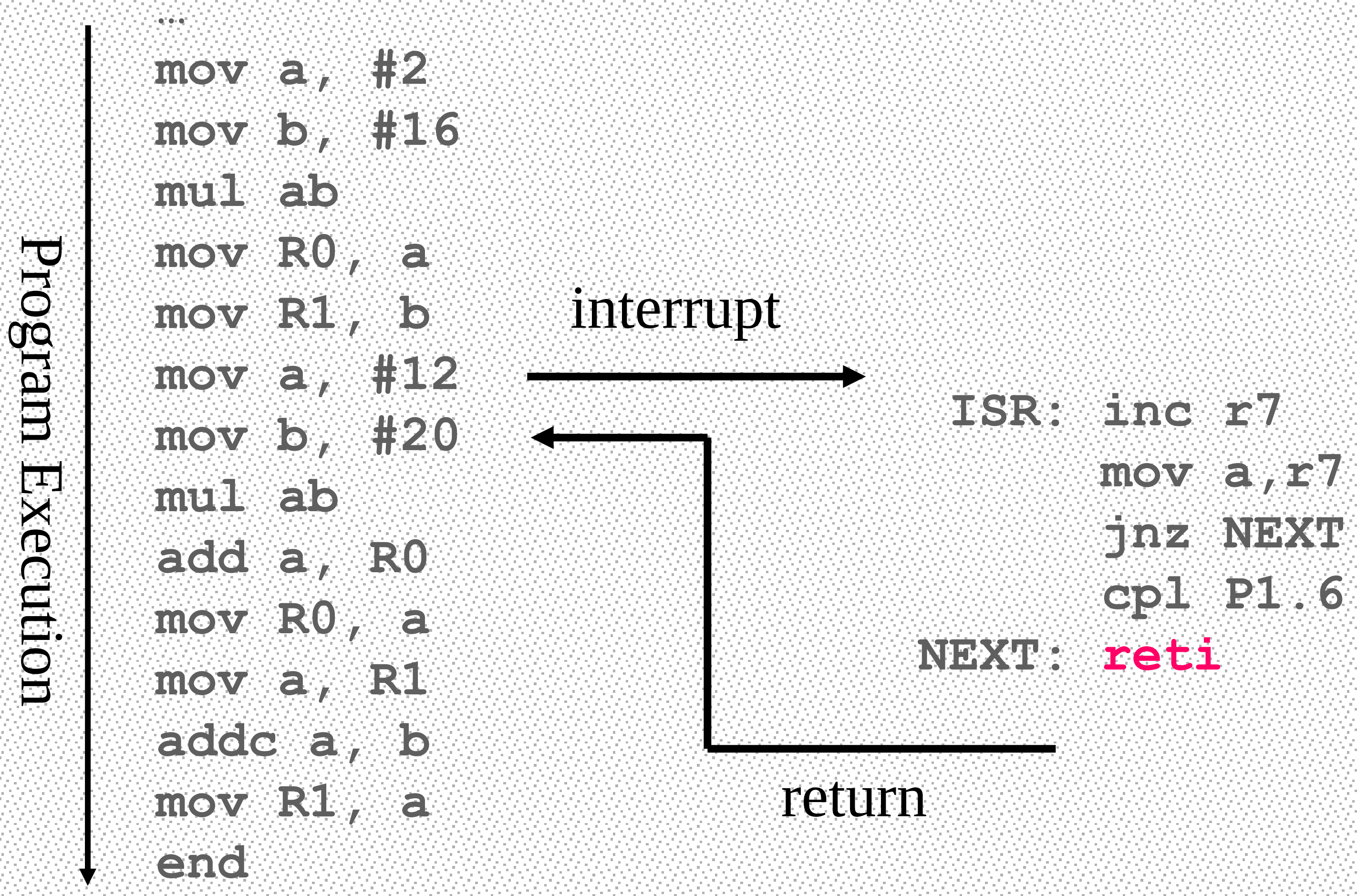
```
 back:      setb p1.2
            acall delay
            Clr p1.2
            acall delay
            sjmp back
```

Example

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

8051 timer

Interrupts



```
…
mov a, #2
mov b, #16
mul ab
mov R0, a
mov R1, b
mov a, #12
mov b, #20
mul ab
add a, R0
mov R0, a
mov a, R1
addc a, b
mov R1, a
end
```

```
ISR:  inc r7
      mov a,r7
      jnz NEXT
      cpl P1.6
NEXT: reti
```

Interrupt Sources

- Original 8051 has 5 sources of interrupts
  - Timer 0 overflow
  - Timer 1 overflow
  - External Interrupt 0
  - External Interrupt 1
  - Serial Port events (buffer full, buffer empty, etc)
- Enhanced version has 22 sources
  - More timers, programmable counter array, ADC, more external interrupts, another serial port (UART)

Interrupt Process

If interrupt event occurs AND interrupt flag for that event is enabled, AND interrupts are enabled, then:

1. Current PC is pushed on stack.
2. Program execution continues at the interrupt vector address for that interrupt.
3. When a RETI instruction is encountered, the PC is popped from the stack and program execution resumes where it left off.

Interrupt Priorities

- What if two interrupt sources interrupt at the same time?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a default priority order.
- Priority can also be set to “high” or “low”.

Interrupt SFRs

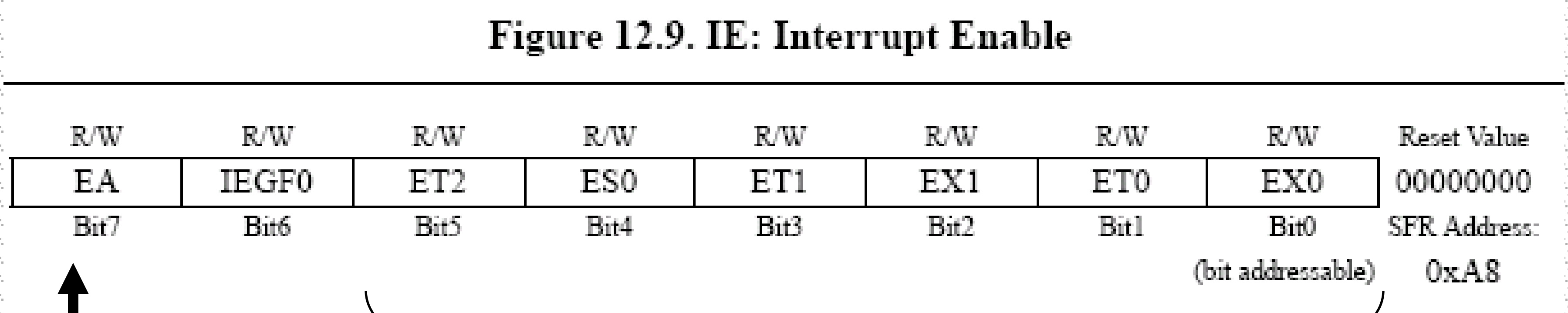


Figure 12.9. IE: Interrupt Enable

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
EA	IEGF0	ET2	ES0	ET1	EX1	ET0	EX0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xA8

Interrupt enables for the 5 original 8051 interrupts:

Timer 2

Serial (UART0)

Timer 1

External 1

Timer 0

External 0

Global Interrupt Enable – must be set to 1 for any interrupt to be enabled

1 = Enable
0 = Disable

Interrupt Vectors

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

External Interrupt 0:	0003h
Timer 0 overflow:	000Bh
External Interrupt 1:	0013h
Timer 1 overflow:	001Bh
Serial :	0023h
Timer 2 overflow(8052+)	002bh

Note: that there are only 8 memory locations between vectors.

Interrupt Vectors

To avoid overlapping Interrupt Service routines, it is common to put JUMP instructions at the vector address. This is similar to the reset vector.

```
        org  009B          ; at EX7 vector
        ljmp EX7ISR
        cseg at 0x100      ; at Main program
Main:  ...                 ; Main program
        ...
EX7ISR:...                 ; Interrupt service routine
        ...                ; Can go after main program
        reti               ; and subroutines.
```

Example Interrupt Service Routine

```
;EX7 ISR to blink the LED 5 times.
;Modifies R0, R5-R7, bank 3.
;----------------------------------------------------
   ISRBLK:    push PSW              ;save state of status word
              mov PSW,#18h          ;select register bank 3
              mov R0, #10           ;initialize counter
    Loop2:    mov R7, #02h          ;delay a while
    Loop1:    mov R6, #00h
    Loop0:    mov R5, #00h
              djnz R5, $
              djnz R6, Loop0
              djnz R7, Loop1
              cpl P1.6              ;complement LED value
              djnz R0, Loop2        ;go on then off 10 times
              pop PSW
              reti
```