Assembly Language and Addressing Mode of the 8051 Microcontrollers

Instructor

Prof. Zhizheng Wu

吴智政

School of Mechatronic Engineering and Automation

OUTLINE

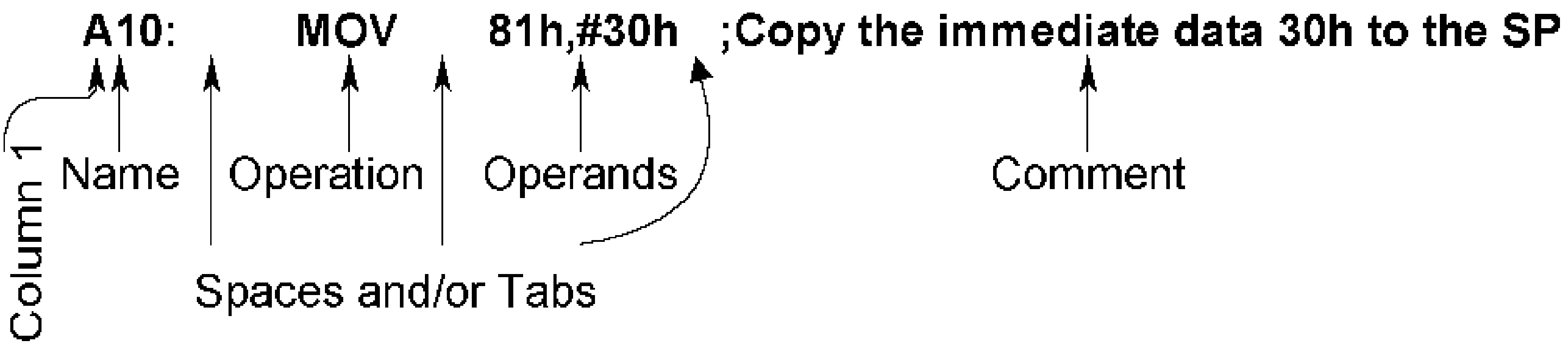
- Introduction to Assembly Language
- Address the 8051 microcontroller on-chip memory
- Address the 8051 microcontroller external memory

☞ Structure of assembly language

- In the early days of the computer, programmers coded in machine language, consisting of 0s and 1s
  - Tedious, slow and prone to error
- Assembly languages, which provided mnemonics for the machine code instructions, plus other features, were developed
  - An Assembly language program consist of a series of lines of Assembly language instructions
  - Assembly language is referred to as a low level language. It deals directly with the internal structure of the CPU
- High level language, i.e. C++, C, Java, …

STRUCTURE OF ASSEMBLY LANGUAGE

- Assembly Language Syntax
  - Syntax = format/rule
  - **[label:]   mnemonic   [operands]   [;comment]**
  - Items in square brackets are optional



- Mnemonic/Operation Code (Opcode)
  - Program = a set of instructions
  - All computers work according to the program
  - All instructions contain a “verb” called mnemonic/operation code, which tells the computer what to do
  - e.g. MOV R0, #12h — “**MOV**” is the opcode

STRUCTURE OF ASSEMBLY LANGUAGE

- Operand
  - Apart from Opcode, an instruction also includes object to act on.
  - The object is called an operand.
  - Operand is optional. Instructions can have one, two or no operands.
  - e.g. MOV **R0, #12h** --- R0 and #12h are two operands

    INC **A** --- A is the only one operand

    NOP --- no operand follows

STRUCTURE OF ASSEMBLY LANGUAGE

- Binary nature of machine instruction
  - Unlike human, computers do not know verbal instructions; they only know **0**s and **1**s
  - Binary data: program should be in a stream of 0s and 1s
  - It is also called ***"Machine Language"***
  - For convenient purpose, machine instructions are usually expressed in hexadecimal (i.e. base-16) format, called machine codes.

    e.g. Mnemonic : ADD A, #10H

    Equivalent Machine codes: 24h 10h (hexadecimal)

- How 8051 Interprets Binary Data
  - Machine instructions can be 3 bytes (24 bits), 2 bytes (16 bits) or 1 byte (8 bits) long
  - The 1st byte (8 bits) is the operation code (opcode)
  - The remaining byte(s) is/are the supplement data for the operation code
  - 1-byte instruction: Contain the opcode only. Actions do not need supplement data.

e.g.

Mnemonic	Equivalent Machine codes	
NOP	00h	(hexadecimal)
ADD A, R0	28h	
INC A	04h	

- How 8051 Interprets Binary Data

  - 2-byte instruction: The 1st byte is the opcode. The 2nd byte may be either an immediate data (a number) or the low-order byte of an address

e.g.

Mnemonic	Equivalent Machine codes
ADD A, #30h	24h 30h
ADD A, 30h	25h 30h

  - 3-byte instruction: The 1st byte is the opcode. The 2nd and the 3rd byte are the low-order byte and the high-order byte of an 16-bit memory address

e.g.

Mnemonic	Equivalent Machine codes
LJMP #0130h	02h 30h 01h

- Pseudo-instructions/Directives

  Beside mnemonics, directives are used to define variables and memory locations where the machine codes are stored. These directives are interpreted by assembler during the conversion of the assembly language program into machine codes.

  - ORG (origin)

  Indicates the beginning of the address of the instructions. The number that comes after ORG can be either hex or decimal.

  Eg. ORG 0030H

  - END

  Indicates to the assembler the end of the source assembly instructions.

- Pseudo-instructions/Directives

- EQU (equate)

Used to define a constant without occupying a memory location. It does not set aside storage for a data item but associates a constant value with a data label so that when the label appears in the program. Its constant value will be substituted for the label.

```
COUNT  EQU   25H
```

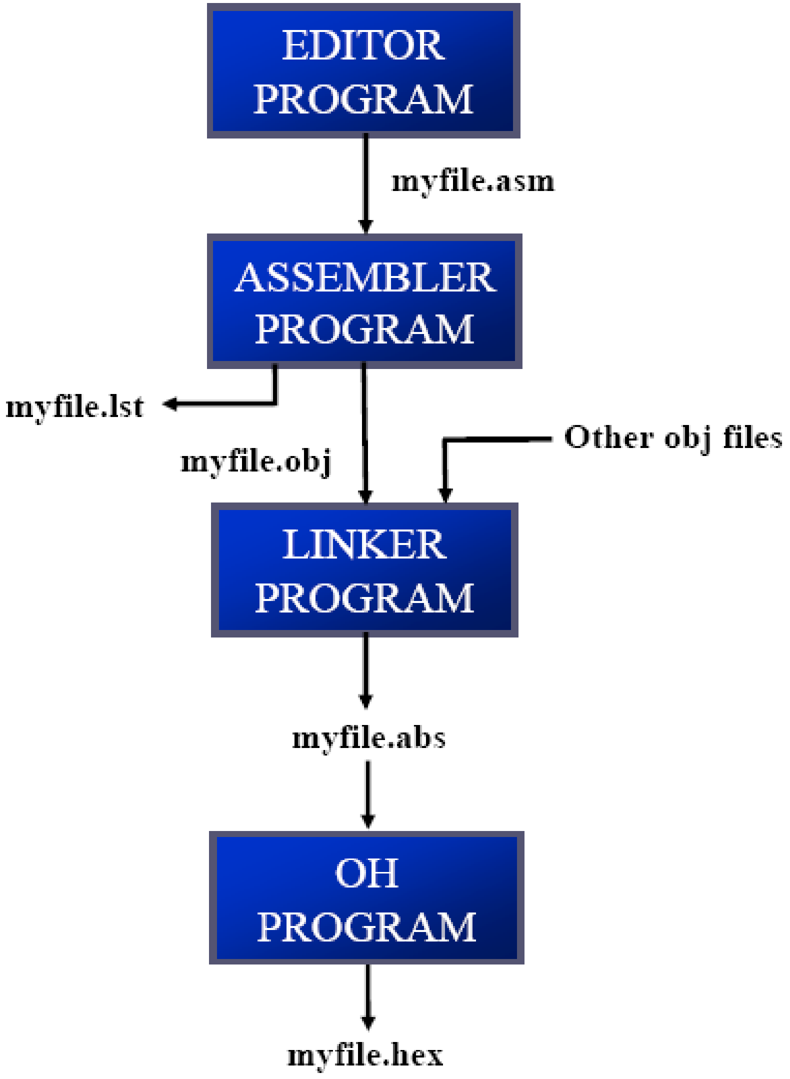
- DB (define byte)

Used to define 8-bit data and store them in assigned memory locations. Define data can be in decimal, binary, hex, or ASCII formats.

```
MYDATA        DB     23H
ASCIICODE:    DB     "APPLE"
```

STRUCTURE OF ASSEMBLY LANGUAGE

- Assembling and Running an 8051 Program



STRUCTURE OF ASSEMBLY LANGUAGE

- Assembly Program runs following the program counter (PC)

  - The program counter points to the address of the next instruction to be executed
    - As the CPU fetches the opcode from the program ROM, the program counter is increasing to point to the next instruction
  - The program counter is 16 bits wide
    - This means that it can access program addresses 0000 to FFFFH, a total of 64K bytes of code

STRUCTURE OF ASSEMBLY LANGUAGE

- All 8051 members start at memory address 0000 when they're powered up
  - Program Counter has the value of 0000
  - The first opcode is burned into ROM address 0000H, since this is where the 8051 looks for the first instruction when it is booted
  - We achieve this by the `ORG` statement in the source program

STRUCTURE OF ASSEMBLY LANGUAGE

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

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

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

STRUCTURE OF ASSEMBLY LANGUAGE

- The program status word (PSW) register, also referred to as the *flag register*, is an 8 bit register
  - Only 6 bits are used
    - These four are CY (*carry*), AC (*auxiliary carry*), P (*parity*), and OV (*overflow*)
      - They are called *conditional flags*, meaning that they indicate some conditions that resulted after an instruction was executed
    - The PSW3 and PSW4 are designed as RS0 and RS1, and are used to change the bank
  - The two unused bits are user-definable

STRUCTURE OF ASSEMBLY LANGUAGE

CY	AC	F0	RS1	RS0	OV	--	P

A carry from D3 to D4

Carry out from the d7 bit

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

Reflect the number of 1s in register A

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

STRUCTURE OF ASSEMBLY LANGUAGE

Instructions that affect flag bits

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

STRUCTURE OF ASSEMBLY LANGUAGE

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

Example 2-2

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

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

Solution:

38	00111000
+ 2F	00101111
67	01100111

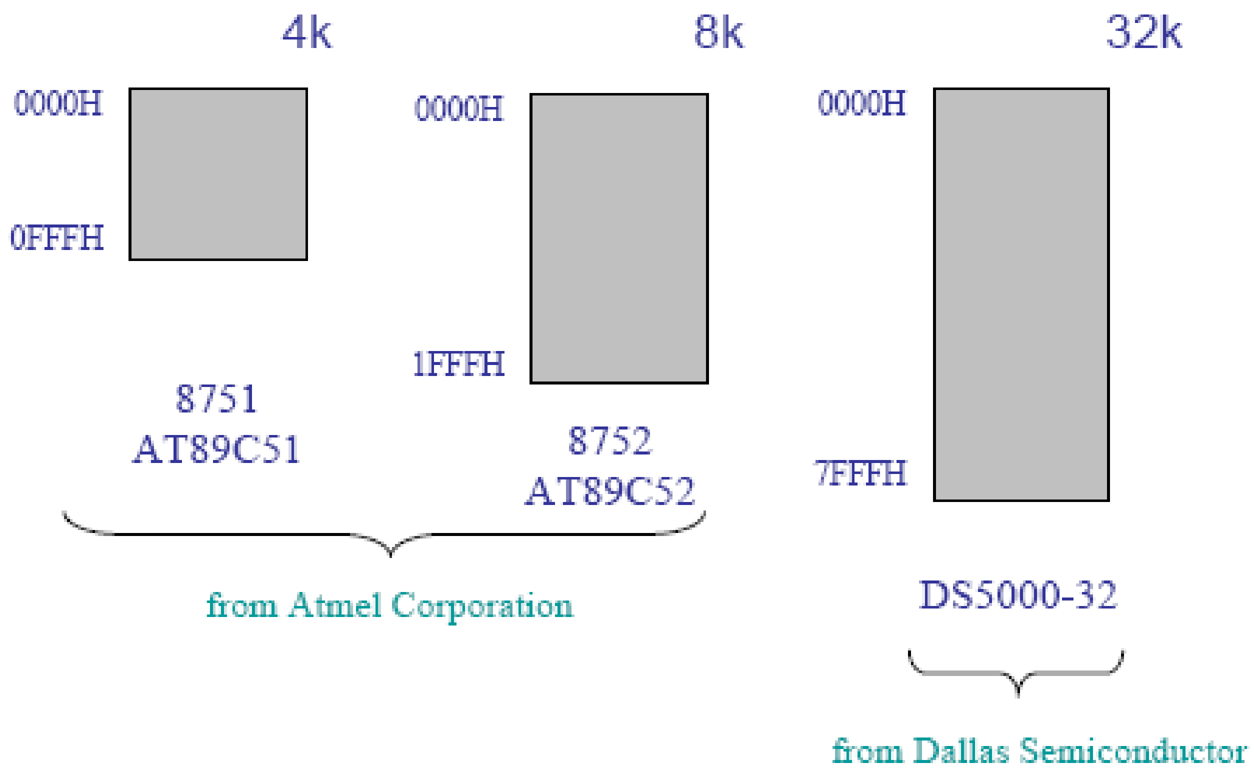
CY = 0 since there is no carry beyond the D7 bit

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

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

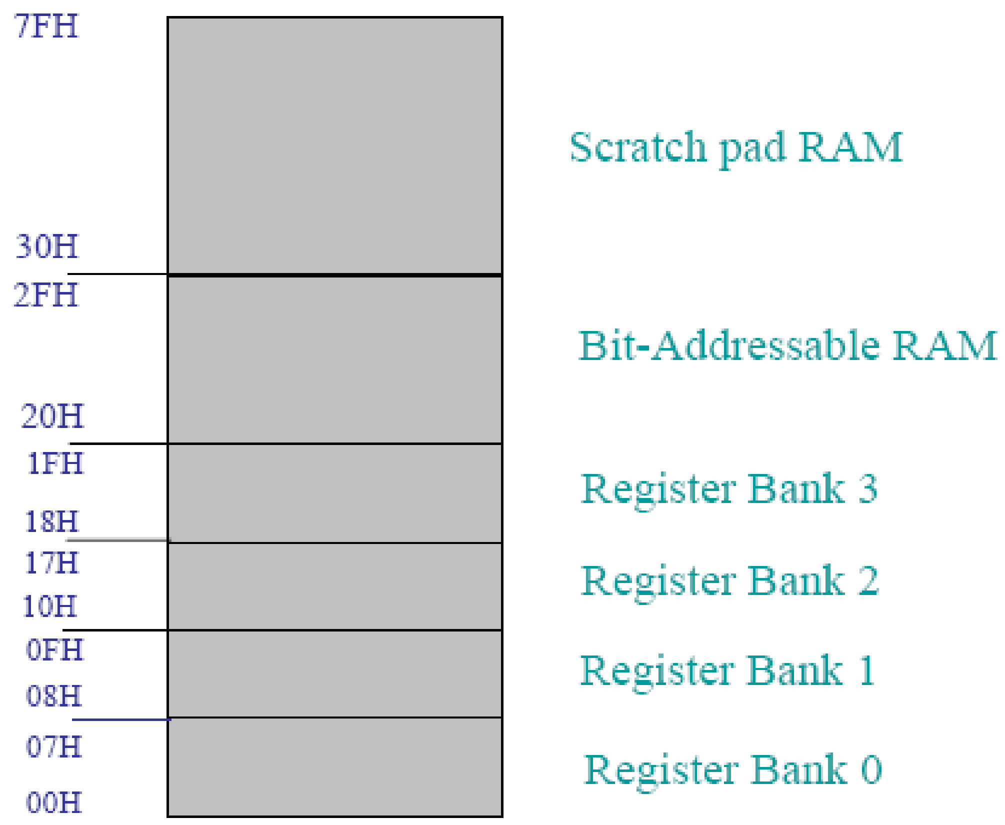
ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Memory mapping in 8051
  - ROM memory map in 8051 family



ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- RAM memory space allocation in the 8051



ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Program Counter (PC)
  - ***PC*** is a 16-bit register
  - ***PC*** is the only register that does not have an internal address
  - Holds the address of the memory location to fetch the program instruction
  - Program ROM may be on the chip at addresses 0000H to 0FFFH (4Kbytes), external to the chip for addresses that exceed 0FFFH
  - Program ROM may be totally external for all addresses from 0000H to FFFFH
  - ***PC*** is automatically incremented (+1) after every instruction byte is fetched

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Data Pointer (DPTR)
  - ***DPTR*** is a 16-bit register
  - ***DPTR*** is made up of two 8-bit registers: ***DPH*** and ***DPL***
  - ***DPTR*** holds the memory addresses for internal and external code access and external data access

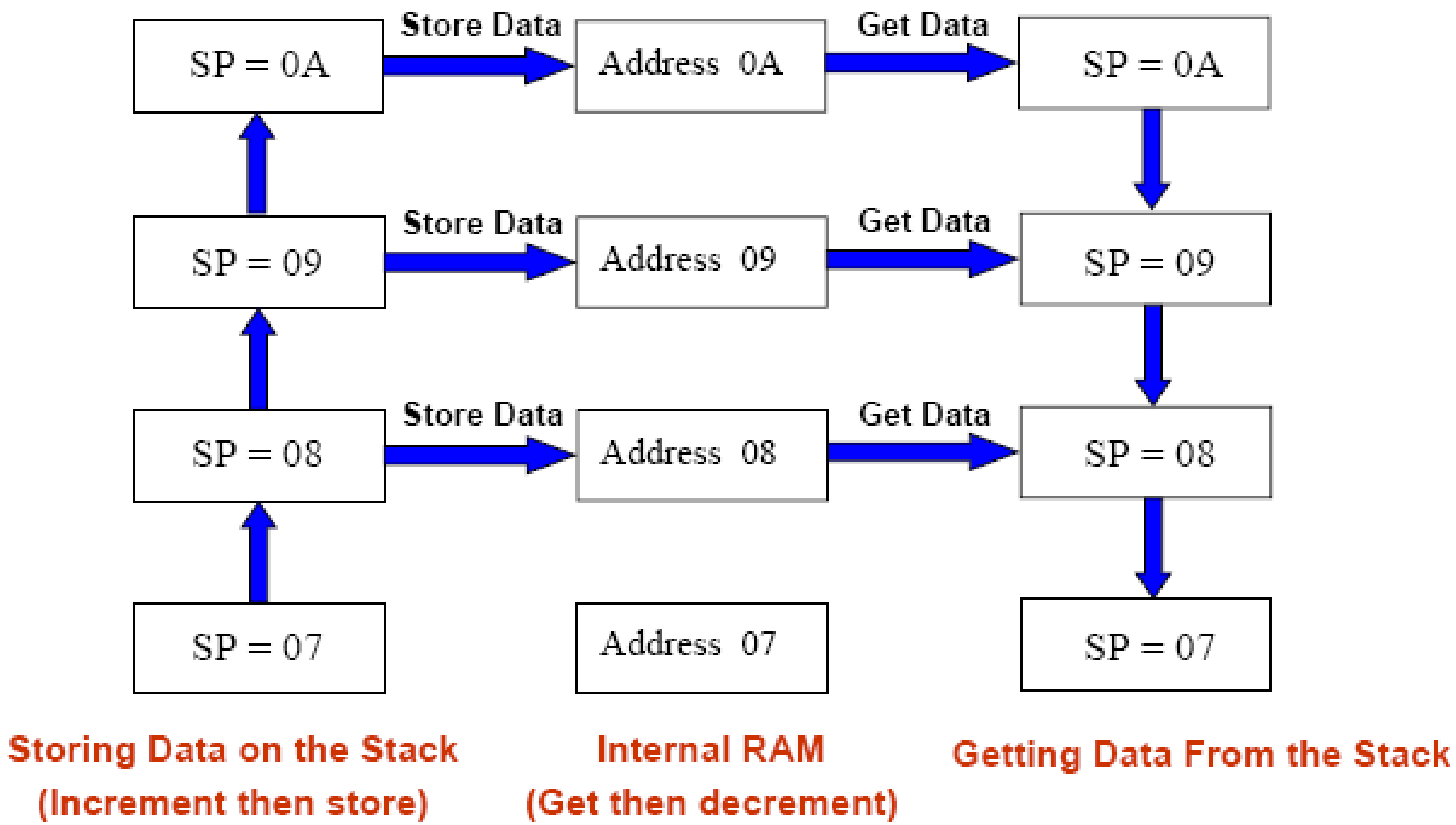
    (eg. MOVC A, @A+DPTR; MOVX A, @DPTR; MOVX @DPTR, A)
  - ***DPTR*** is under the control of program instructions and can be specified by its 16-bit name, or by each individual byte name, ***DPH*** and ***DPL***
  - ***DPTR*** does not have a single internal address; ***DPH*** and ***DPL*** are each assigned an address (83H and 82H)

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Stack and Stack Pointer (SP)
  - ***SP*** is a 8-bit register used to hold an internal RAM address that is called the "***top of the stack***"
  - ***Stack*** refers to an area of internal RAM that is used in conjunction with certain opcodes to store and retrieve data quickly
  - ***SP*** holds the internal RAM address where the last byte of data was stored by a stack operation
  - When data is to be placed on the stack, the ***SP*** increments before storing data on the stack so that the stack ***grows up*** as data is stored
  - As data is retrieved from the stack, the byte is read from the stack, and then the ***SP*** decrements to point to the next available byte of stored data
  - ***SP*** = 07H after reset

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

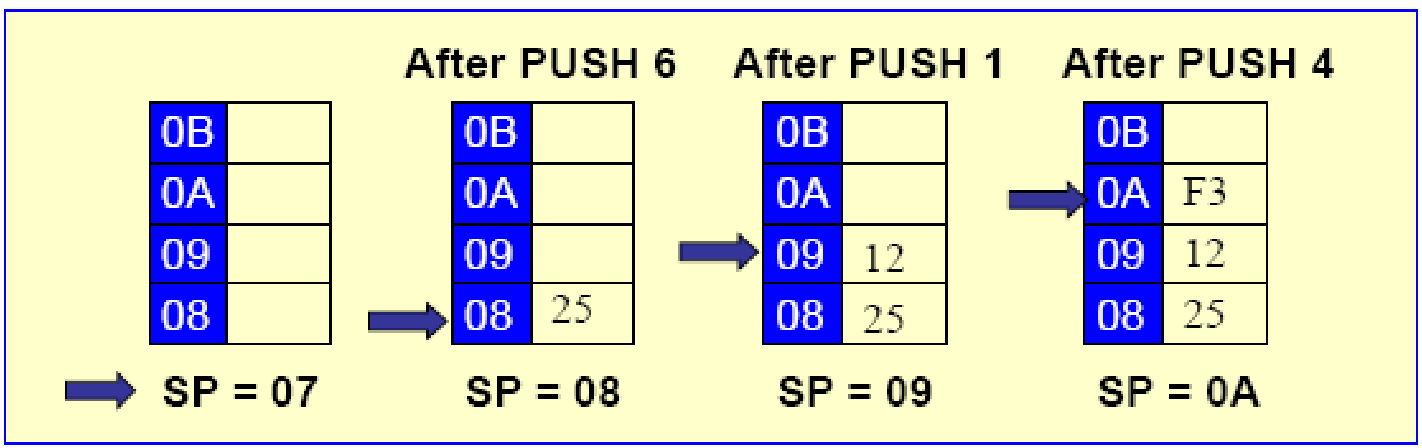
- Stack Operation



ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Example 1
  - Show the stack and stack pointer for the following code. Assume the default stack area.

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



ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Example 2
  - Examine the stack, show the contents of the registers and ***SP*** after execution of the following instruction. All values are in hex.

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

Start SP = 0B

0B	54
0A	F9
09	76
08	6C

05	??
04	??
03	??
02	??

After POP 3 — **SP = 0A**

0B	54
0A	F9
09	76
08	6C

05	??
04	??
03	54
02	??

After POP 5 — **SP = 09**

0B	54
0A	F9
09	76
08	6C

05	F9
04	??
03	54
02	??

After POP 2 — **SP = 08**

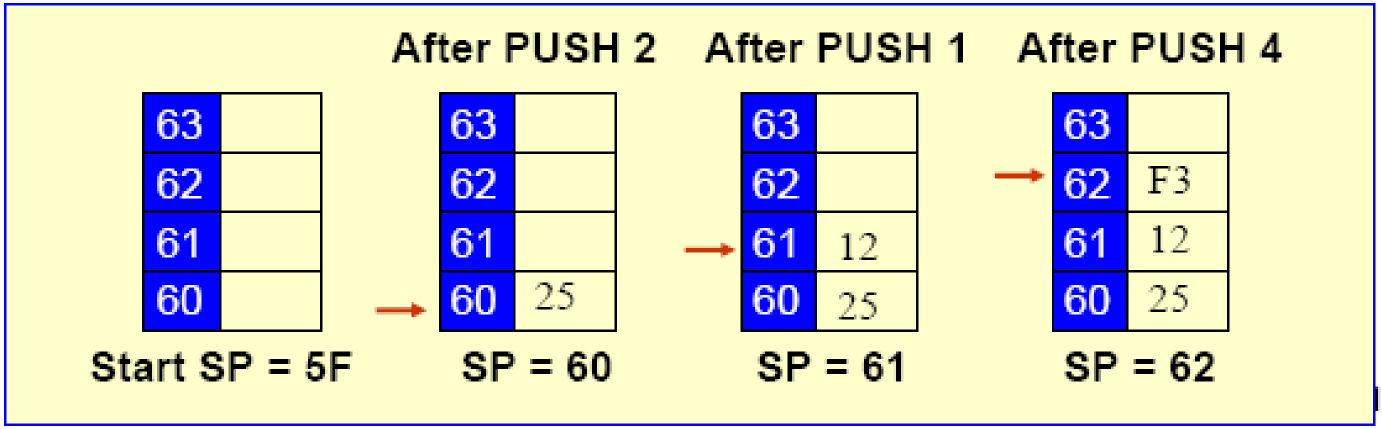
0B	54
0A	F9
09	76
08	6C

05	F9
04	??
03	54
02	76

ADDRESS THE 8051 MOCROCONTROLLER
ON-CHIP MEMORY

- Example 3
  - Show the stack and stack pointer for the following.

```
MOV   SP, #5FH
MOV   R2, #25H
MOV   R1, #12H
MOV   R4, #0F3H
PUSH  2
PUSH  1
PUSH  4
```



- Addressing Modes
  - Immediate addressing
  - Register addressing
  - Direct addressing
  - Register Indirect addressing
  - Indexed addressing
  - Relative addressing
  - Absolute addressing
  - Long addressing
  - Bit addressing

- Immediate Addressing Mode

MOV A, #65H

MOV R6, #65H

MOV DPTR, #2343H

MOV P1, #65H

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Register Addressing Mode

```
MOV    Rn,  A     ;  n=0,...,7
ADD    A,  Rn
MOV    DPL,  R6
```

~~MOV DPTR, A~~

~~MOV Rm, Rn~~

◆ The source and destination registers must match in size

◆ The movement of data between Rn registers is not allowed

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Direct Addressing Mode

☞ Although the entire of 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM loc. 30 – 7FH.

```
MOV     R0, 40H
MOV     56H, A
MOV     A, 4        ; ≡ MOV A, R4
MOV     6, 2        ; copy R2 to R6
                    ; MOV  R6, R2 is invalid !
```

☞ Contrast this with immediate addressing mode

✓ There is no "#" sign in the operand

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Some Examples

```
MOV   A,  #72H        ; A=72H
MOV   R4, #62H        ; R4=62H
MOV   B,  0F9H        ; B=the content of F9'th byte of RAM

MOV   DPTR,  #7634H
MOV   DPL,   #34H
MOV   DPH,   #76H

MOV   P1, A           ; mov A to port 1
```

Note 1:

MOV A, #72H ≠ MOV A, 72H

After instruction "MOV A, 72H" the content of 72th byte of RAM will replace in Accumulator.

Note 2:

MOV A, R3 ≡ MOV A, 3

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Direct Addressing Mode

☞ Only direct addressing mode is allowed for pushing or popping the stack

PUSH A is invalid

✓ Pushing the accumulator A onto the stack must be coded as

PUSH 0E0H

☞ Example:

```
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 ?
POP    02      ; pop top of stack into R2; now R2 =
               R6
```

- Register Indirect Addressing Mode
  - In this mode, register is used as a pointer to the data. In other word, the content of register R0 or R1 is sources or target in MOV, ADD and SUBB instructions.

MOV A, @Ri ; move content of RAM location where address is held by Ri into A (i=0 or 1)

MOV @R1, B

- Only register R0 and R1 are used for this purpose

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Indexed Addressing Mode
  - This mode is widely used in accessing data elements of look-up table entries located in the program (code) space ROM at the 8051

MOVC A,@A+DPTR

(A,@A+PC)

A= content of address A +DPTR from ROM

Note:

Because the data elements are stored in the program (code) space ROM of the 8051, it uses the instruction MOVC instead of MOV. The "C" means code.

- Relative, Absolute, & Long Addressing
  - ☞ Used only with jump and call instructions:

SJMP

ACALL

AJMP

LCALL

LJMP

- Relative Addressing
  - Relative address (offset) is an 8-bit signed value (-128 to 127) which is added to the program counter to form the address of the next instruction
  - This detail is no concern to the user since the jump destinations are usually specified as labels and the assembler determines the relative offset
  - Advantage of relative addressing: position independent codes

- Absolute Addressing
  - Absolute addressing is only used with ACALL and AJMP.
  - The 11 least significant bits of the destination address comes from the opcode and the upper five bits are the current upper five bits in the program counter (PC).
  - The destination is in the same 2K (2^{11}) bytes of the source.

ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Long Addressing
  - ☞ Long addressing is used only with the LCALL and LJMP instructions.
  - ☞ These 3-byte instructions include a full 16-bit destination address as bytes 2 and 3.
  - ☞ The full 64K code space is available.
  - ☞ The instruction is long and position dependent.
  - Example:

    LJMP 8AF2H ; Jumps to memory location 8AF2H

    Machine code: 02 F2 8A

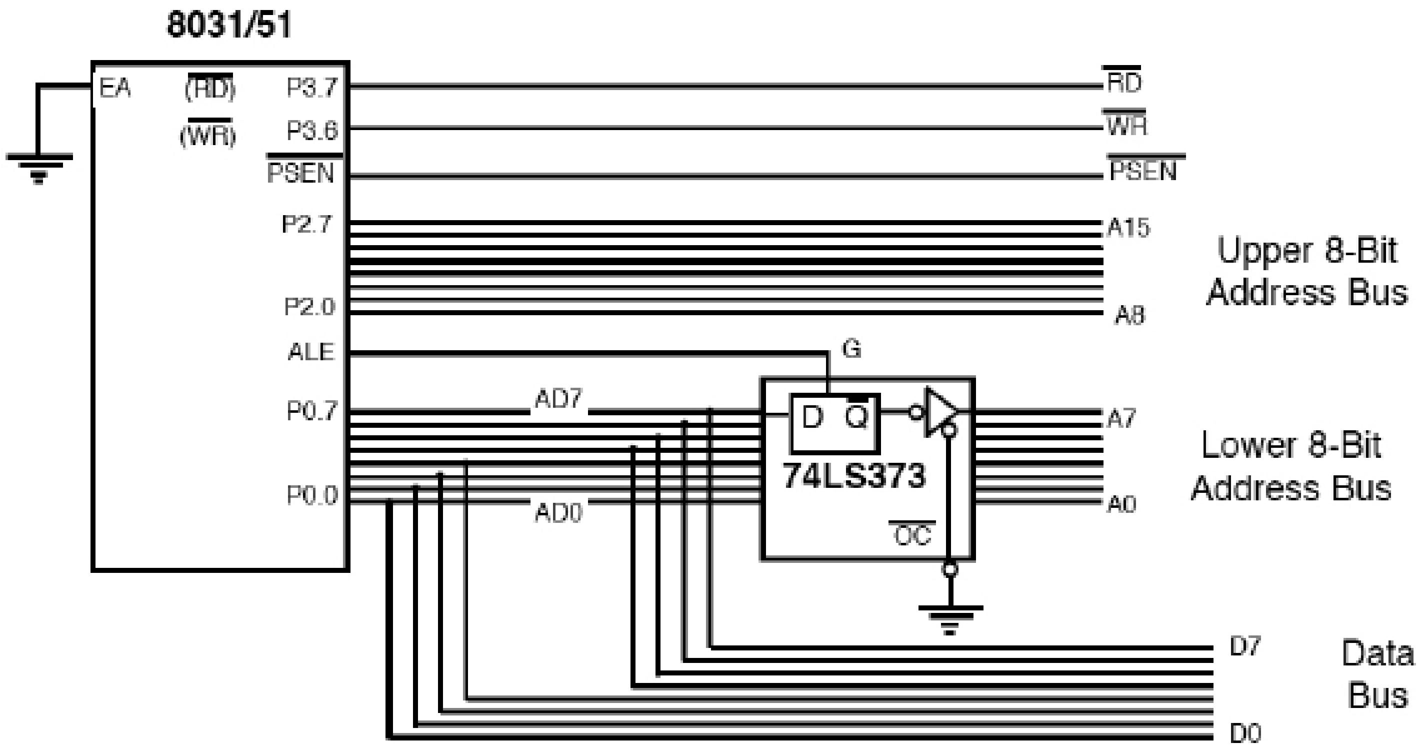
ADDRESS THE 8051 MOCROCONTROLLER ON-CHIP MEMORY

- Bit Addressing
  - 8051 contains 210 bit-addressable locations.
  - 128 of these locations are at addresses 20H to 2FH and the rest are in the special function registers.

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)

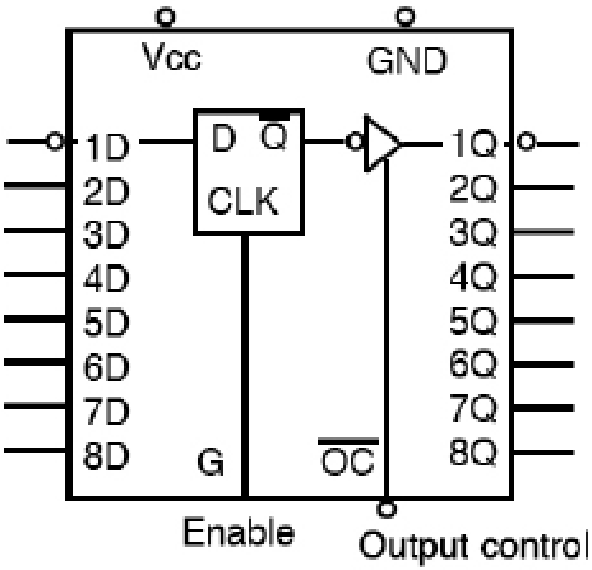
ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- Address Multiplexing for External Memory



Multiplexing the address (low-byte) and data bus

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY



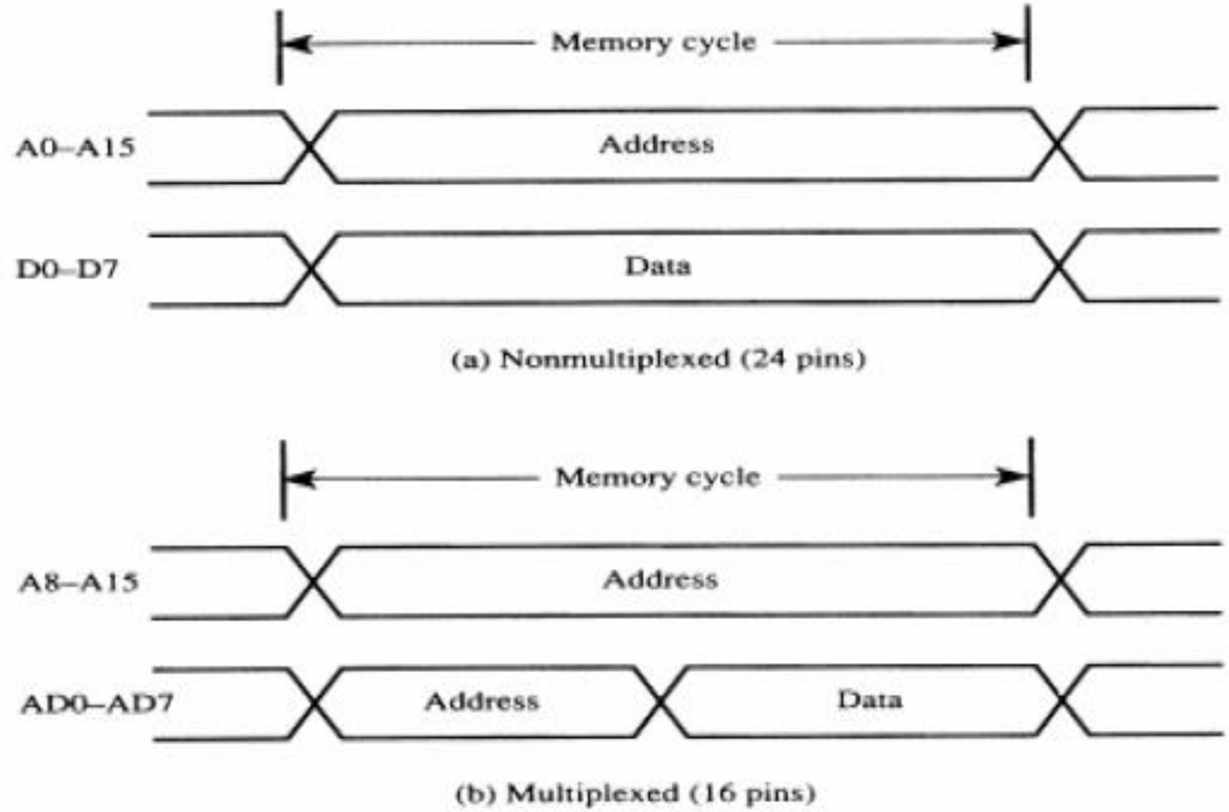
Funtion Table

Output control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

Figure 3–14 74LS373 D Latch

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- Address Multiplexing for External Memory



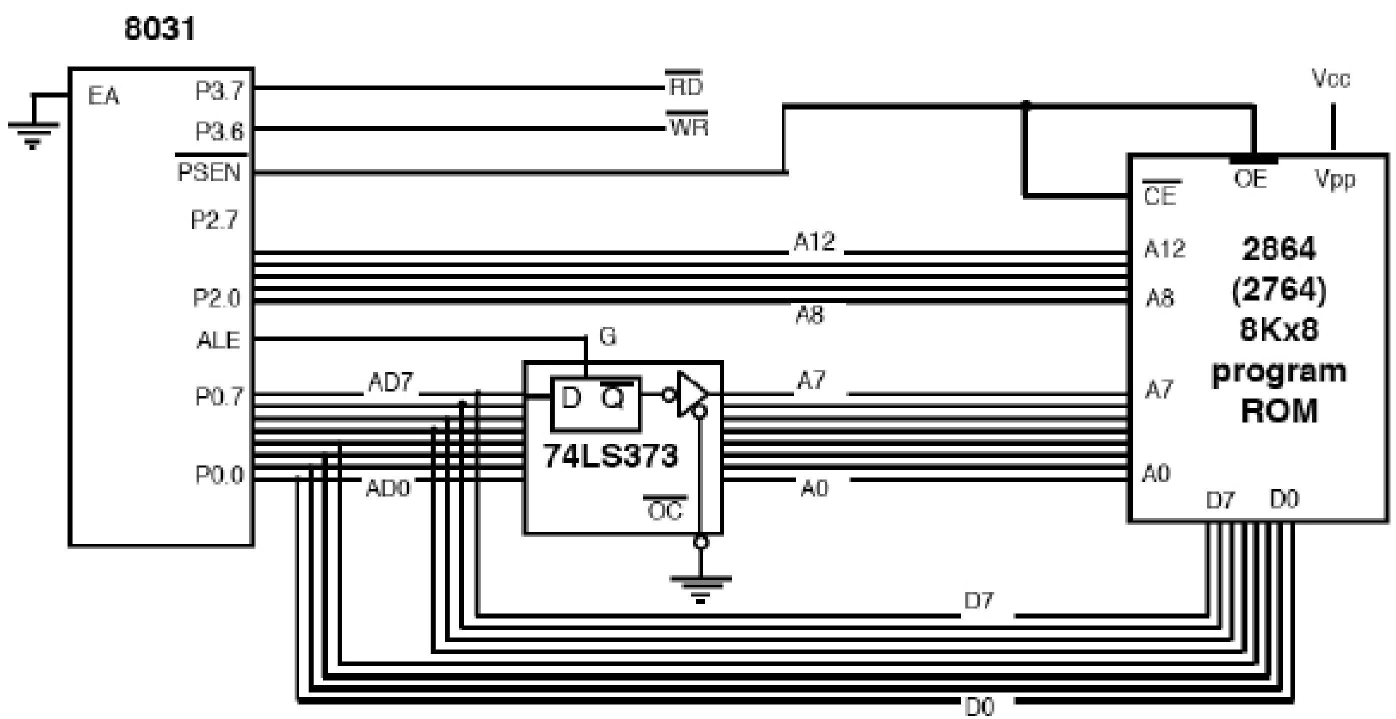
Multiplexing the address (low-byte) and data bus

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- **Accessing external code memory:** An 8031 microcontroller with no on-chip ROM is to be connected to an external 8KByte code ROM chip.

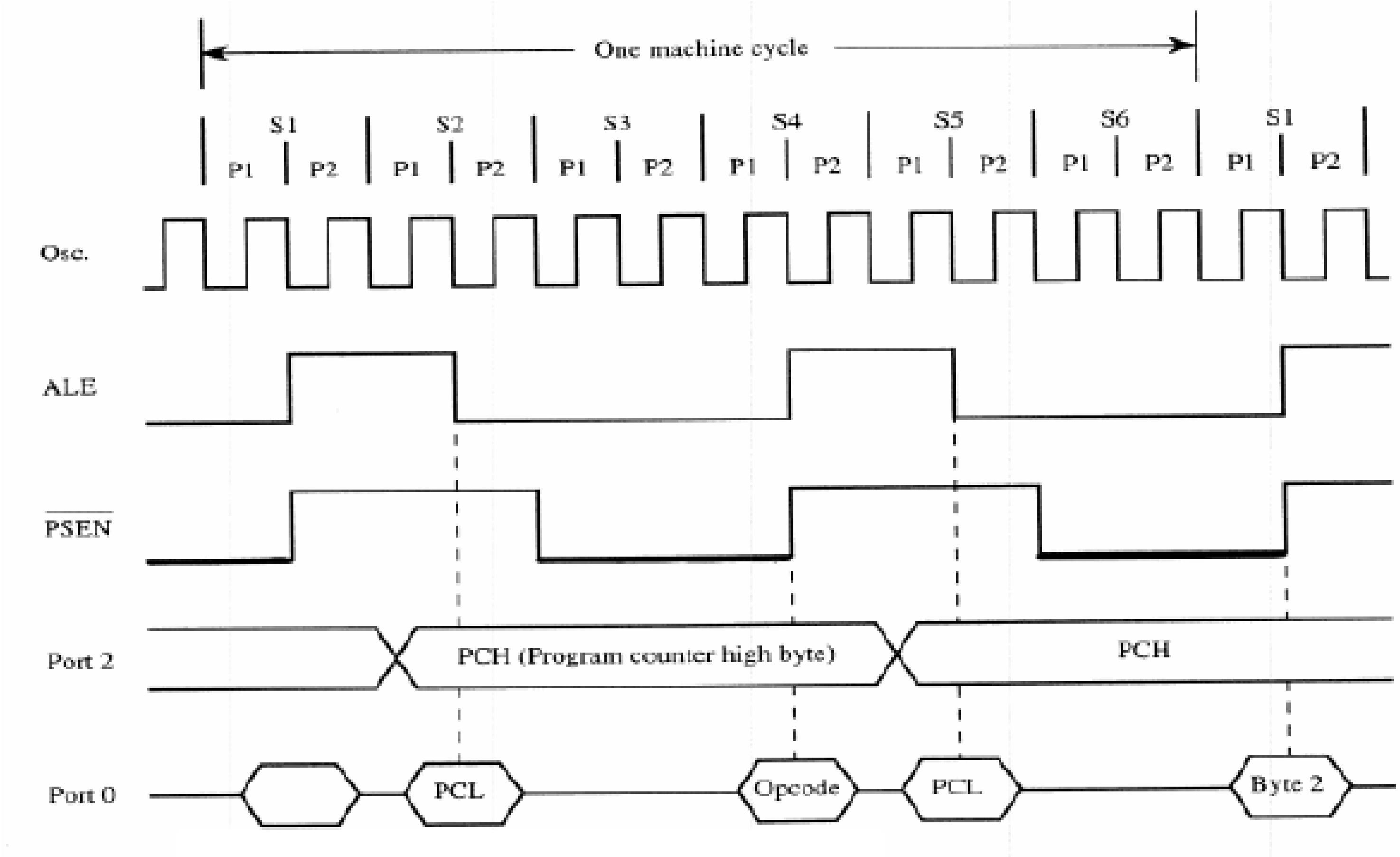
  - Need 13-bit address bus (why?)
  - The lower 8 bits on the address bus come from Port 0
  - Only bits 8-12 from Port 2 are used in the address bus upper bits
  - The PSEN pin is connected to the CE (chip enable) pin and OE (output enable) pin on the external ROM chip

- Accessing External Code Memory



Accessing external code memory

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY



Reading timing for external code memory

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- Accessing external data memory: An 8051 microcontroller is to be connected to an 8KByte external data ROM chip. The data is to be addressed using the addresses 0000 – 1FFFH

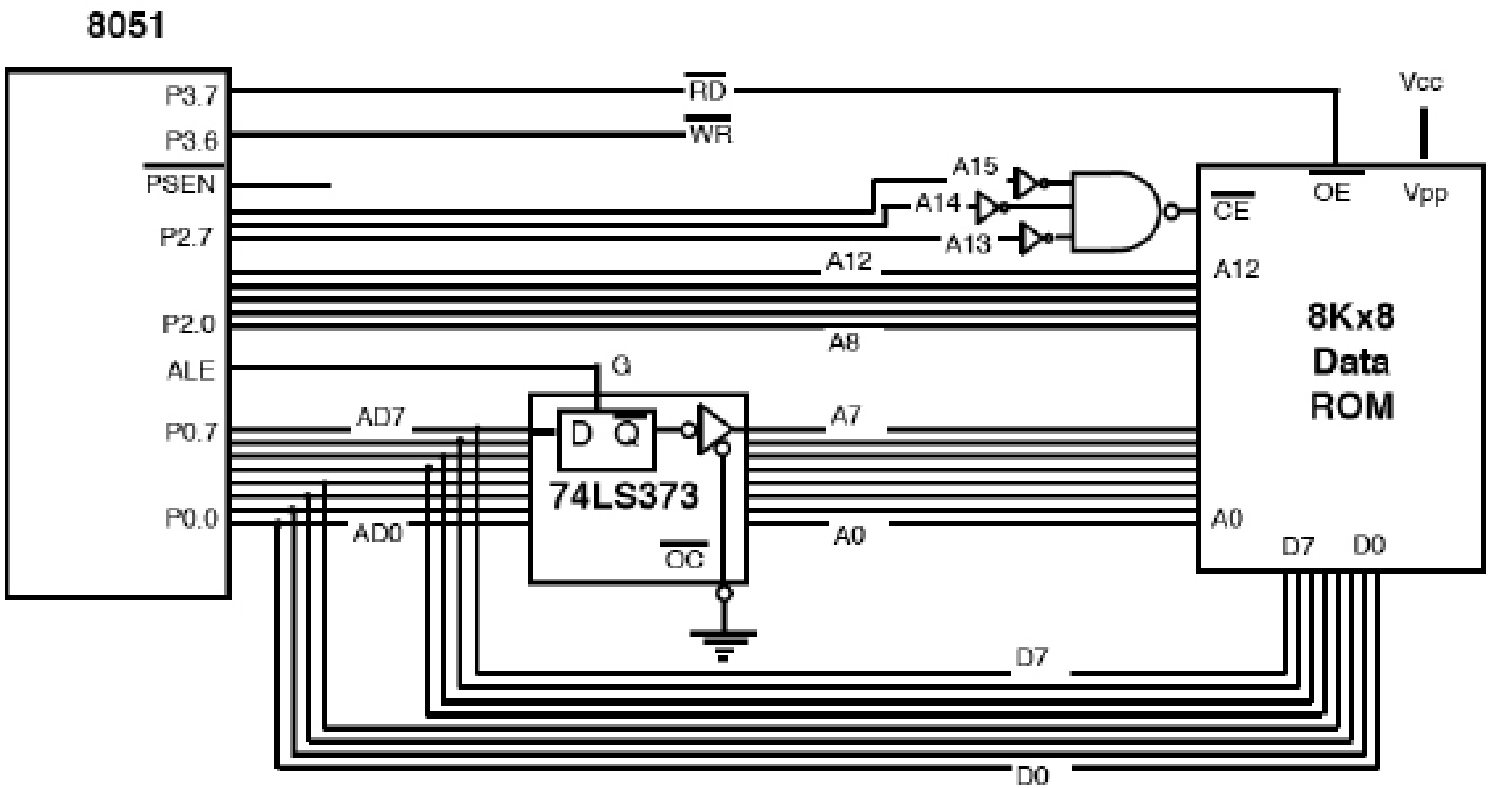
  - The Output enable pin is connected to the Read pin (P3.7, pin 17) of the 8051 microcontroller

  - The CE pin is connected to a logical circuit that selects the data ROM chip only when the specified address is within the expected range of 0000 – 1FFF

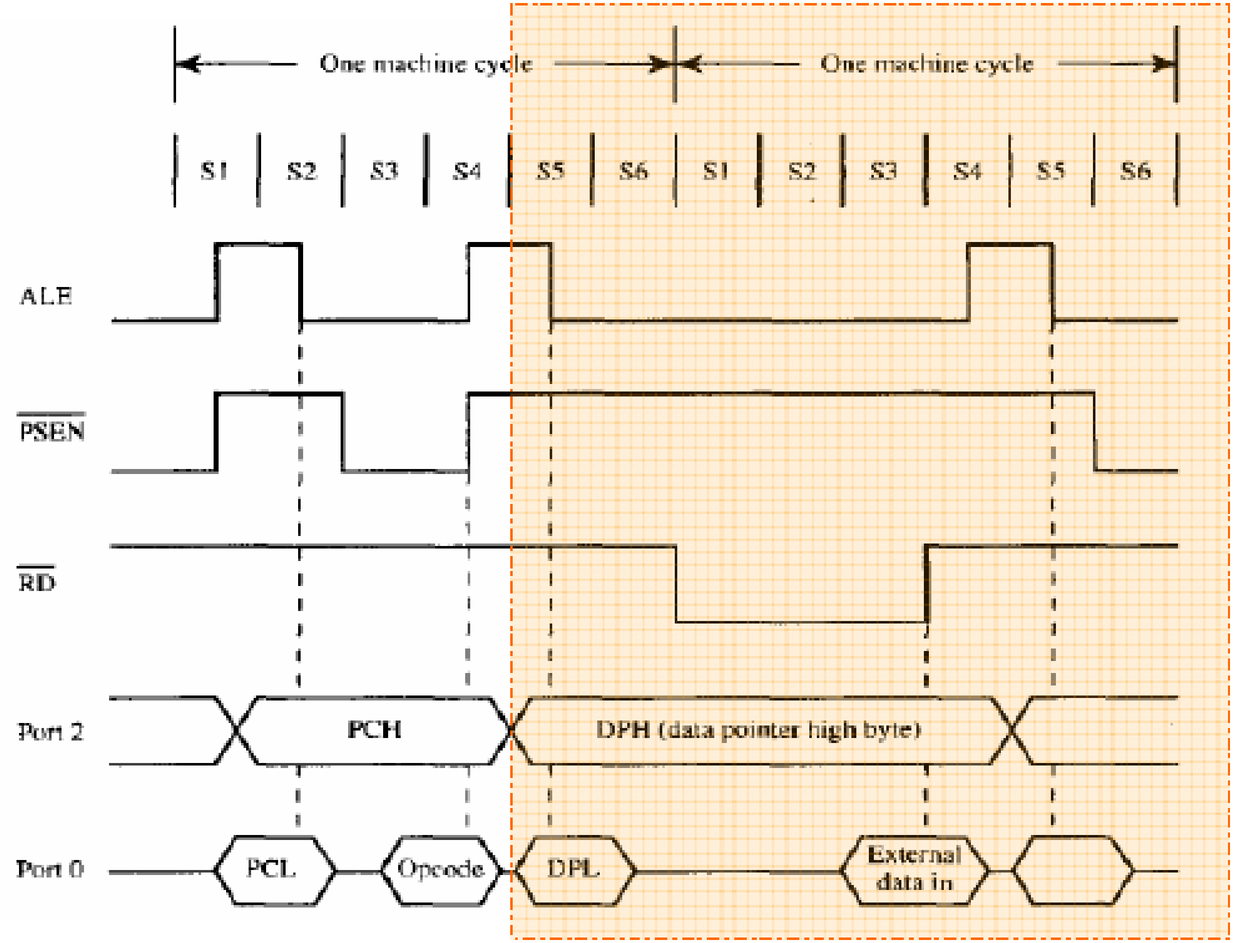
		Address Bits															
		A 15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Data ROM Address	Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0				0				0				0			
	End	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
		1				F				F				F			

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- Accessing external data memory



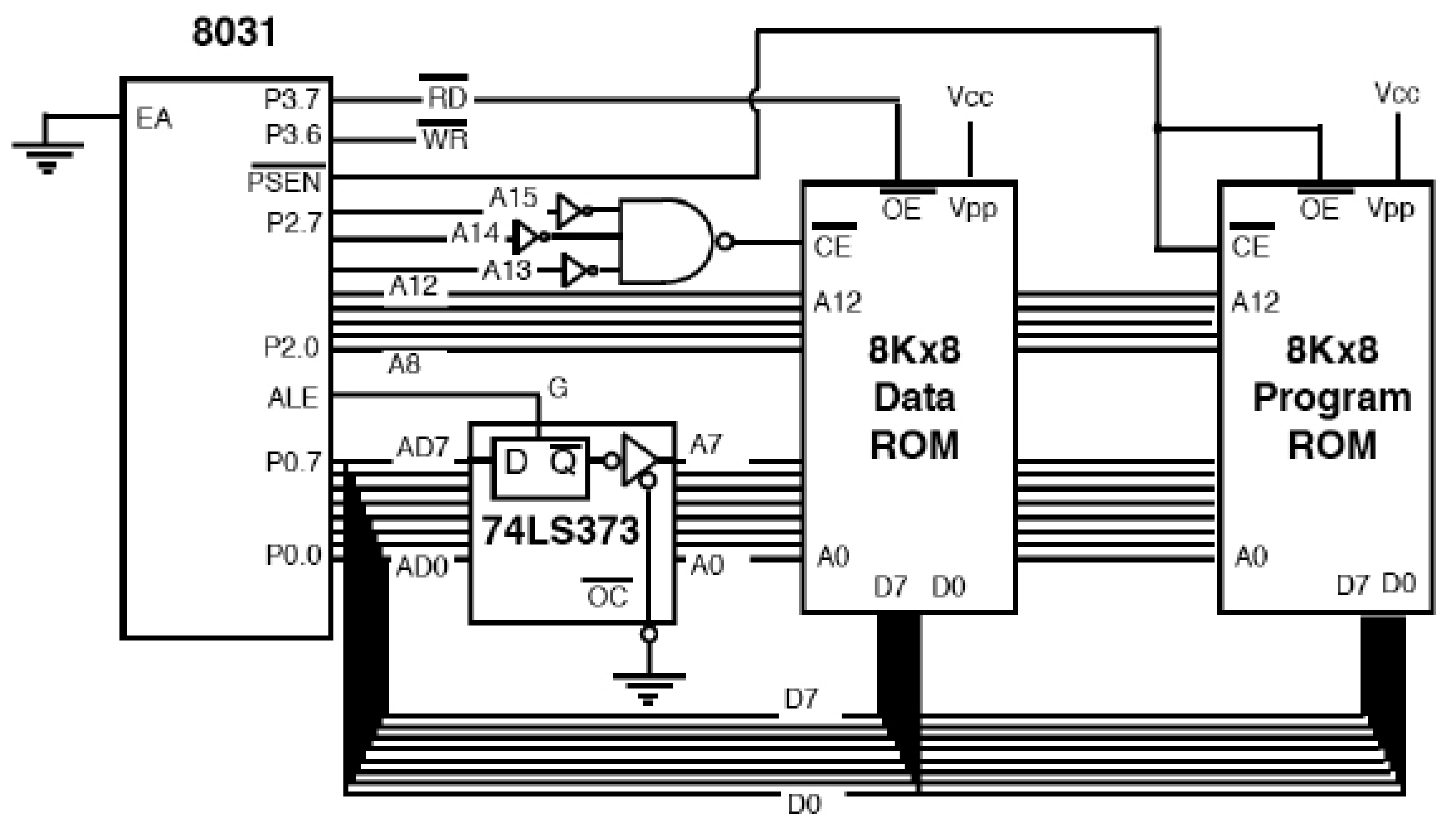
ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY



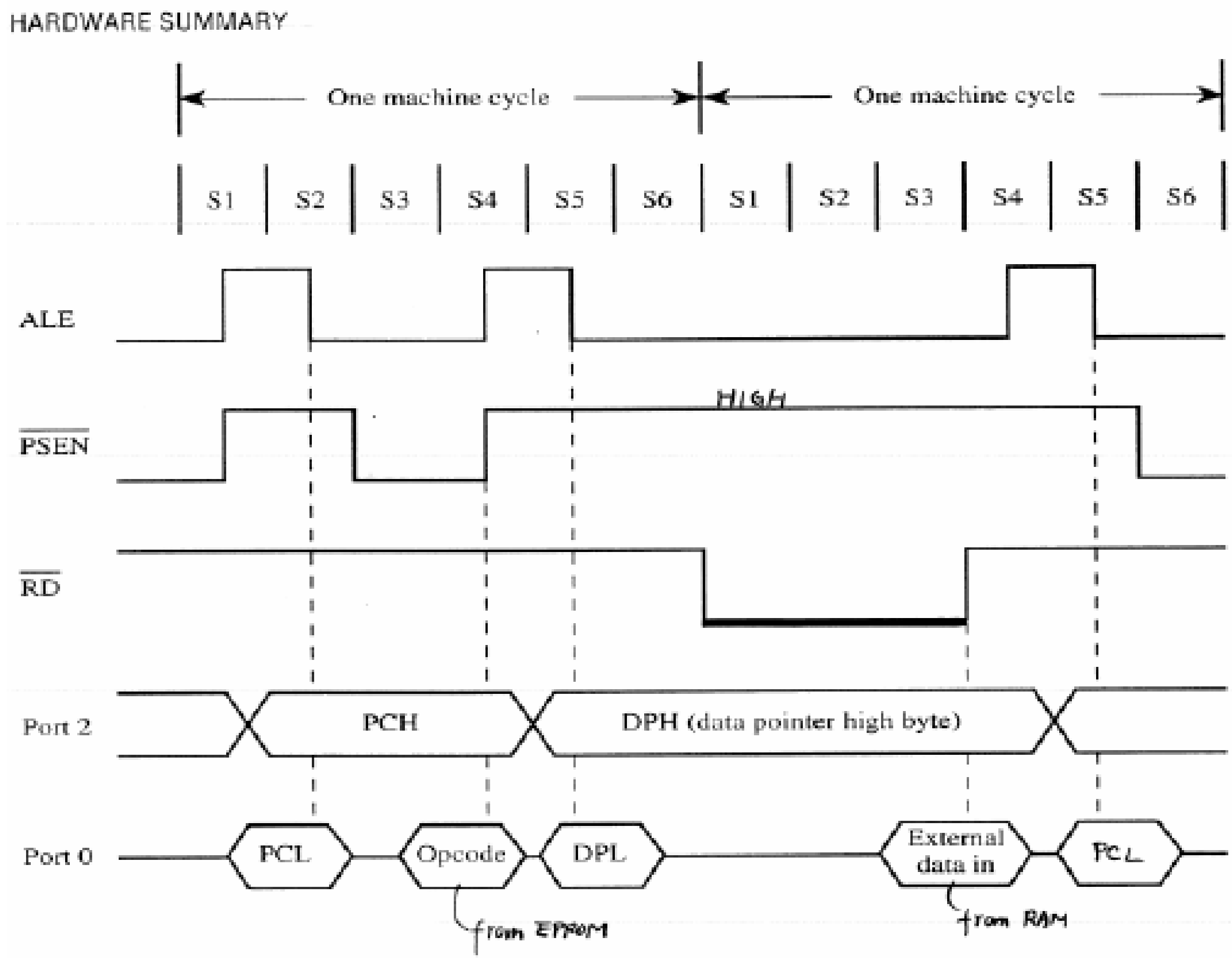
Reading timing for external data memory

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- Accessing external memory



ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY



ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- 8031 Interfacing both with ROM and RAM as
  - ➢ A 16KByte data RAM chip to be used for data addresses starting at 0000,
  - ➢ A 16KByte data ROM chip to be used for data addresses starting at 8000, and
  - ➢ A 16KByte code ROM chip

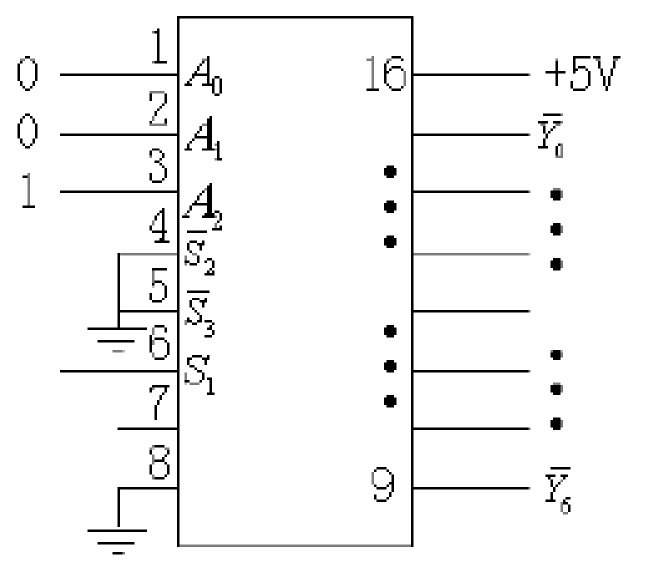
Solution:

☞ Since all the chips have 16K different addresses, 14-bit addresses need to be used

		Address Bits															
		B	A														
		A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Data ROM Address	Start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		0				0				0				0			
	End	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		3				F				F				F			
Data RAM Address	Start	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		8				0				0				0			
	End	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
		B				F				F				F			

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

- The 74LS138 is a 3-to-8 decoder which enables one of eight outputs Y0 to Y7 (only Y0 to Y3 are shown in the figure) based on the values of the A, B, and C. In the figure below, C is fixed at 0 since its pin is connected to ground. Therefore, different values of A and B lead to different outputs at the pins Y0 to Y7.



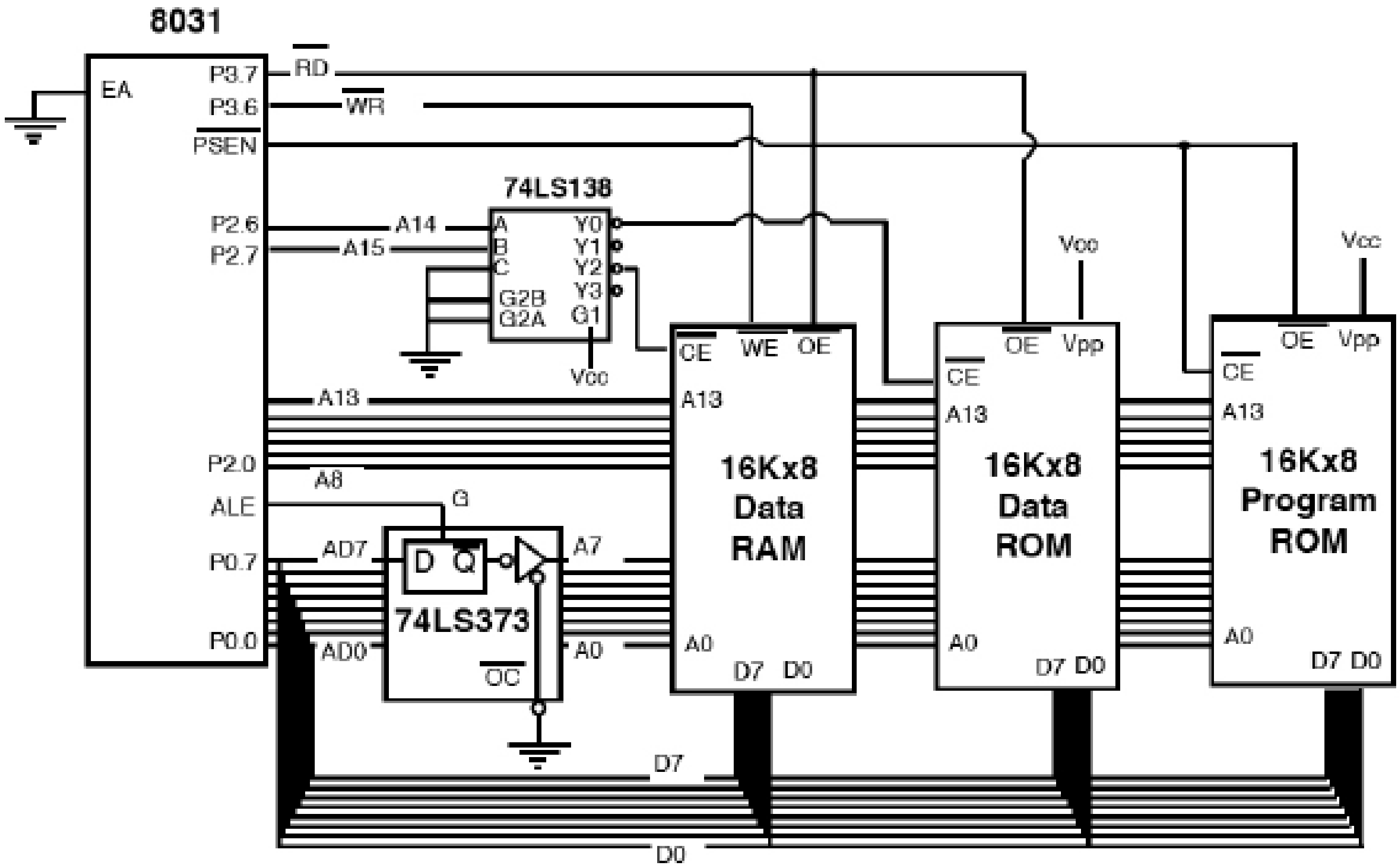
The truth table for a 3-to-8 decoder

- The 74LS138 3-to-8 decoder

Inputs			Outputs							
A	B	C	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

The truth table for a 3-to-8 decoder

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY



8031 interfacing external memory

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

x	2^x
10	1K
11	2K
12	4K
13	8K
14	16K
15	32K
16	64K
17	128K
18	256K
19	512K
20	1M
21	2M
22	4M
23	8M
24	16M
25	32M
26	64M
27	128M

Table 3–1 Powers of 2

ADDRESS THE 8051 MOCROCONTROLLER EXTERNAL MEMORY

Example 5-7

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

(a) This method uses a counter

```
        ORG 0000
        MOV DPTR, #MYDATA      ; Load ROM pointer
        MOV R0, #40H           ; Load RAM pointer
        MOV R2, #7             ; Load counter
  Back: CLR A                  ; A=0
        MOVC A,@A+DPTR         ; Move data from code space
        MOV @R0, A             ; Save it in RAM
        INC DPTR               ; Increment from pointer
        INC R0                 ; Increment from pointer
        DJNZ R2, BACK          ; Loop until counter = 0
 HERE:  SJMP HERE
        ORG 250
MYDATA: DB “AMERICA”
        END
```