MSE381: Feedback Control Systems

Lecture 2

8051 Microcontroller and Embedded Programming

Mohammad Narimani, *Ph.D., P.Eng.*
Lecturer
School of Mechatronic Systems Engineering
Simon Fraser University

SFU SIMON FRASER UNIVERSITY
SURREY

Outline

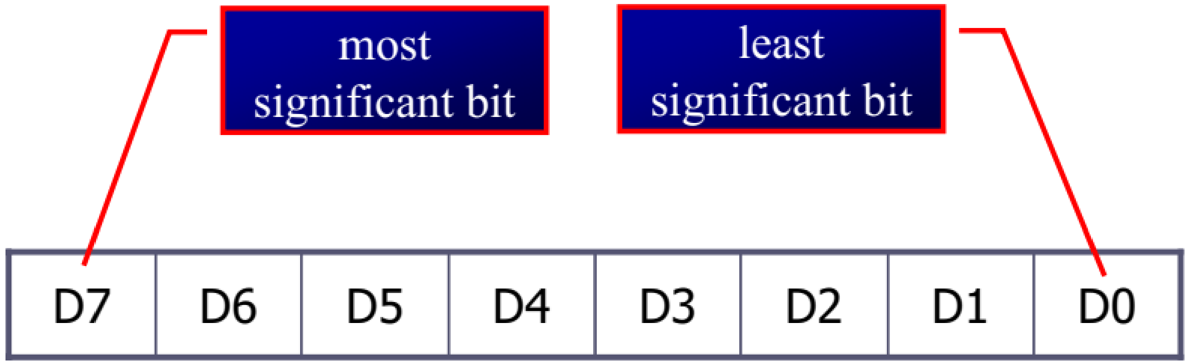
- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and Program Status Word (PSW) register
- Register banks and stack
- Hardware connection and Intel HEX file

Inside the 8051- registers and MOV & ADD instructions

- Register are used to store information temporarily, while the information could be
  - a byte of data to be processed, or
  - an address pointing to the data to be fetched
- The vast majority of 8051 registers are 8-bit registers
  - here is only one data type: 8 bits

Inside the 8051- registers and MOV & ADD instructions

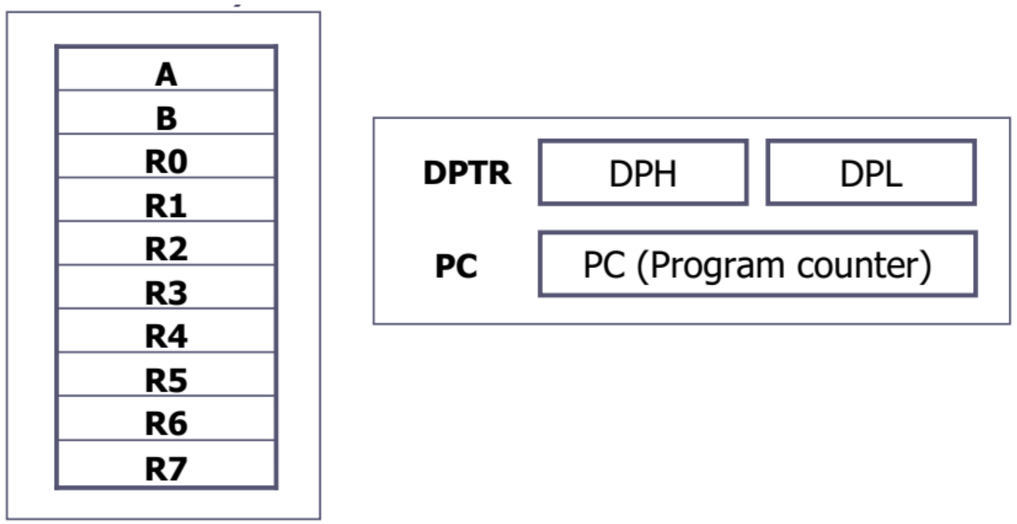
- The 8 bits of a register are shown from MSB D7 to the LSB D0
  - With an 8-bit data type, any data larger than 8 bits must be broken into 8-bit chunks before it is processed



8 bit Registers

Inside the 8051- registers and MOV & ADD instructions

- The most widely used registers
  - A or ACC (Accumulator)
    - For all arithmetic and logic instructions
  - B, R0, R1, R2, R3, R4, R5, R6, R7
  - DPTR (data pointer), and PC (program counter)



Inside the 8051- registers and MOV & ADD instructions

- **MOV destination, source** ;copy source to destination
  - The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand

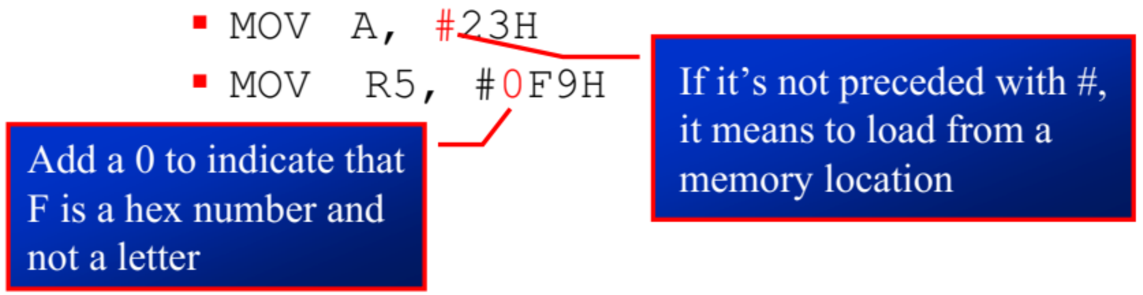
"#" signifies that it is a value

```
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
```

For a full list of available instructions in 8051, check: https://www.win.tue.nl/~aeb/comp/8051/set8051.html or http://www.keil.com/support/man/docs/is51/is51_instructions.htm

Inside the 8051- registers and MOV & ADD instructions

- Notes on programming
  - Value (proceeded with #) can be loaded directly to registers A, B, or R0 – R7



```
▪ MOV   A,  #23H
▪ MOV   R5,  #0F9H
```

Add a 0 to indicate that F is a hex number and not a letter

If it's not preceded with #, it means to load from a memory location

  - If values 0 to F moved into an 8-bit register, the rest of the bits are assumed all zeros
    - "MOV A, #5", the result will be A=05; i.e., A = 00000101 in binary
  - Moving a value that is too large into a register will cause an error

Inside the 8051- registers and MOV & ADD instructions

- **ADD A, source** ; ADD the source operand to the accumulator
  - The ADD instruction tells the CPU to add the source byte to register A and put the result in register A
  - Source operand can be either a register or immediate data, but the destination must always be register A
    - "ADD R4, A" and "ADD R2, #12H" are invalid since A must be the destination of any arithmetic operation

```
MOV A, #25H      ;load 25H into A
MOV R2, #34H     ;load 34H into R2
ADD A, R2        ;add R2 to Accumulator
                 ;(A = A + R2)
```

Outline

- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and PSW register
- Register banks and stack
- Hardware connection and Intel HEX file

8051 assembly programming

- 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 provide 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

8051 assembly programming

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

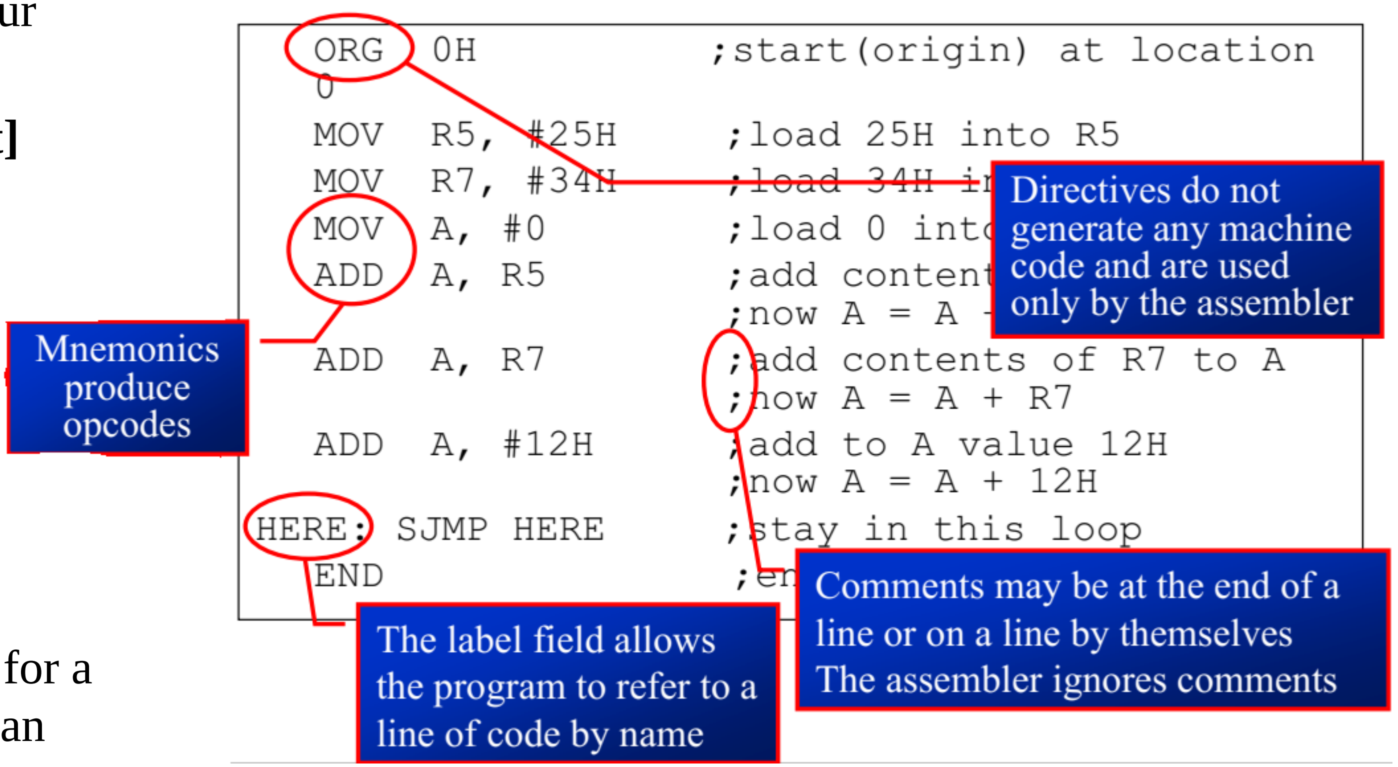
8051 assembly programming

- An Assembly language instruction consists of four fields:

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

- **OPCODE:** It is a number interpreted by your machine(virtual or silicon) that represents the operation to perform.

- **MNEMONIC:** a mnemonic is a symbolic name for a single executable machine language instruction (an opcode)



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

Mnemonics produce opcodes
Directives do not generate any machine code and are used only by the assembler
The label field allows the program to refer to a line of code by name
Comments may be at the end of a line or on a line by themselves The assembler ignores comments

8051 assembly programming

- The steps of Assembly language program are outlines as follows:

  - 1) An editors or word processors is used to create and/or edit the program
    - Notice that the editor must be able to produce an ASCII file
    - For many assemblers, the source file has the extension “asm”

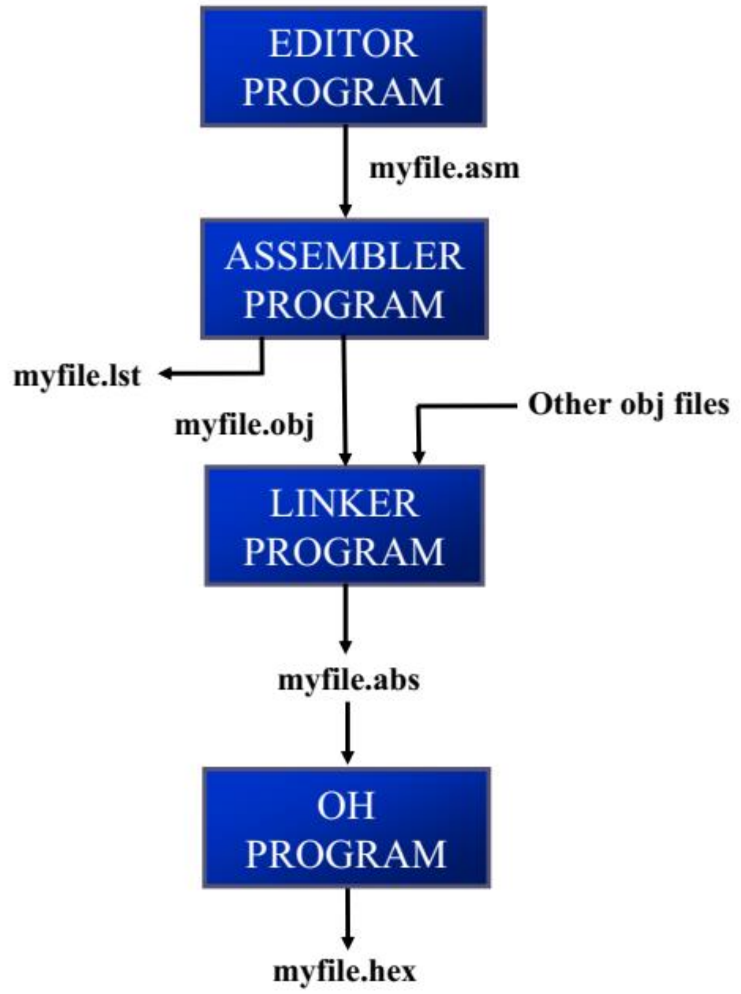
  - 2) The “asm” source file containing the program code created in step 1 is fed to an 8051 assembler
    - The assembler converts the instructions into machine code
    - The assembler will produce an object file and a list file
    - The extension for the object file is “obj” while the extension for the list file is “lst”

8051 assembly programming

- 3) Assembler require a third step called linking
    - The linker program takes one or more object code files and produce an absolute object file with the extension “abs”
    - This abs file is used by 8051 trainers that have a monitor program

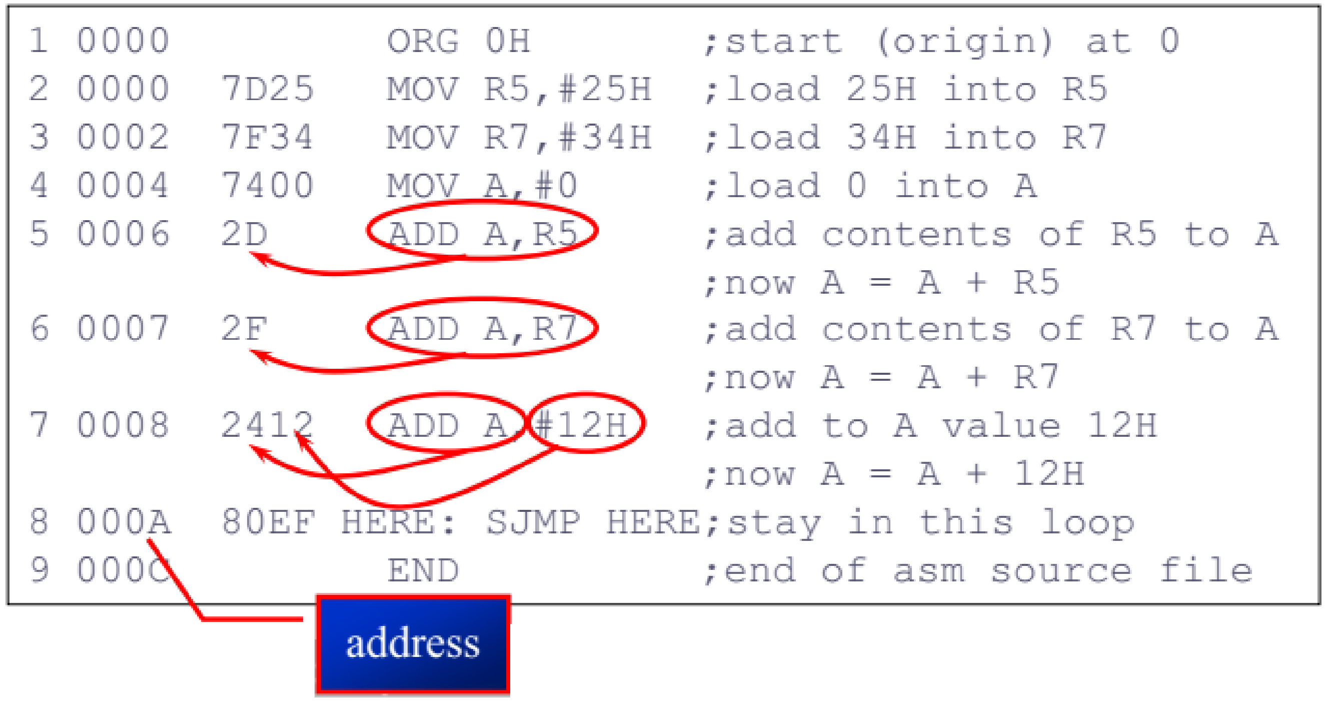
- 4) Next the “abs” file is fed into a program called “OH” (object to hex converter) which creates a file with extension “hex” that is ready to burn into ROM
    - This program comes with all 8051 assemblers
    - Recent Windows-based assemblers combine step 2 through 4 into one step

8051 assembly programming



8051 assembly programming

- The lst (list) file, which is optional, is very useful to the programmer
  - It lists all the opcodes and addresses as well as errors that the assembler detected
  - The programmer uses the lst file to find the syntax errors or debug



```
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
```

Outline

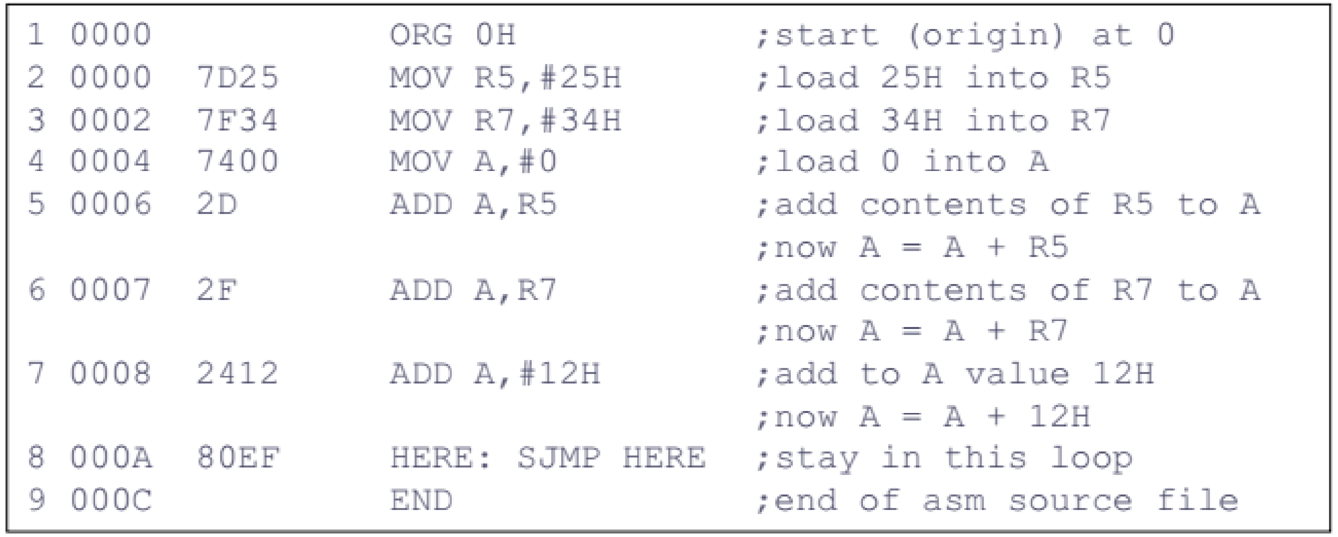
- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and PSW register
- Register banks and stack
- Hardware connection and Intel HEX file

Program counter and ROM space

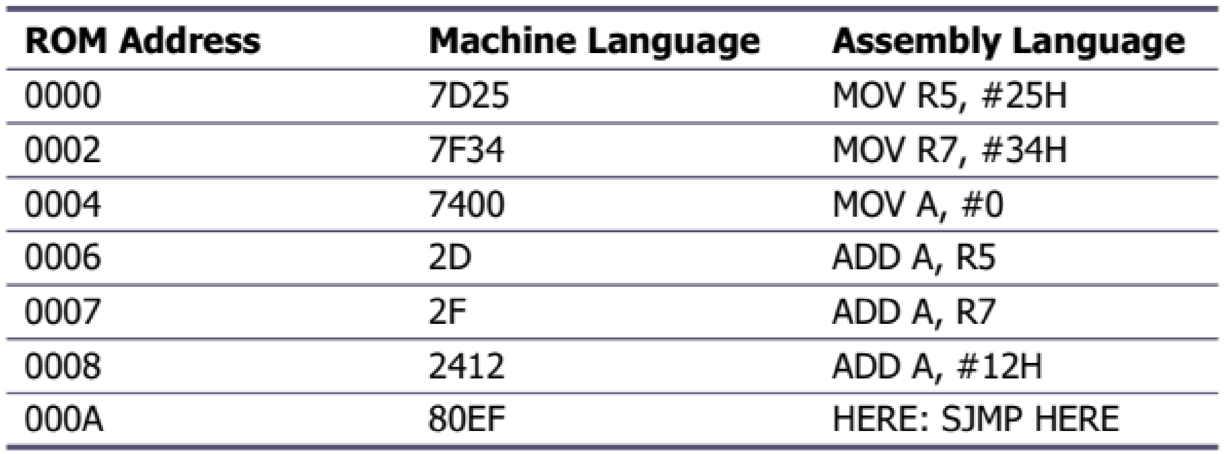
- The program counter (PC) points to the address of the next instruction to be executed
  - As the CPU fetches the opcode from the program ROM, the program counter is increasing to point to the next instruction
- The program counter is 16 bits wide
  - This means that it can access program addresses 0000H to FFFFH, a total of 64K bytes of code
- All 8051 members start at memory address 0000H when they're powered up
  - Program Counter has the value of 0000H
  - 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

Program counter and ROM space

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



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



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

Program counter and ROM space

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

ROM contents

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

Program counter and ROM space

- A step-by-step description of the action of the 8051 upon applying power on it

  1. When 8051 is powered up, the PC has 0000 and starts to fetch the first opcode from location 0000 of program ROM
     - Upon executing the opcode 7D, the CPU fetches the value 25 and places it in R5

  2. Upon executing the opcode 7F, the value 34H is moved into R7
     - The PC is incremented to 0004

In computing, an opcode (abbreviated from operation code, also known as instruction syllable, instruction parcel or opstring) is the portion of a machine language instruction that specifies the operation to be performed.

Program counter and ROM space

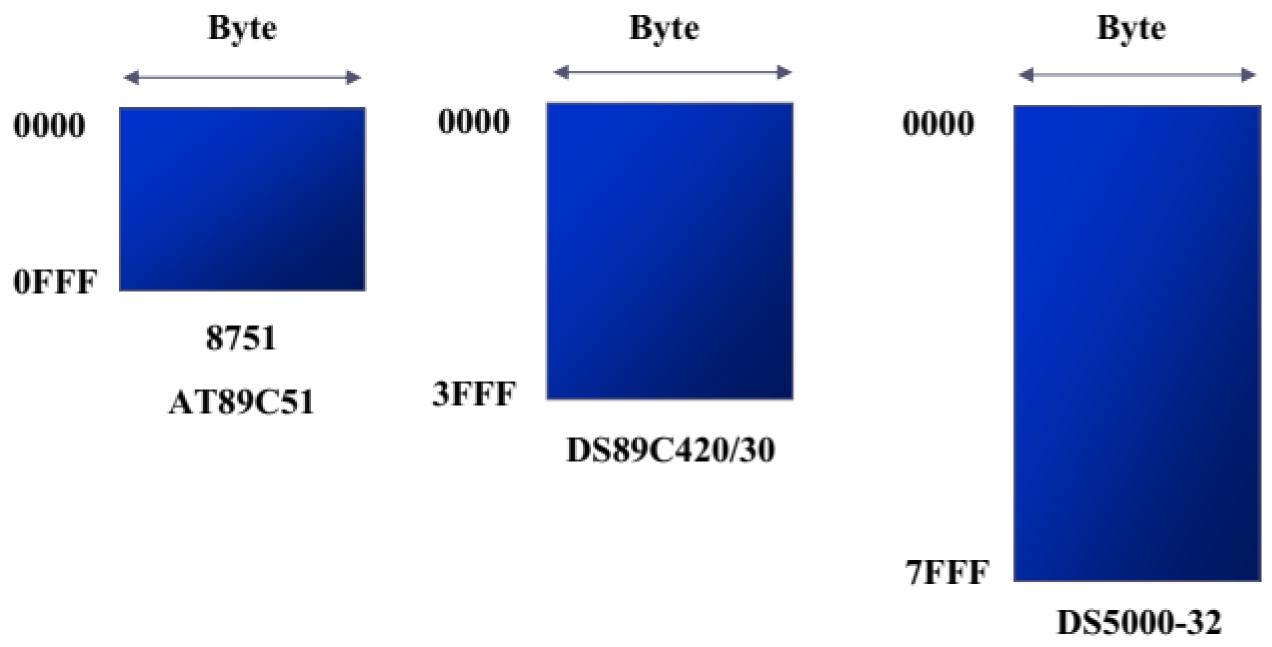
3. The instruction at location 0004 is executed and now PC = 0006

4. After the execution of the 1-byte instruction at location 0006, PC = 0007

5. Upon execution of this 1-byte instruction at 0007, PC is incremented to 0008
   - This process goes on until all the instructions are fetched and executed
   - The fact that program counter points at the next instruction to be executed explains some microprocessors call it the instruction pointer

Program counter and ROM space

- No member of 8051 family can access more than 64K bytes of opcode
  - The program counter is a 16-bit register



Outline

- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and PSW register
- Register banks and stack
- Hardware connection and Intel HEX file

8051 data types and directives

- 8051 microcontroller has only one data type - 8 bits
  - The size of each register is also 8 bits
  - It is the job of the programmer to break down data larger than 8 bits (00H to FFH, or 0 to 255 in decimal)
  - The data types can be positive or negative

8051 data types and directives

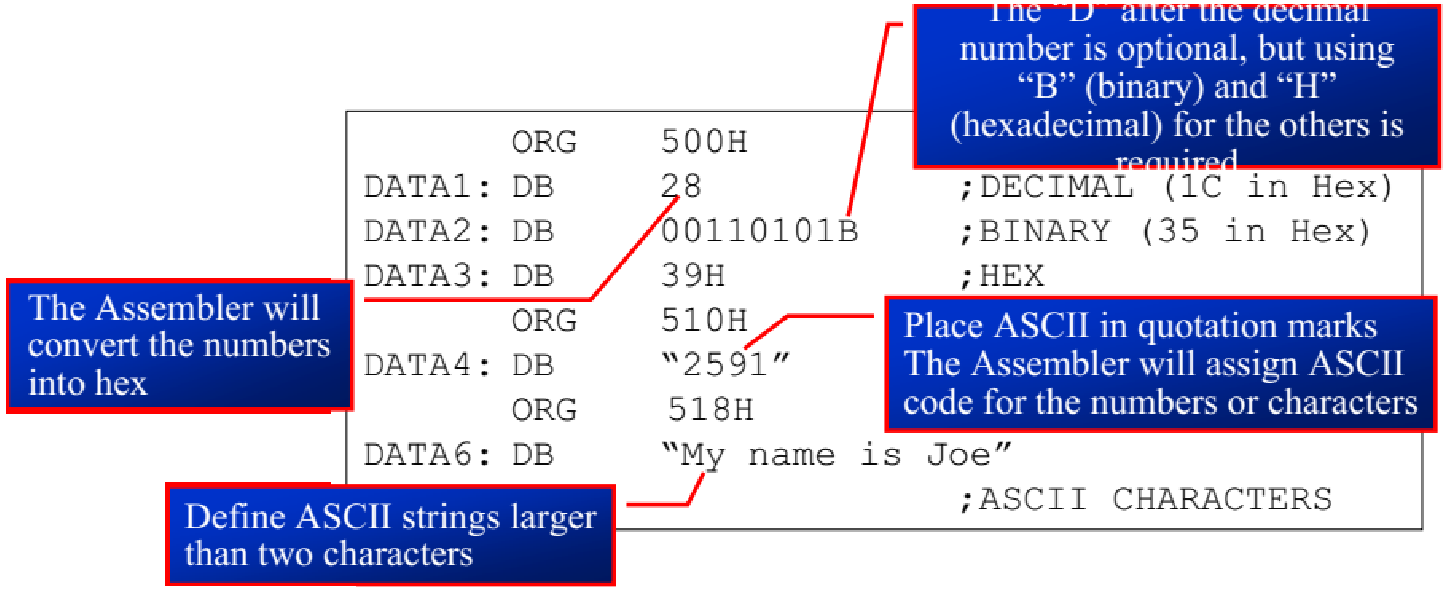
- Assembler Directives
  - Directives do many things;
    - ✓ tell the assembler to set aside space for variables,
    - ✓ tell the assembler to include additional source files
    - ✓ establish the start address for your program.

The directives available are shown below:

DB, EQU, ORG, END

8051 data types and directives

- **DB (define Byte):** The DB directive is the most widely used data directive in the assembler
  - It is used to define the 8-bit data
  - When DB is used to define data, the numbers can be in decimal, binary, hex, ASCII formats



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

The "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others is required
The Assembler will convert the numbers into hex
Place ASCII in quotation marks The Assembler will assign ASCII code for the numbers or characters
Define ASCII strings larger than two characters

8051 data types and directives

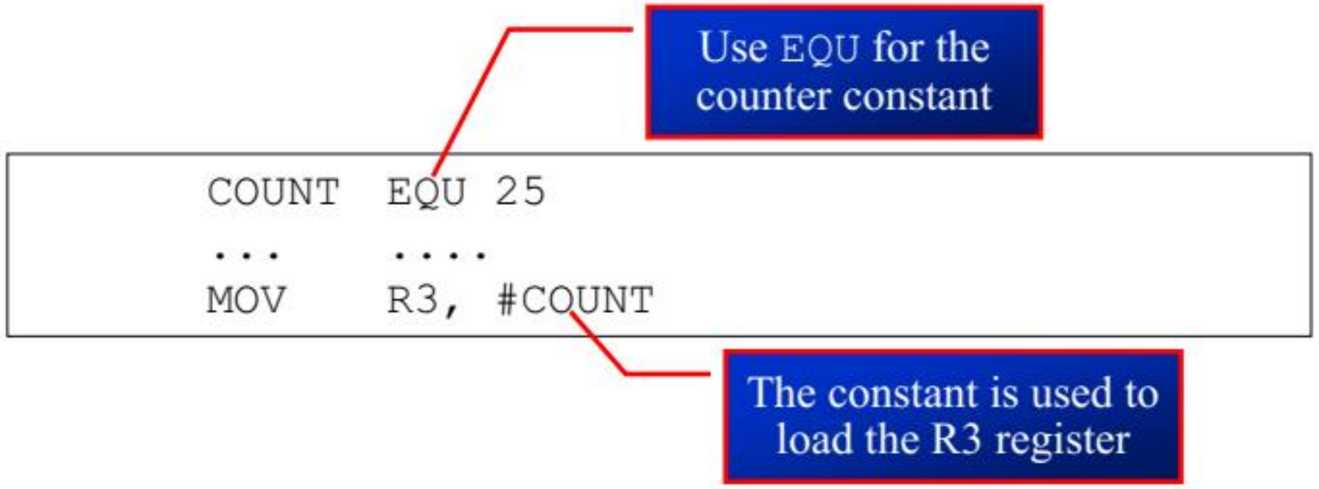
- ORG (origin)
  - The ORG directive is used to indicate the beginning of the address
  - The number that comes after ORG can be either in hex and decimal
    - If the number is not followed by H, it is decimal and the assembler will convert it to hex
- END
  - This indicates to the assembler the end of the source (asm) file
  - The END directive is the last line of an 8051 program
    - Means that in the code anything after the END directive is ignored by the assembler

8051 data types and directives

- EQU (equate)
  - This is used to define a constant without occupying a memory location
  - The EQU directive does not set aside storage for a data item but associates a constant value with a data label
    - When the label appears in the program, its constant value will be substituted for the label

8051 data types and directives

- Assume that there is a constant used in many different places in the program, and the programmer wants to change its value throughout
  - By the use of EQU, one can change it once and the assembler will change all of its occurrences



```
COUNT   EQU 25
...     ....
MOV     R3, #COUNT
```

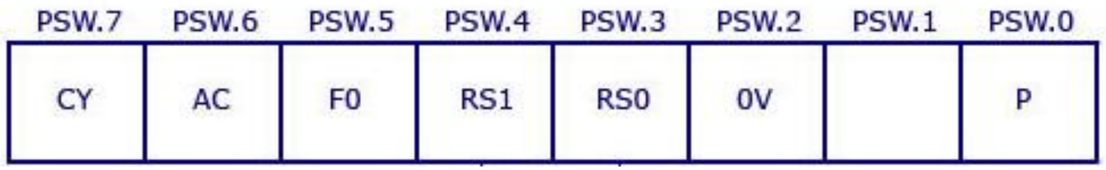
Use EQU for the counter constant
The constant is used to load the R3 register

Outline

- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and PSW register
- Register banks and stack
- Hardware connection and Intel HEX file

Flag bits and PSW register

- The program status word (PSW) register, also referred to as the flag register, is an 8 bit register



PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV		P

  - 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 PSW.3 and PSW.4 are designed as RS0 and RS1, and are used to change the bank
  - The two unused bits are user-definable

Flag bits and PSW register

CY	AC	F0	RS1	RS0	OV	--	P

A carry from D3 to D4

Carry out from the d7 bit

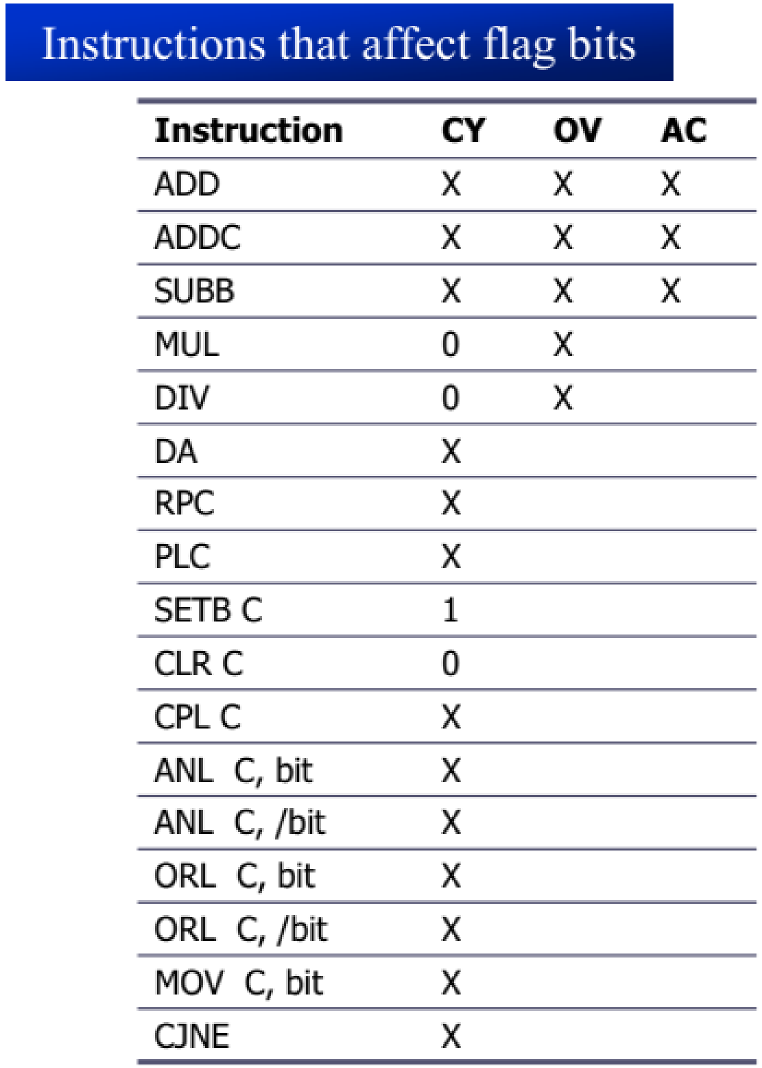
CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
--	PSW.5	Available to the user for general purpose
RS1	PSW.4	Register Bank selector bit 1.
RS0	PSW.3	Register Bank selector bit 0.
OV	PSW.2	Overflow flag.
--	PSW.1	User definable bit.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle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

Reflect the number of 1s in register A

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

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

Flag bits and PSW register

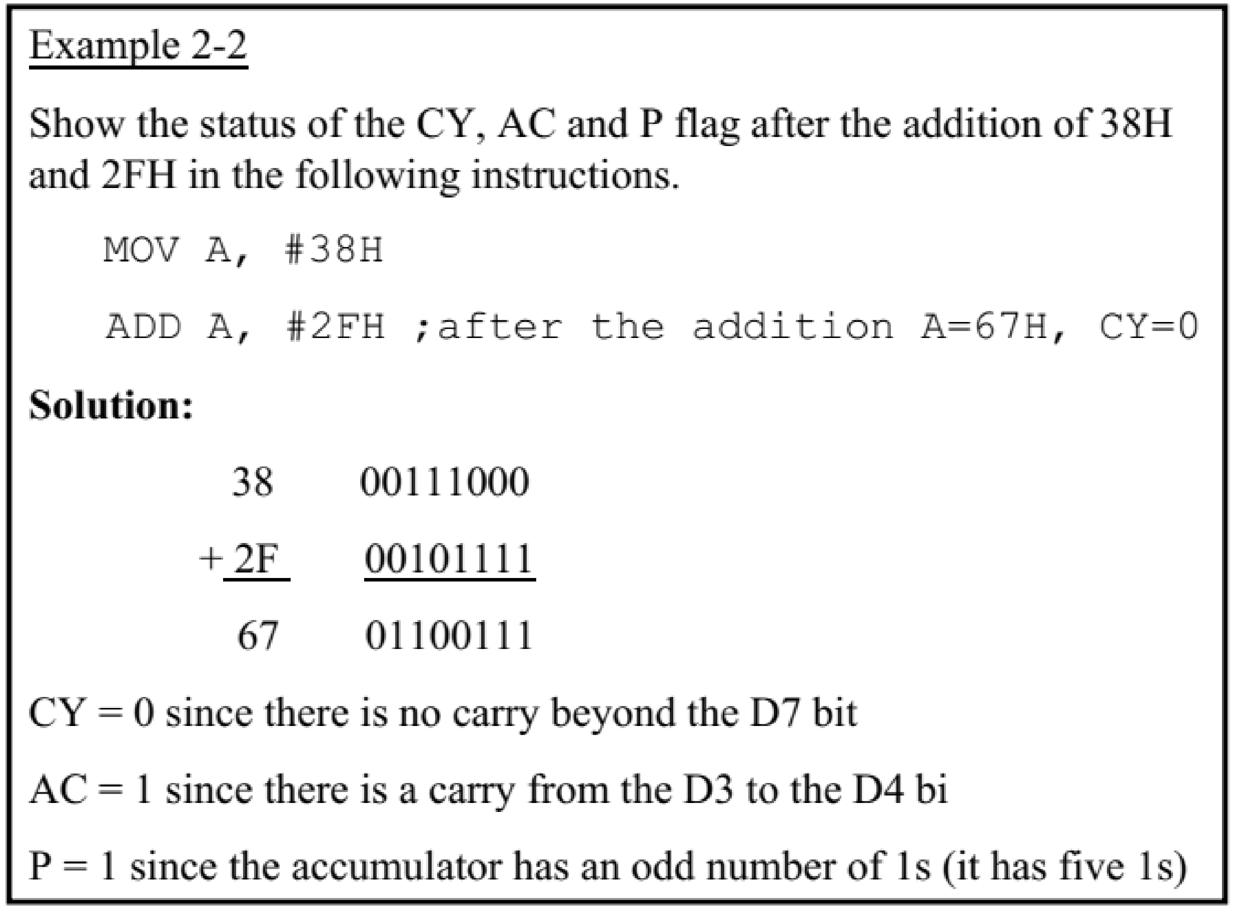


Instructions that affect flag bits

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

Flag bits and PSW register

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



Example 2-2

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

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

Solution:

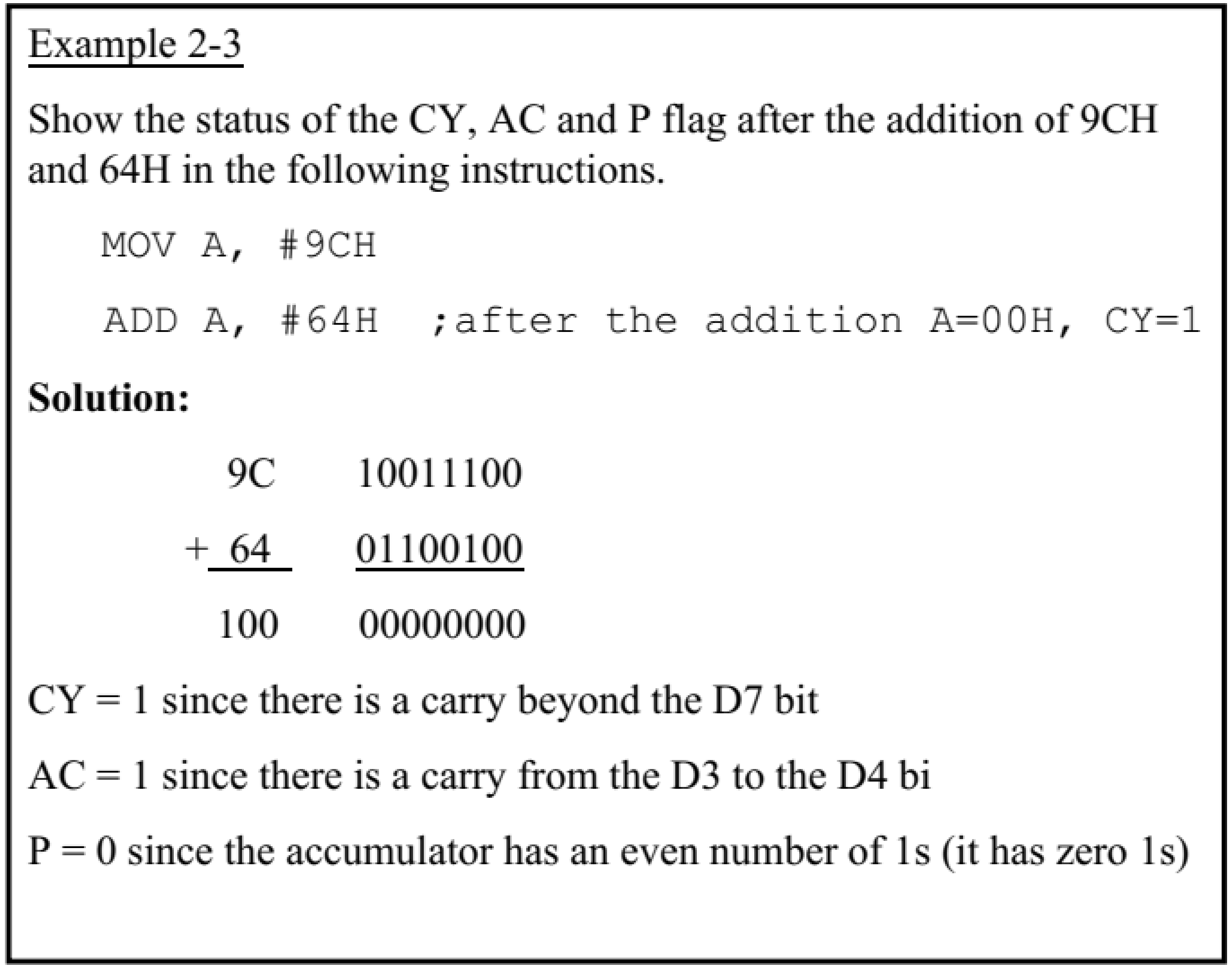
```
      38      00111000
    + 2F      00101111
      67      01100111
```

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

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

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

Flag bits and PSW register



Example 2-3

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

```
    MOV  A,  #9CH

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

Solution:

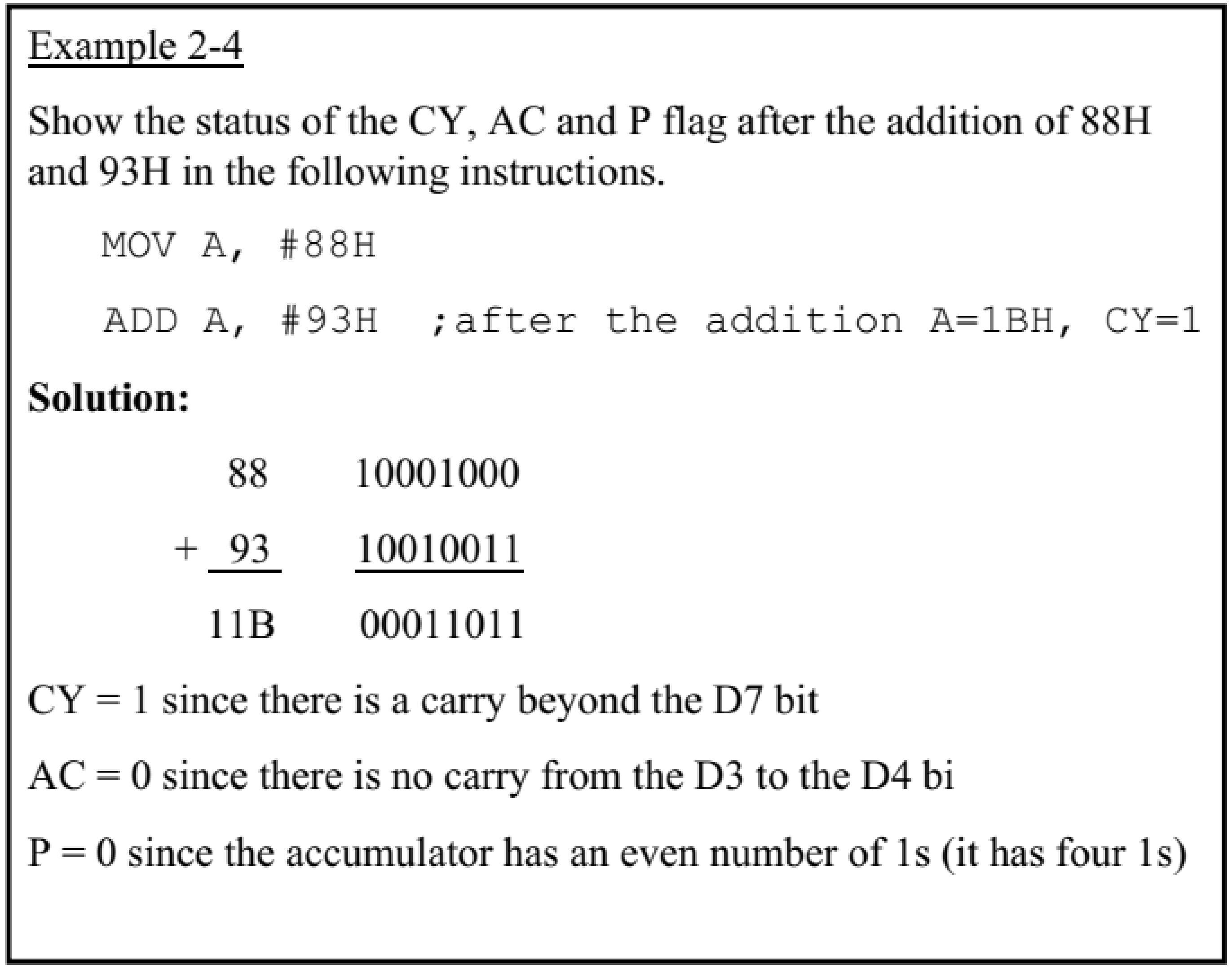
```
          9C       10011100
      +   64       01100100
         100       00000000
```

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

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

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

Flag bits and PSW register



Example 2-4

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

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

Solution:

```
     88      10001000
  +  93      10010011
    11B      00011011
```

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

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

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

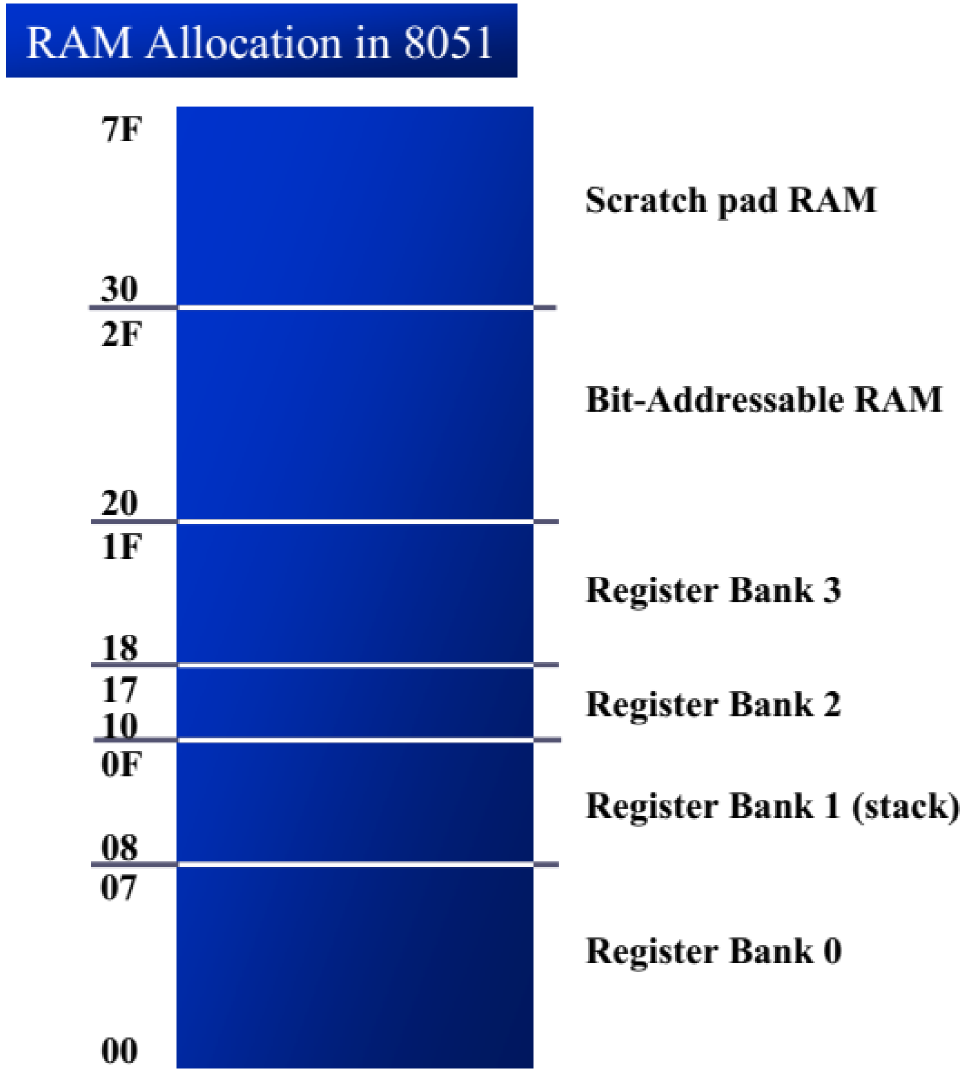
Outline

- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and PSW register
- Register banks and stack
- Hardware connection and Intel HEX file

Register banks and stack

- There are 128 bytes of RAM in the 8051
  - Assigned addresses 00H to 7FH
- The 128 bytes are divided into three 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

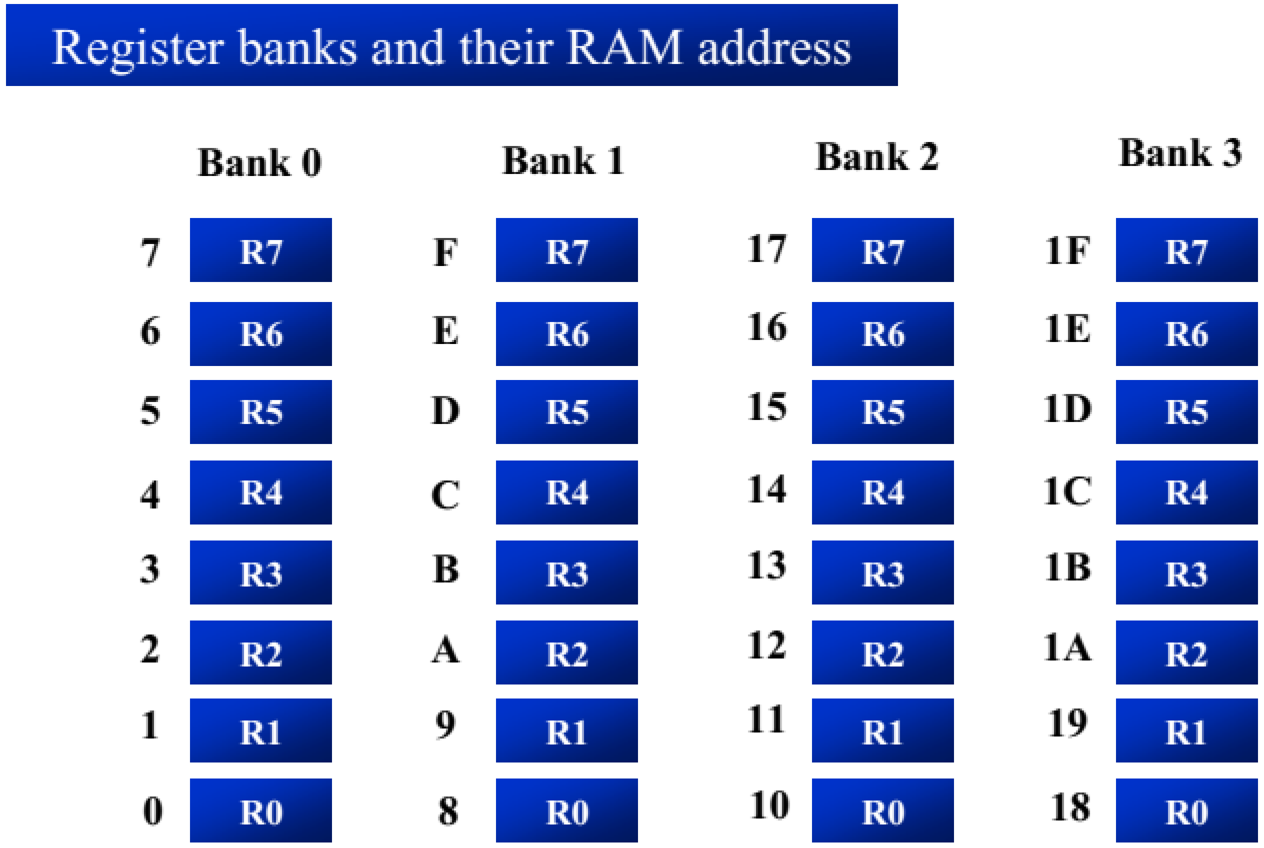
Register banks and stack



Register banks and stack

- These 32 bytes are divided into 4 banks of registers in which each bank has 8 registers, R0-R7
  - RAM location from 0 to 7 are set aside for bank 0 of R0-R7 where R0 is RAM location 0, R1 is RAM location 1, R2 is RAM location 2, and so on, until memory location 7 which belongs to R7 of bank 0
  - It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations
- Register bank 0 is the default when 8051 is powered up

Register banks and stack

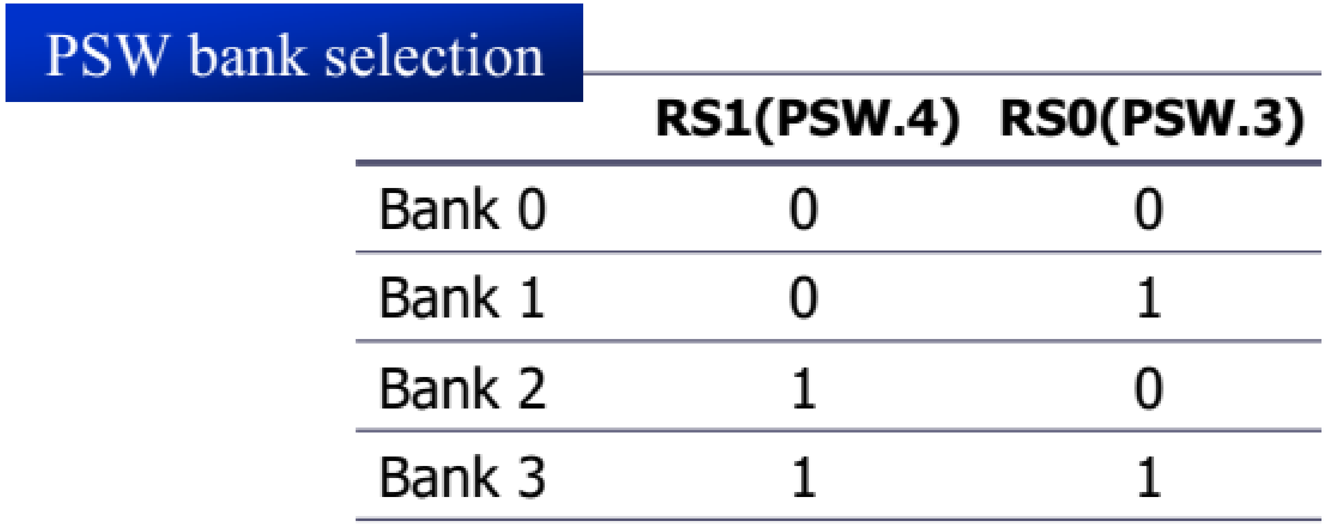


Register banks and their RAM address

	Bank 0		Bank 1		Bank 2		Bank 3
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

Register banks and stack

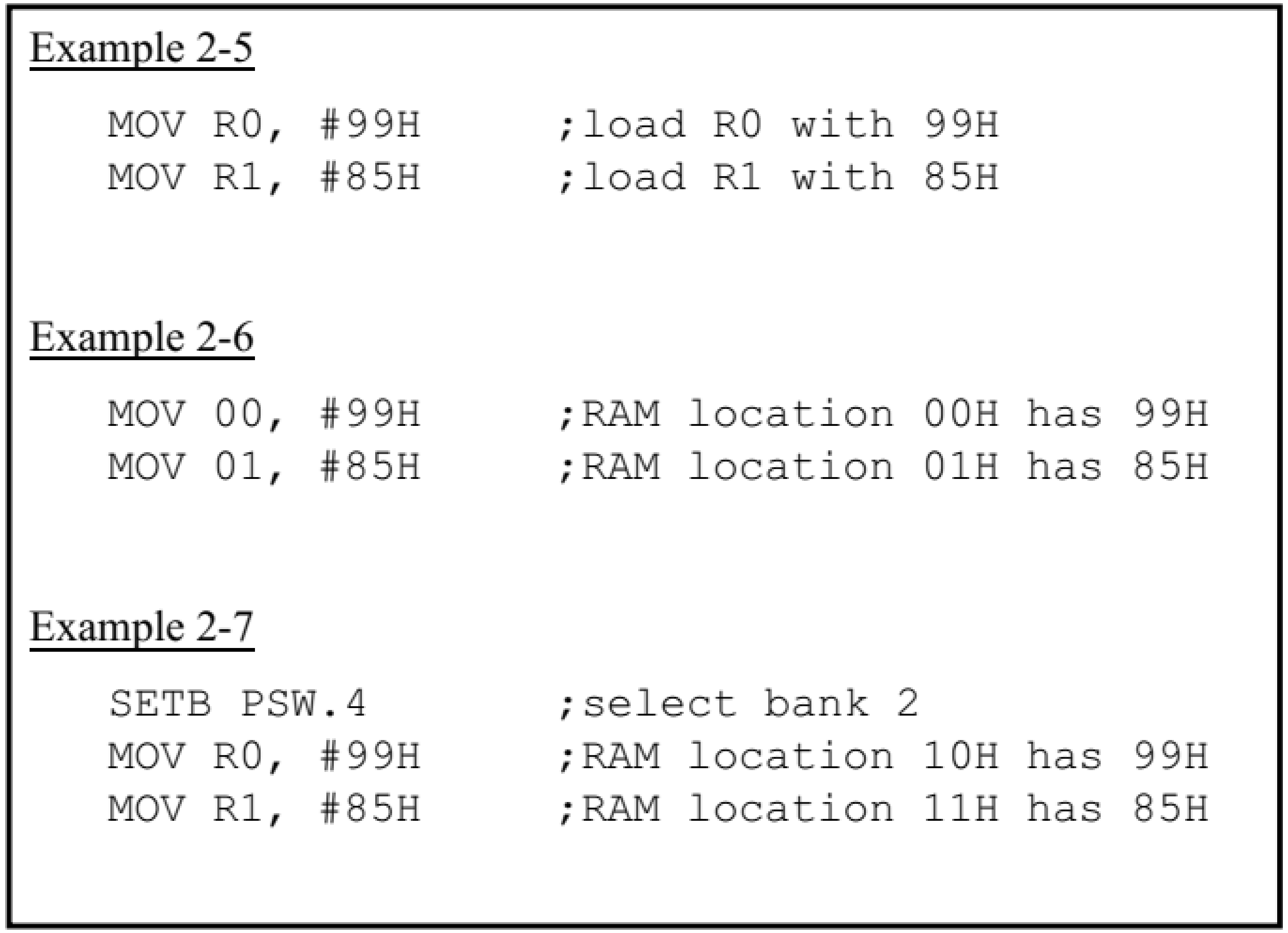
- We can switch to other banks by use of the PSW register
  - Bits D4 and D3 of the PSW are used to select the desired register bank
  - Use the bit-addressable instructions SETB and CLR to access PSW.4 and PSW.3



PSW bank selection

	RS1(PSW.4)	RS0(PSW.3)
Bank 0	0	0
Bank 1	0	1
Bank 2	1	0
Bank 3	1	1

Register banks and stack



Example 2-5

```
MOV R0, #99H        ;load R0 with 99H
MOV R1, #85H        ;load R1 with 85H
```

Example 2-6

```
MOV 00, #99H        ;RAM location 00H has 99H
MOV 01, #85H        ;RAM location 01H has 85H
```

Example 2-7

```
SETB PSW.4          ;select bank 2
MOV R0, #99H        ;RAM location 10H has 99H
MOV R1, #85H        ;RAM location 11H has 85H
```

Register banks and stack

- The **stack** is a section of RAM used by the CPU to store information temporarily

- The register used to access the stack is called the SP (stack pointer) register

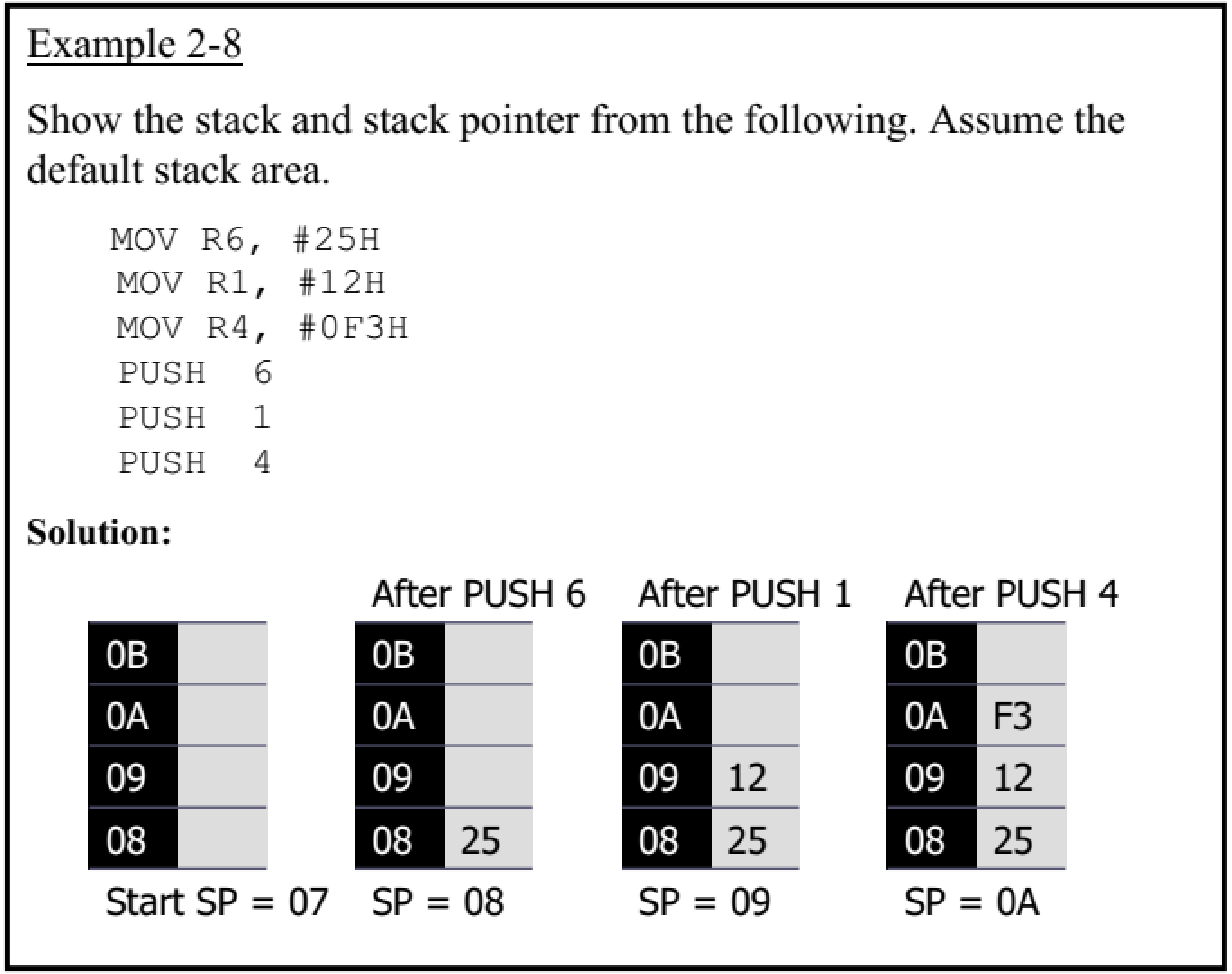
- The stack pointer in the 8051 is only 8 bit wide, which means that it can take value of 00H to FFH

- When the 8051 is powered up, the SP register contains value 07
    - RAM location 08 is the first location being used for the stack by the 8051

Register banks and 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

Register banks and stack



Example 2-8

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

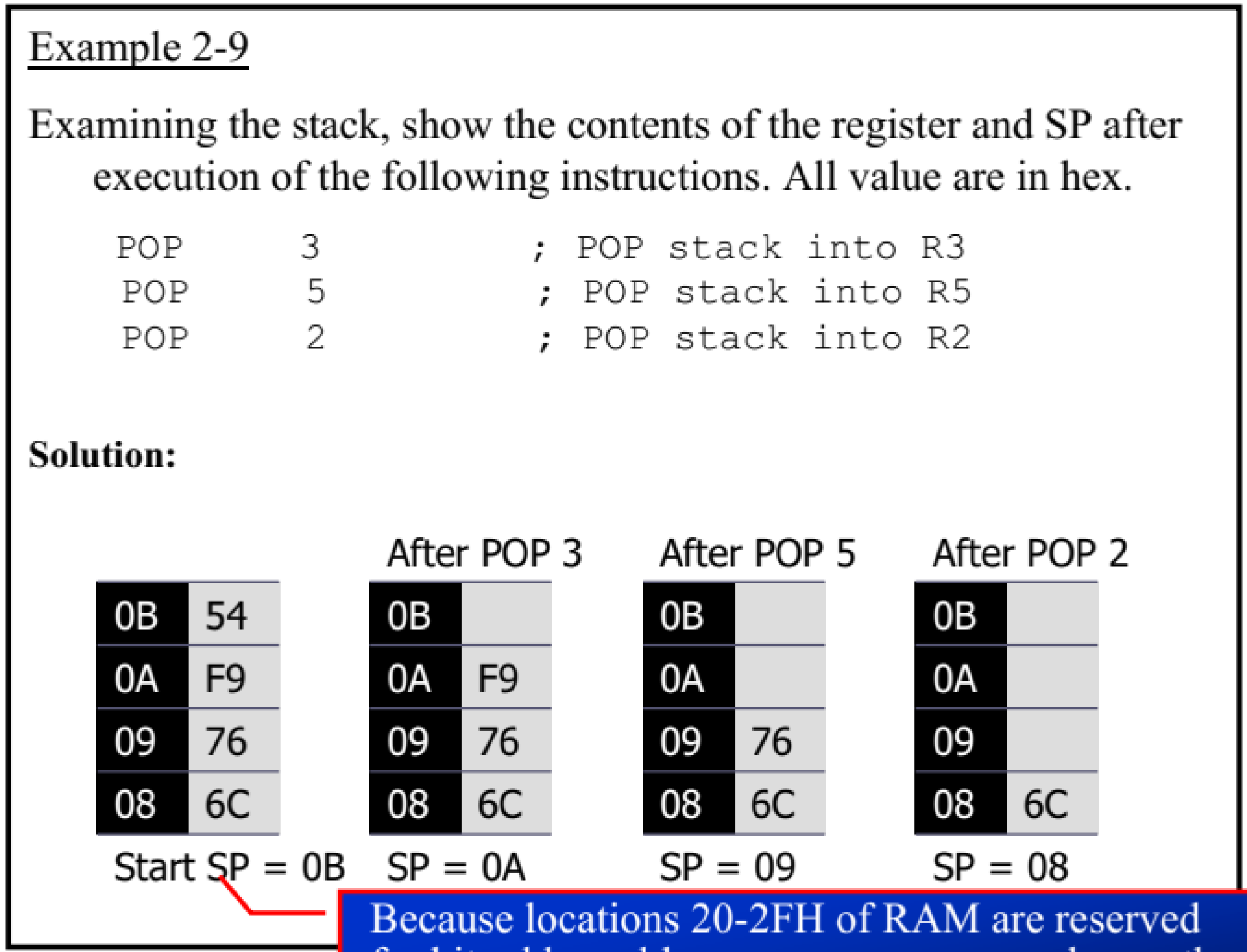
```
MOV R6, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH   6
PUSH   1
PUSH   4
```

Solution:

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

SFU SIMON FRASER UNIVERSITY SURREY

Register banks and stack



Example 2-9

Examining the stack, show the contents of the register and SP after execution of the following instructions. All value 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			
0A	F9	F9		
09	76	76	76	
08	6C	6C	6C	6C
	Start SP = 0B	SP = 0A	SP = 09	SP = 08

Because locations 20-2FH of RAM are reserved for bit-addressable memory, so we can change the SP to other RAM location by using the instruction "`MOV SP, #XX`"

SFU SIMON FRASER UNIVERSITY SURREY

Register banks and stack

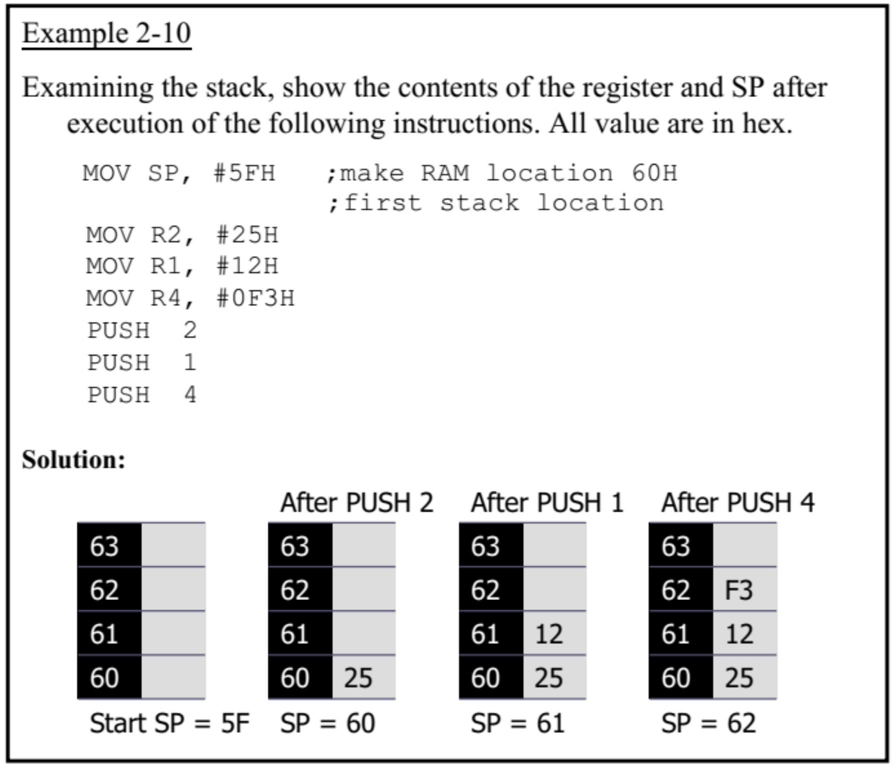
- The CPU also uses the stack to save the address of the instruction just below the CALL instruction
  - This is how the CPU knows where to resume when it returns from the called subroutine

Register banks and stack

- The reason of incrementing SP after push is
  - Make sure that the stack is growing toward RAM location 7FH, from lower to upper addresses
  - Ensure that the stack will not reach the bottom of RAM and consequently run out of stack space
  - If the stack pointer were decremented after push
    - We would be using RAM locations 7, 6, 5, etc. which belong to R7 to R0 of bank 0, the default register bank

Register banks and stack

- When 8051 is powered up, register bank 1 and the stack are using the same memory space
  - We can reallocate another section of RAM to the stack



Example 2-10

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

```
MOV SP, #5FH    ;make RAM location 60H
                ;first stack location
MOV R2, #25H
MOV R1, #12H
MOV R4, #0F3H
PUSH  2
PUSH  1
PUSH  4
```

Solution:

After PUSH 2
After PUSH 1
After PUSH 4
63
62
61
60
25
12
F3
Start SP = 5F
SP = 60
SP = 61
SP = 62

Outline

- Inside the 8051- registers and MOV & ADD instructions
- 8051 assembly programming
- Program counter and ROM space
- 8051 data types and directives
- Flag bits and PSW register
- Register banks and stack
- Hardware connection and Intel HEX file

Hardware connection and Intel HEX file

- 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

Hardware connection and Intel HEX file

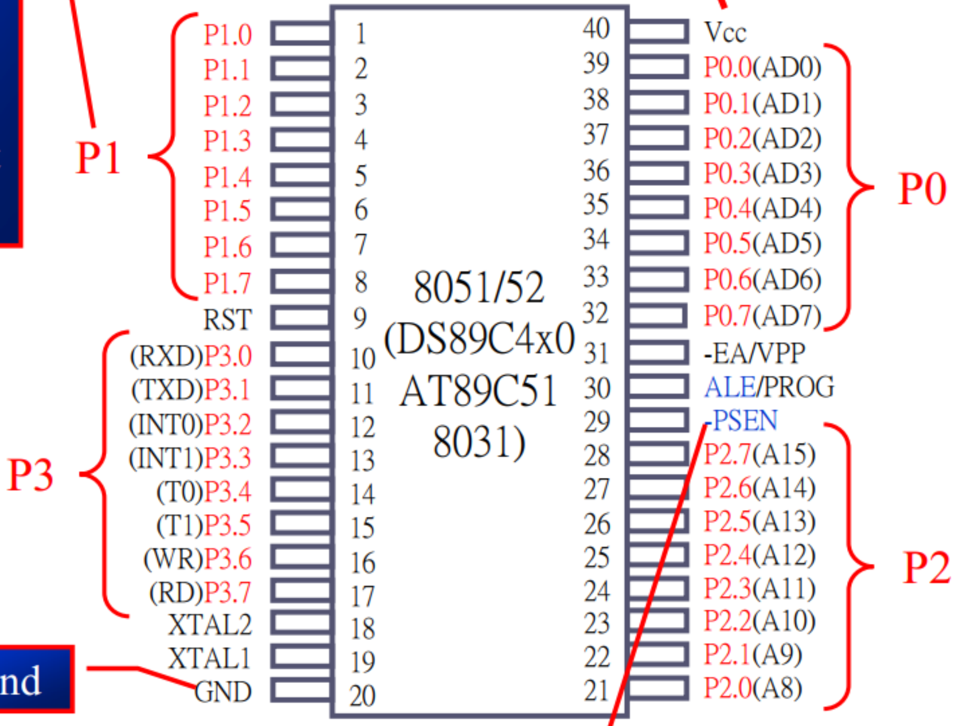
A total of 32 pins are set aside for the four ports P0, P1, P2, P3, where each port takes 8 pins

Provides +5V supply voltage to the chip

Vcc, GND, XTAL1, XTAL2, RST, -EA are used by all members of 8051 and 8031 families

Grond

-PSEN and ALE are used mainly in 8031-baded systems

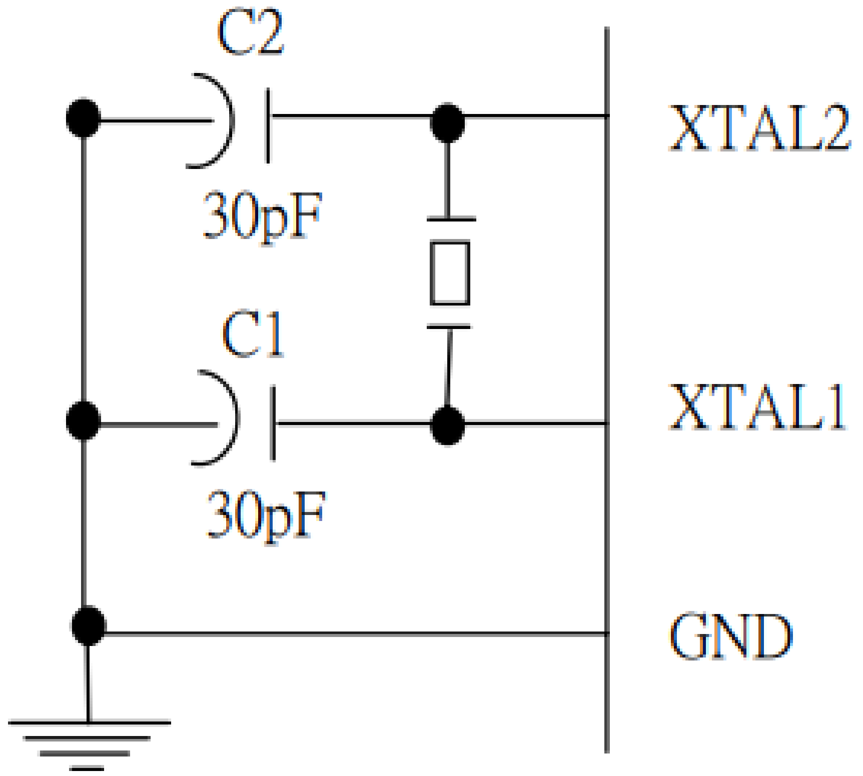


Hardware connection and Intel HEX file

- The 8051 has an on-chip oscillator but requires an external clock to run it

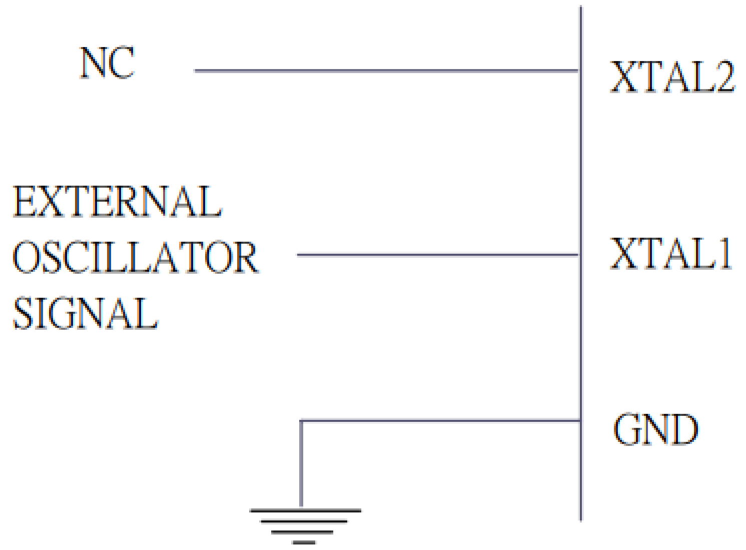
- A quartz crystal oscillator is connected to inputs XTAL1 (pin19) and XTAL2 (pin18)

- The quartz crystal oscillator also needs two capacitors of 30 pF value



Hardware connection and Intel HEX file

- 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

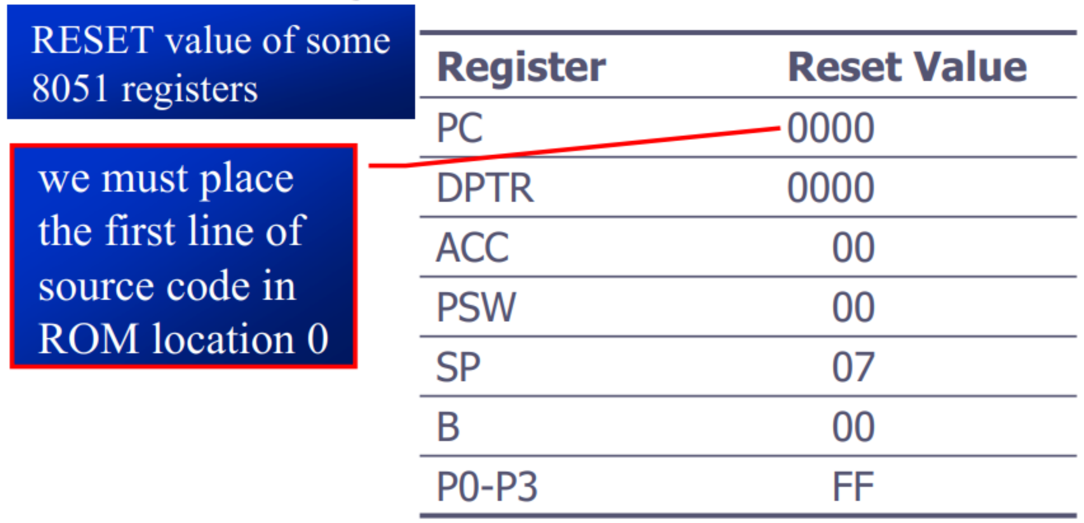


Hardware connection and Intel HEX file

- The speed of 8051 refers to the maximum oscillator frequency connected to XTAL
  - ex. A 12-MHz chip must be connected to a crystal with 12 MHz frequency or less
  - We can observe the frequency on the XTAL2 pin using the oscilloscope

Hardware connection and Intel HEX file

- 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
    - Upon applying a high pulse to this pin, the microcontroller will reset and terminate all activities
    - Activating a power-on reset will cause all values in the registers to be lost



RESET value of some 8051 registers

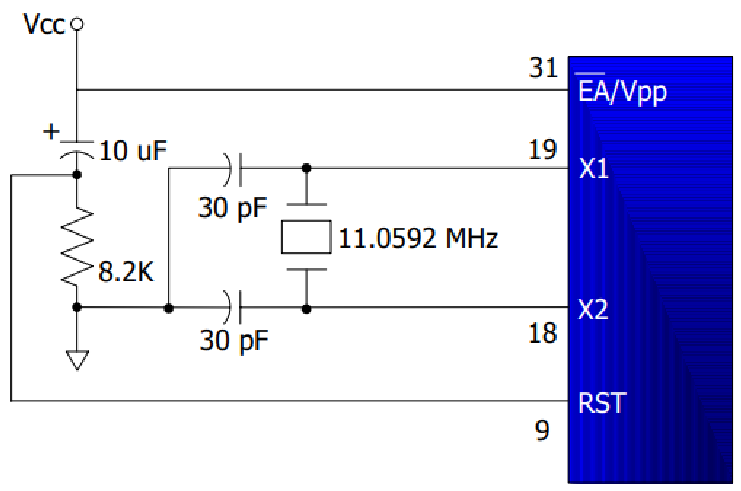
we must place the first line of source code in ROM location 0

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

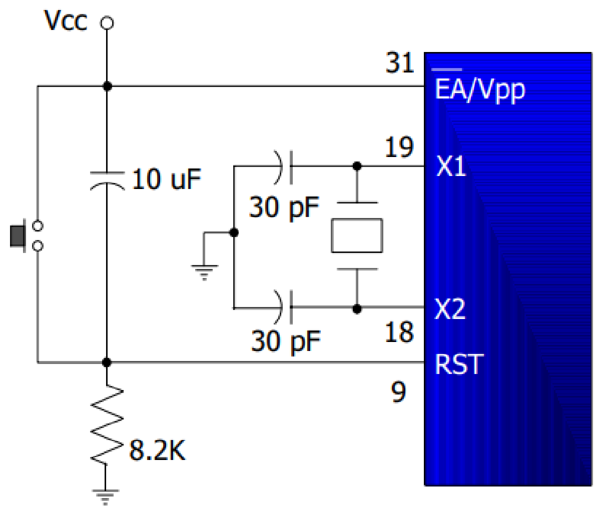
Hardware connection and Intel HEX file

- In order for the RESET input to be effective, it must have a minimum duration of 2 machine cycles
  - In other words, the high pulse must be high for a minimum of 2 machine cycles before it is allowed to go low

Power-on RESET circuit



Power-on RESET with debounce



SFU SIMON FRASER UNIVERSITY SURREY

Hardware connection and Intel HEX file

- EA, “external access”, is an input pin and must be connected to Vcc or GND
    - EA pin is connected to Vcc
    - The 8031 and 8032 family members do no have on-chip ROM, so code is stored on an external ROM and is fetched by 8031/32
        - EA pin must be connected to GND to indicate that the code is stored externally

Hardware connection and Intel HEX file

- The following two pins are used mainly in 8031-based systems
- PSEN, “program store enable”, is an output pin
  - This pin is connected to the OE pin of the ROM
- ALE, “address latch enable”, is an output pin and is active high
  - Port 0 provides both address and data
    - The 8031 multiplexes address and data through port 0 to save pins
    - ALE pin is used for demultiplexing the address and data by connecting to the G pin of the 74LS373 chip

Hardware connection and Intel HEX file

- Intel hex file is a widely used file format
  - Designed to standardize the loading of executable machine codes into a ROM chip
- Loaders that come with every ROM burner (programmer) support the Intel hex file format
  - In many newer Windows-based assemblers the Intel hex file is produced automatically (by selecting the right setting)
  - In DOS-based PC you need a utility called OH (object-to-hex) to produce that

Hardware connection and Intel HEX file

- In the DOS environment
  - The object file is fed into the linker program to produce the abs file
  - The abs file is used by systems that have a monitor program
  - Then the abs file is fed into the OH utility to create the Intel hex file
  - The hex file is used only by the loader of an EPROM programmer to load it into the ROM chip

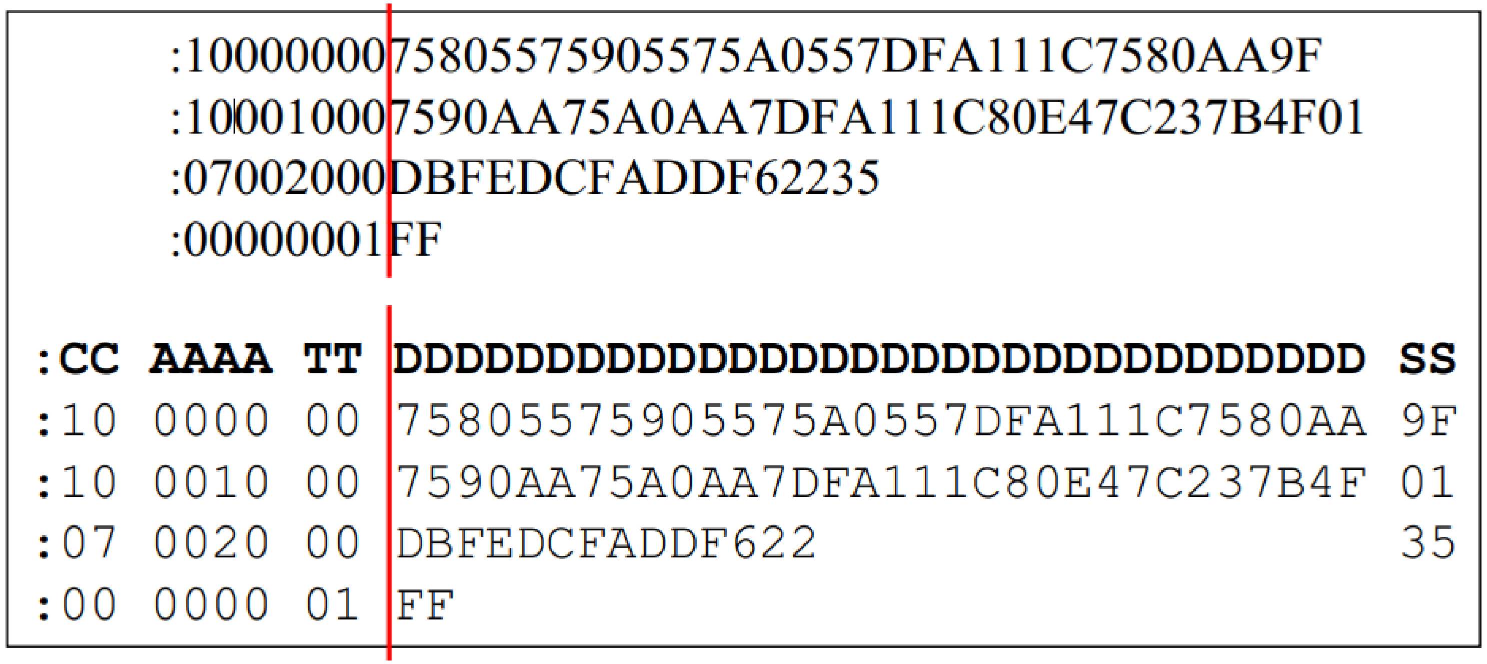
Hardware connection and Intel HEX file

The location is the address where the opcodes (object codes) are placed

```
LOC  OBJ     LINE
0000         1              ORG 0H
0000 758055  2   MAIN:      MOV P0,#55H
0003 759055  3              MOV P1,#55H
0006 75A055  4              MOV P2,#55H
0009 7DFA    5              MOV R5,#250
000B 111C    6              ACALL MSDELAY
000D 7580AA  7              MOV P0,#0AAH
0010 7590AA  8              MOV P1,#0AAH
0013 75A0AA  9              MOV P2,#0AAH
0016 7DFA    10             MOV R5,#250
0018 111C    11             ACALL MSDELAY
001A 80E4    12             SJMP MAIN
             13 ;--- THE 250 MILLISECOND DELAY.
             14 MSDELAY:
001C 7C23    15 HERE3:      MOV R4,#35
001E 7B4F    16 HERE2:      MOV R3,#79
0020 DBFE    17 HERE1:      DJNZ R3,HERE1
0022 DCFA    18             DJNZ R4,HERE2
0024 DDF6    19             DJNZ R5,HERE3
0026 22      20             RET
             21             END
```

Hardware connection and Intel HEX file

- The hex file provides the following:
  - The number of bytes of information to be loaded
  - The information itself
  - The starting address where the information must be placed



```
:100000007580557590557 5A0557DFA111C7580AA9F
:100010007590AA75A0AA7DFA111C80E47C237B4F01
:07002000DBFEDCFADDF62235
:00000001FF

:CC AAAA TT DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD SS
:10 0000 00 75805575905575A0557DFA111C7580AA 9F
:10 0010 00 7590AA75A0AA7DFA111C80E47C237B4F 01
:07 0020 00 DBFEDCFADDF622                   35
:00 0000 01 FF
```

Hardware connection and Intel HEX file

Each line starts with a colon

Count byte – how many bytes, 00 to 16, are in the line

16-bit address – The loader places the first byte of data into this memory address

Type –

00, there are more lines to come after this line

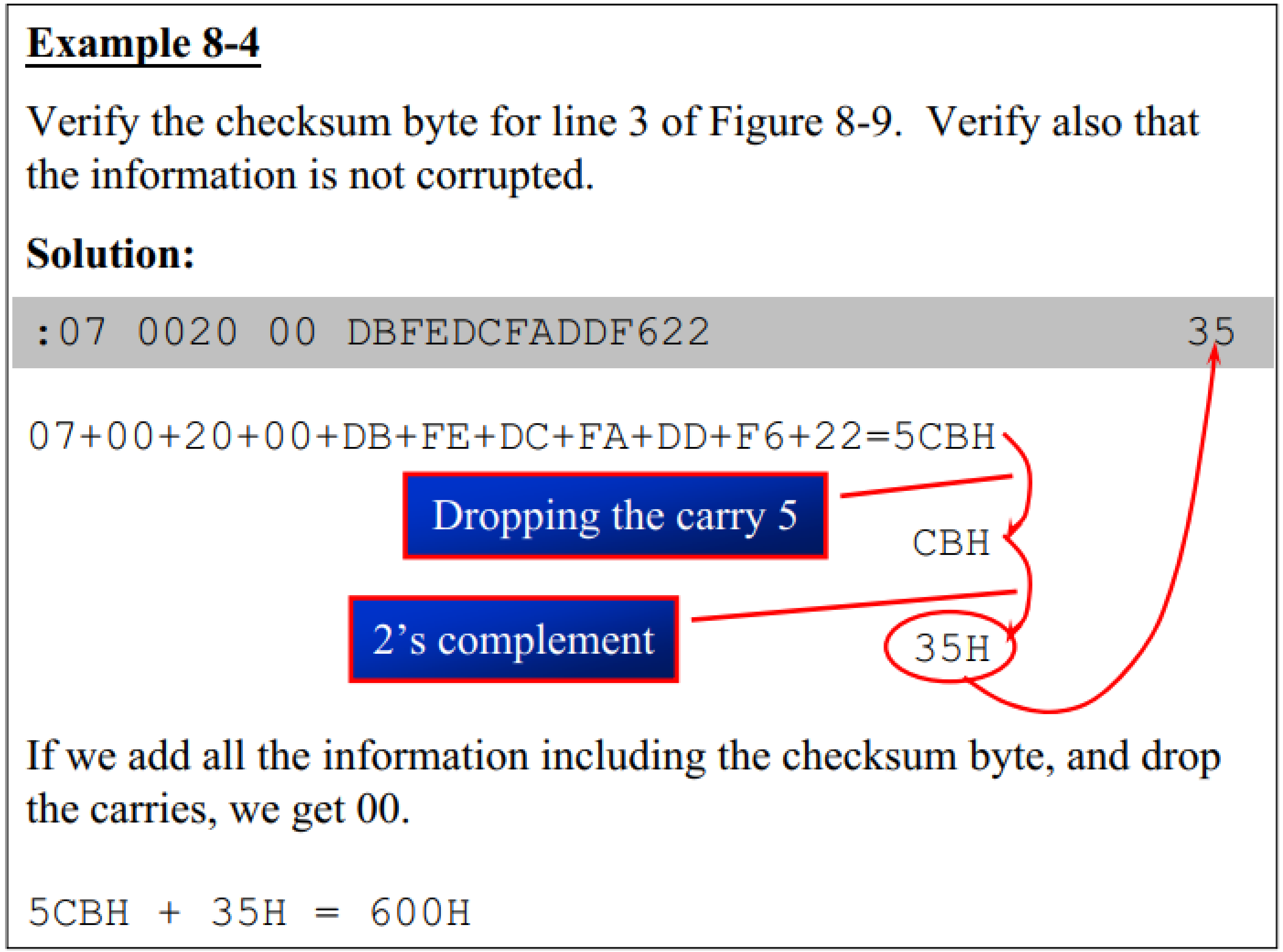
01, this is the last line and the loading should stop after this line

```
:CC AAAA TT DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD SS
:10 0000 00 75805575905575A0557DFA111C7580AA 9F
:10 0010 00 7590AA75A0AA7DFA111C80E47C237B4F 01
:07 0020 00 DBFEDCFADDF622                   35
:00 0000 01 FF
```

Real information (data or code) – There is a maximum of 16 bytes in this part. The loader places this information into successive memory locations of ROM

Single byte – this last byte is the checksum byte of everything in that line

Hardware connection and Intel HEX file



Example 8-4

Verify the checksum byte for line 3 of Figure 8-9. Verify also that the information is not corrupted.

Solution:

```
:07 0020 00 DBFEDCFADDF622                  35
```

```
07+00+20+00+DB+FE+DC+FA+DD+F6+22=5CBH
```

Dropping the carry 5

```
CBH
```

2's complement

```
35H
```

If we add all the information including the checksum byte, and drop the carries, we get 00.

```
5CBH + 35H = 600H
```