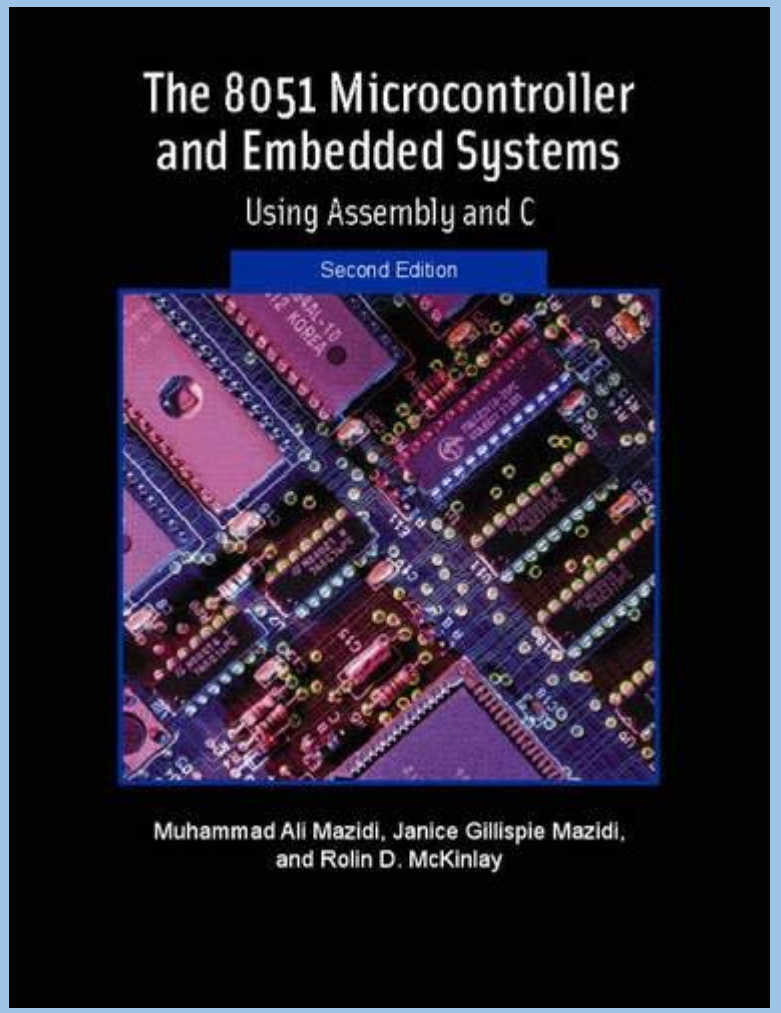
THE 8051 MICROCONTROLLER & Embedded Systems

Muhammad Ali Mazidi, Janice Mazidi
& Rolin McKinlay



8051 ASSEMBLY LANGUAGE PROGRAMMING

Chapter 2

Objectives

Upon completion of this chapter, you will be able to:

- >> List the registers of the 8051 microcontroller
- >> Manipulate data using the registers and MOV instructions
- >> Code simple 8051 Assembly language instructions
- >> Assemble and run an 8051 program
- >> Describe the sequence of events that occur upon 8051 power-up
- >> Examine programs in ROM code of the 8051
- >> Explain the ROM memory map of the 8051
- >> Detail the execution of 8051 Assembly language instructions
- >> Describe 8051 data types
- >> Explain the purpose of the PSW (program status word) register
- >> Discuss RAM memory space allocation in the 8051
- >> Diagram the use of the stack in the 8051
- >> Manipulate the register banks of the 8051
- >> Understand the RISC and CISC architectures

Objectives

- Section 2.1, Looking at the inside of the 8051 and demonstrating some of the widely used registers of the 8051 such as MOV and ADD,
- Section 2.2, examining Assembly language and machine language programming and defining terms such as mnemonics, opcode, operand, etc.,
- Section 2.3, step-by-step execution of an 8051 program,
- Section 2.4, examining the role of the program counter,
- Section 2.5, looking at some widely used Assembly language directives, pseudocode, and data types related to the 8051
- Section 2.6, discussing the flag bits and how they are affected by arithmetic instructions
- Section 2.7, allocation of RAM memory inside the 8051 plus the stack and register banks of the 8051
- Section 2.8, the concepts of RISC and CISC architectures

Inside the 8051

The vast majority of 8051 registers are 8-bit registers.

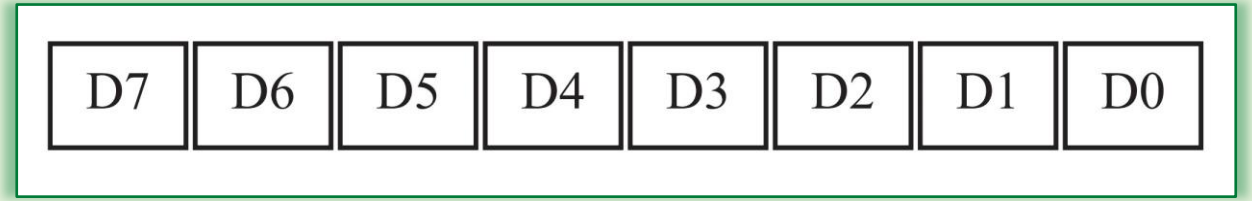


Figure 2-1 (b). Some 8051 16-bit Registers

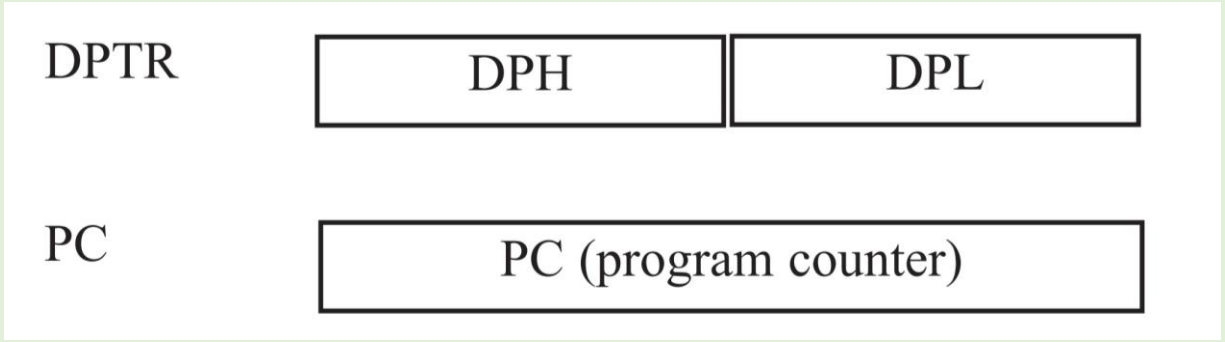
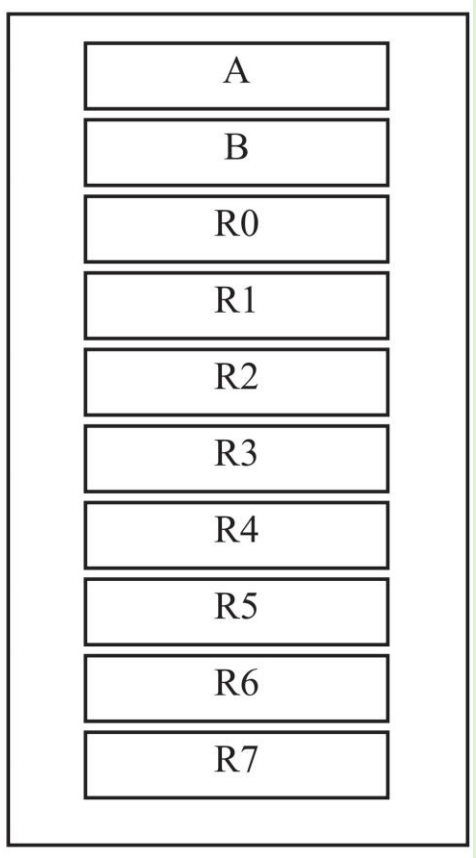


Figure 2-1 (a). Some 8-bit Registers of the 8051



MOV Instruction

Simply stated,
the MOV instruction copies data from one location to another.

MOV Instruction Format:

```
MOV destination, source ;copy source to dest
```

This instruction tells the CPU
to move (in reality, copy) the source operand to the destination operand.

ADD Instruction

ADD Instruction Format:

```
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.

Structure of Assembly Language

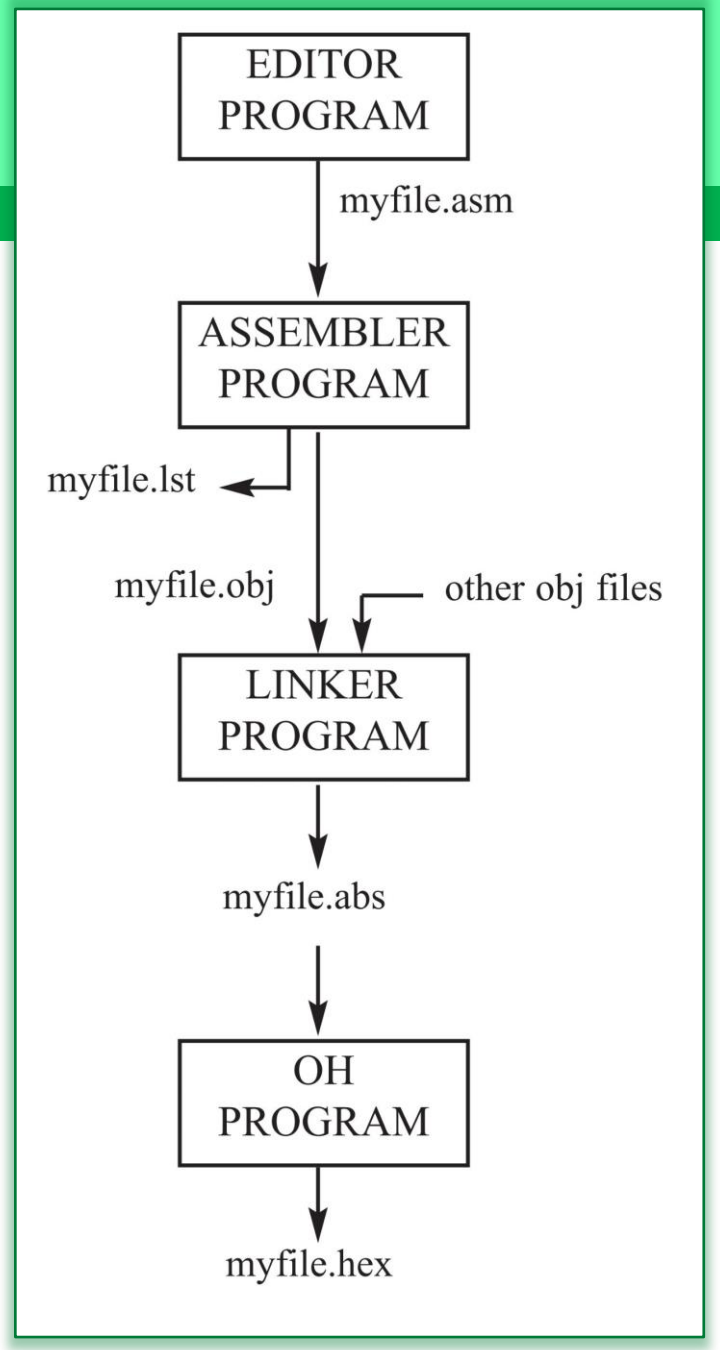
An Assembly language instruction consists of four fields:

```
[label:]    mnemonic    [operands]    [;comment]
```

Brackets indicate that a field is optional, and not all lines have them.
Brackets should not be typed in.

Assembling and Running an 8051 Program

Figure 2-2. Steps to Create a Program



The “asm” file is also called the source file and for this reason some assemblers require that this file have the “src” extension.

Placing Code in Program ROM

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	80FE	HERE: SJMP HERE

Note: 7D opcode, 25 operand ; 7D opcode to move value to register R5

Placing Code in Program ROM

Program 2-1: 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

Example 2-1

Find the ROM memory address of each of the following 8051 chips.
(a)AT89C51 with 4KB (b) DS89C420 with 16KB (c) DS5000-32 with 32KB

Solution:

(a) With 4K bytes of on-chip ROM memory space, we have 4096 bytes (4x1024 = 4096). This maps to address locations of 0000 to 0FFFH. Notice that 0 is always the first location.
(b) (b) With 16K bytes of on-chip ROM memory space, we have 16,384 bytes (16 x 1024 = 16,384), which gives 0000 - 3FFFH.
(c) (c) With 32K bytes we have 32,768 bytes (32 x 1024 = 32,768). Converting 32,768 to hex, we get 8000H; therefore, the memory space is 0000 to 7FFFH.

ROM Memory Map in the 8051 Family

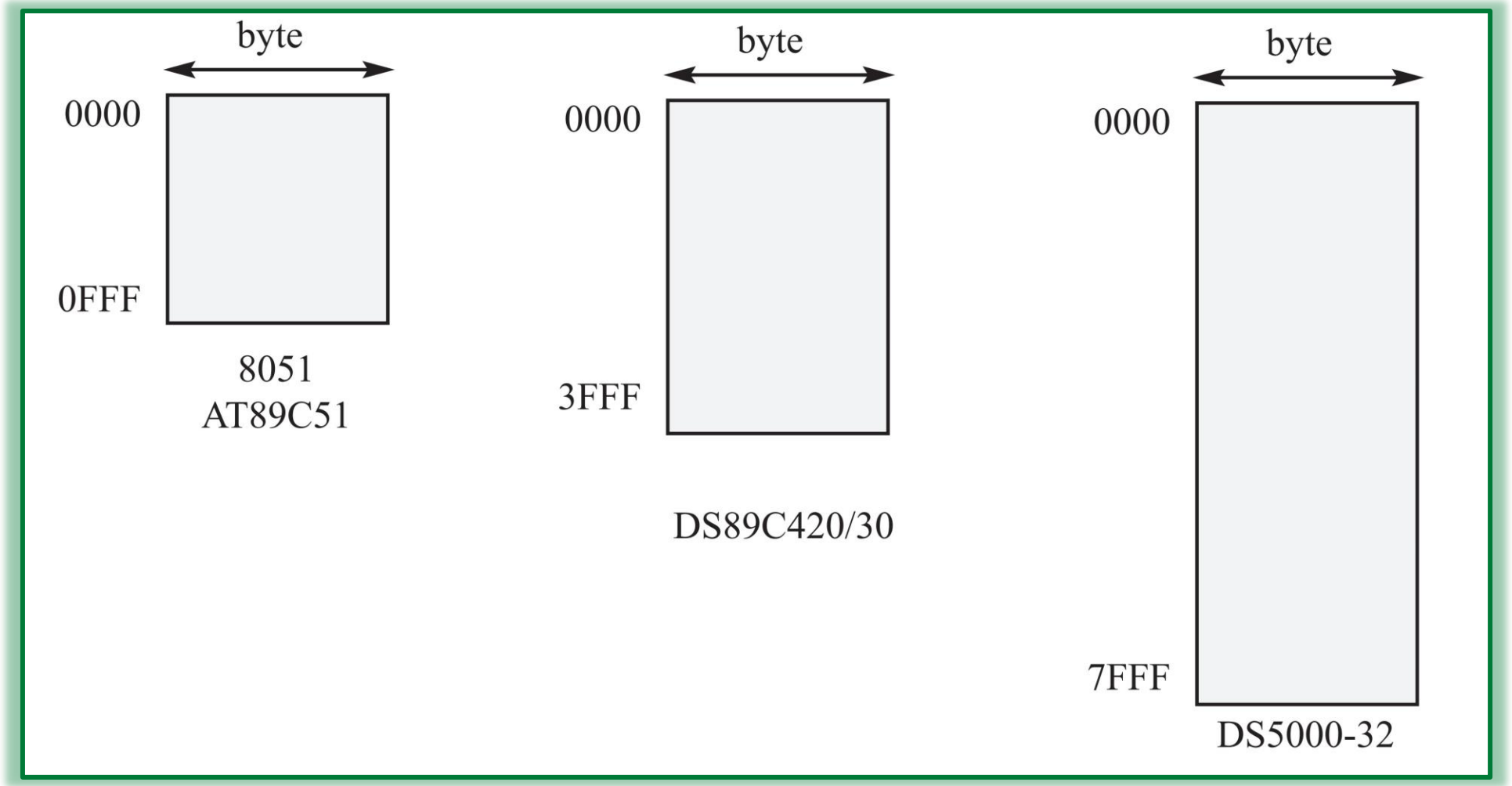


Figure 2-3. 8051 On-Chip ROM Address Range

PSW (Program Status Word) Register

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

CY	AC	F0	RS1	RS0	OV	--	P

CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary carry flag.
F0	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 instuction cycle to indicate an odd/even number of 1 bits in the accumulator.

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

Figure 2-4. Bits of the PSW Register

Table 2-1: 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		
RRC	X		
RLC	X		
SETB C	1		

Instruction	CY	OV	AC
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		

Note: X can be 0 or 1.

Example 2-2

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

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

Solution:

38	00111000
+ 2F	00101111
67	01100111

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

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

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

Example 2-3

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

```
MOV A,#9CH
ADD A,#64H   ;after addition A=00 and CY=1
```

Solution:

9C	10011100
+ 64	01100100
100	00000000

CY = 1 since there is a carry beyond the D7 bit.
AC = 1 since there is a carry from the D3 to the D4 bit.
P = 0 since the accumulator has an even number of 1s (it has zero 1s).

Example 2-4

Show the status of the CY, AC, and P flags 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 bit.
P = 0 since the accumulator has an even number of 1s (it has four 1s).

8051 Register Banks and Stack

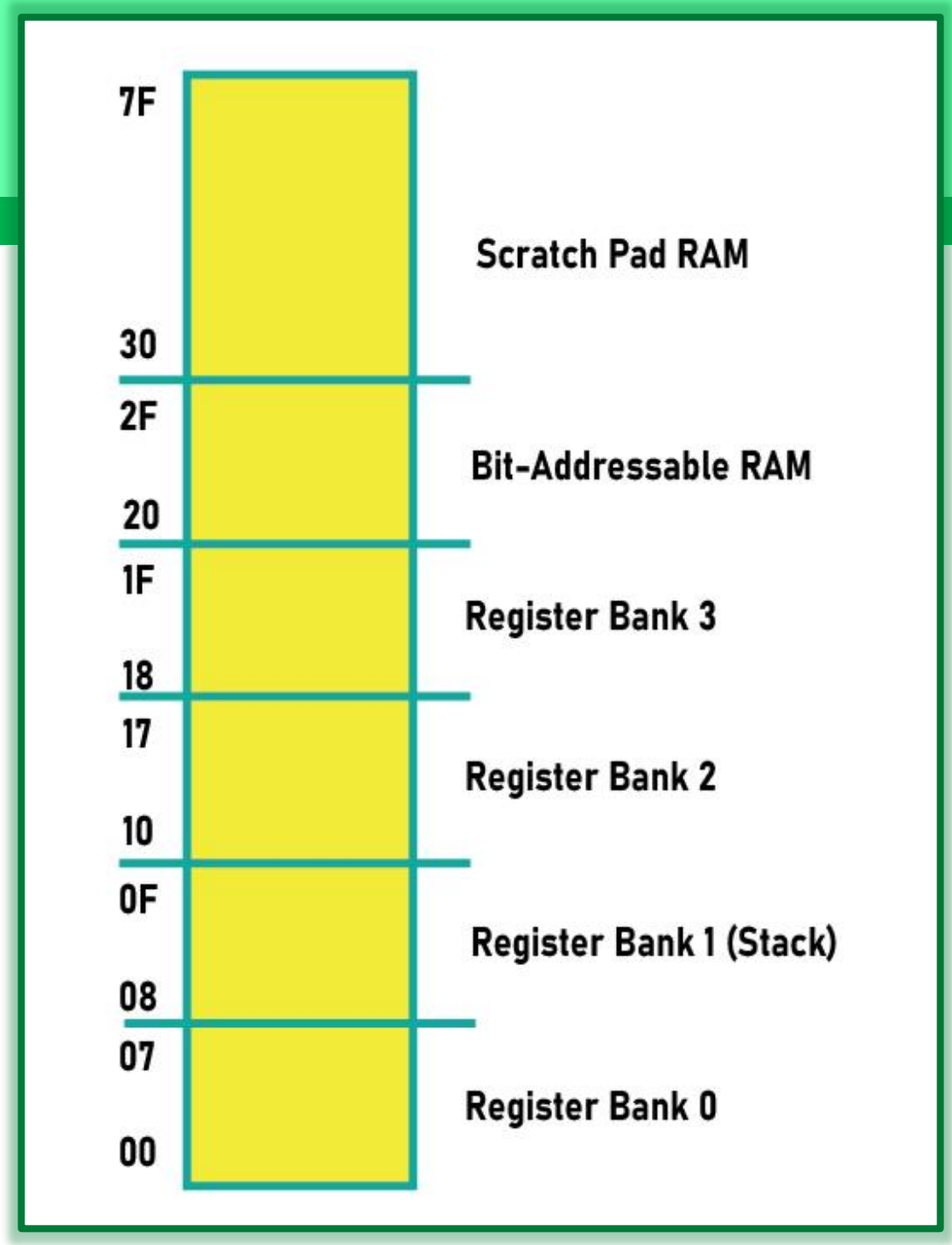


Figure 2-5. RAM Allocation in the 8051

8051 Register Banks and Stack

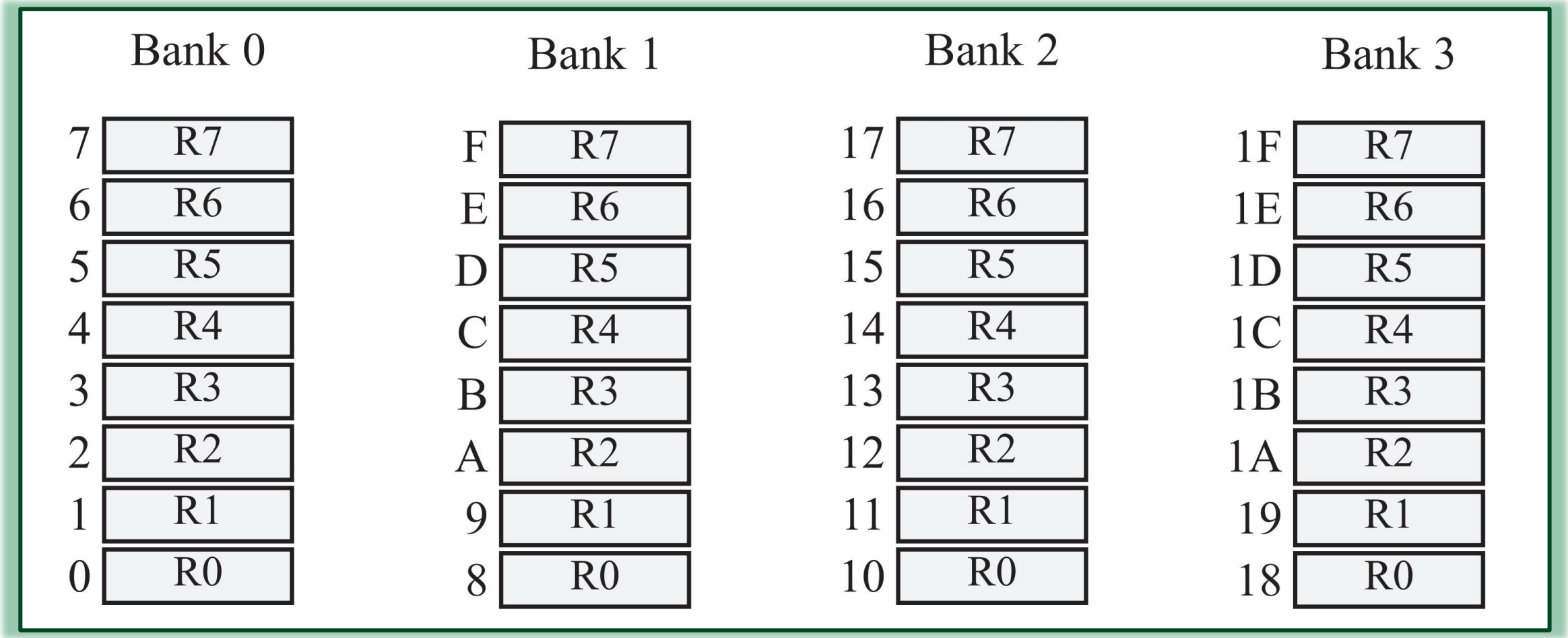


Figure 2-6. 8051 Register Banks and their RAM Addresses

Example 2-5

State the contents of RAM locations after the following program:

```
MOV R0,#99H    ;load R0 with value 99H
MOV R1,#85H    ;load R1 with value 85H
MOV R2,#3FH    ;load R2 with value 3FH
MOV R7,#63H    ;load R7 with value 63H
MOV R5,#12H    ;load R5 with value 12H
```

Solution:

After the execution of the above program we have the following:

RAM location 0 has value 99H
RAM location 1 has value 85H
RAM location 2 has value 3FH
RAM location 7 has value 63H
RAM location 5 has value 12H

Example 2-6: Default Register Bank

Repeat Example 2-5 using RAM addresses instead of register names.

Solution:

This is called direct addressing mode and uses the RAM address location for the destination address.

```
MOV 00,#99H     ;load R0 with value 99H
MOV 01,#85H     ;load R1 with value 85H
MOV 02,#3FH     ;load R2 with value 3FH
MOV 07,#63H     ;load R7 with value 63H
MOV 05,#12H     ;load R5 with value 12H
```

How to Switch Register Banks

Table 2-2: PSW Bits Bank Selection

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

Example 2-7

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

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

Solution:

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

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

Stack in 8051

- Register used to access the stack is called stack pointer (SP).
- SP is 8bit wide and can take values of 00 to FFH.
- When 8051 is powered up, SP register contains value 07.
- Ram location 08 is the first locations used by stack
- Storing of CPU register in the stack is called a PUSH, and pulling the contents off the stack back into CPU register is called POP.

Pushing onto the stack:

- SP points to the last used location of the stack.
- For every byte of data saved on the stack, SP is incremented only once.
- To push registers on the stack, their RAM addresses must be used.

Popping from the stack:

- With every pop, the top byte of the stack is copied to the register specified by the instruction
- Stack pointer is decremented by one with every pop.

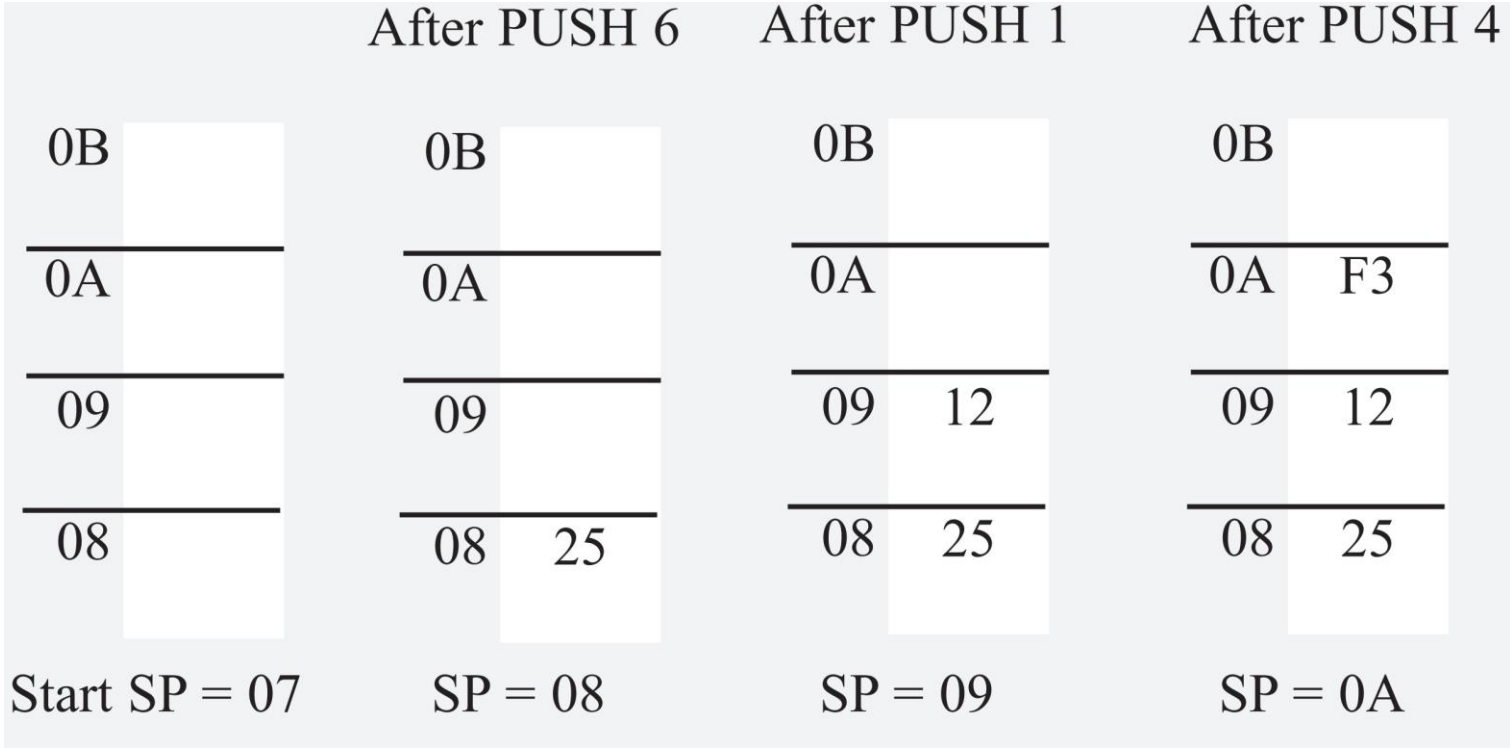
Example 2-8: Pushing onto the Stack

Show the stack and stack pointer for the following. Assume the default stack area and register 0 is selected.

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

Note: "PUSH 6" pushed register R6 onto the stack

Solution:

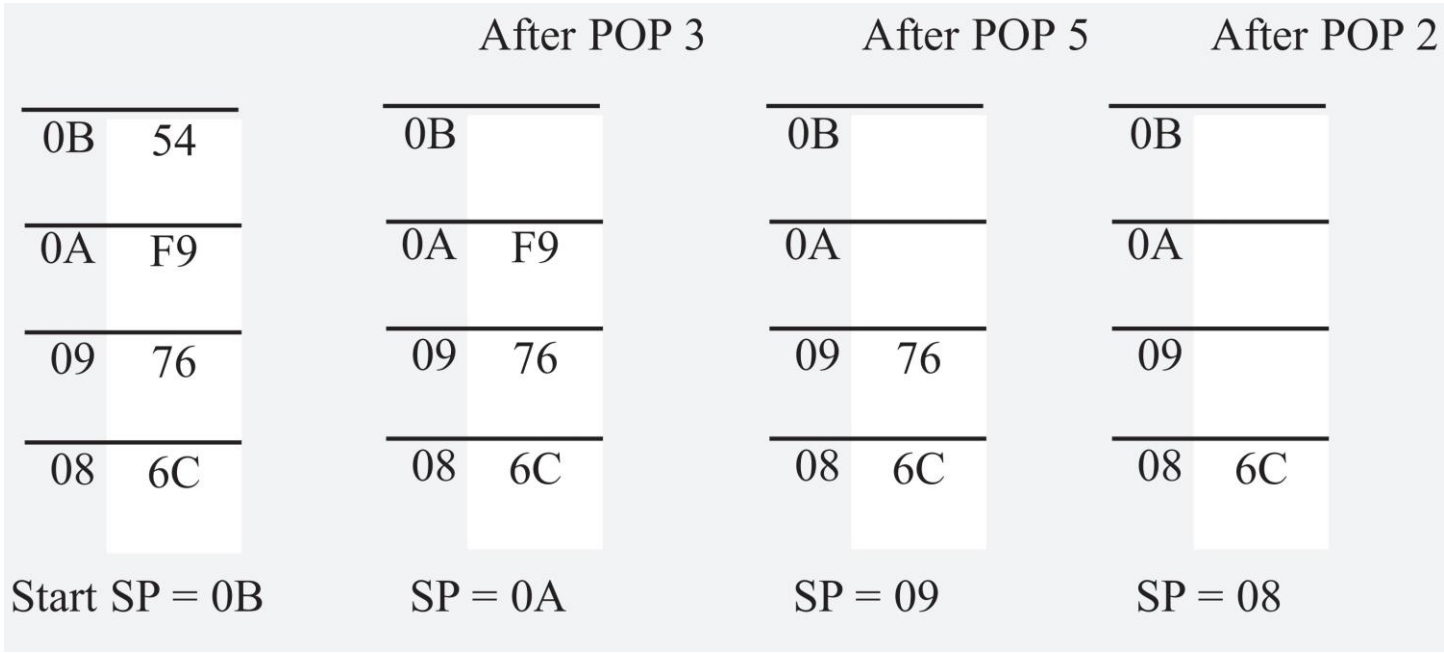


Example 2-9: Popping from the Stack

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

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

Solution:

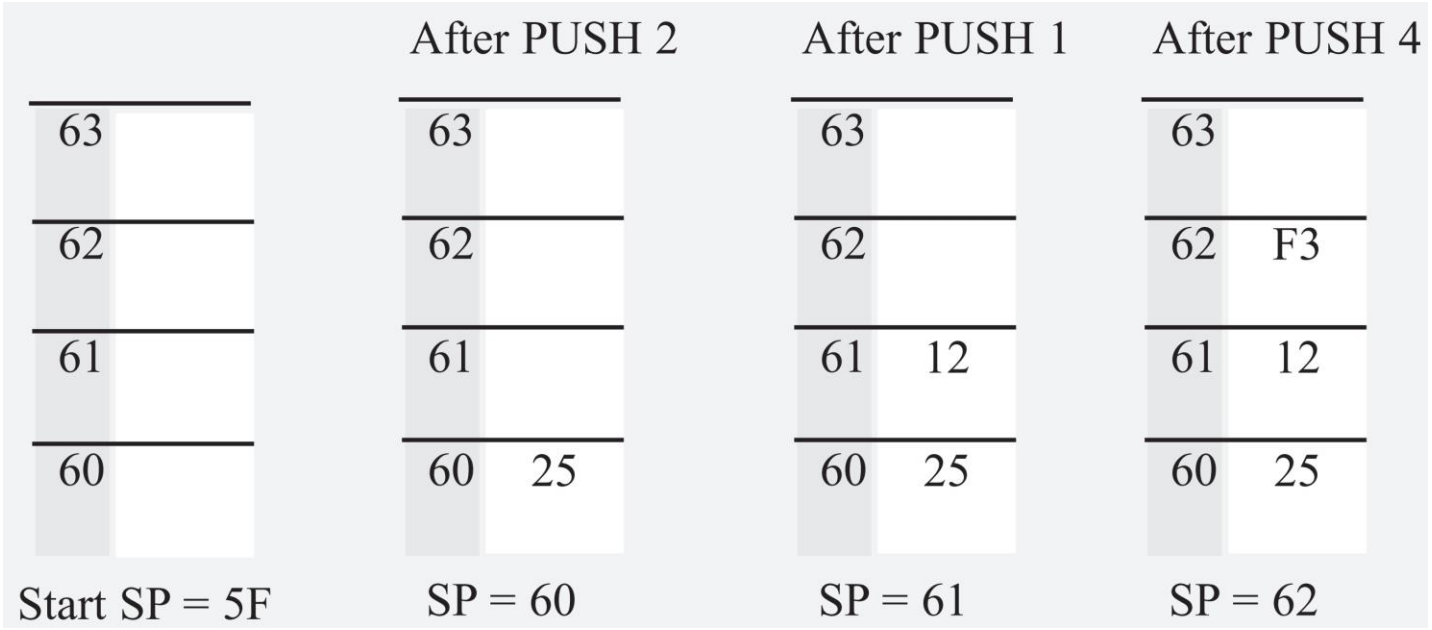


Example 2-10: Stack and Bank 1 Conflict

Show the stack and stack pointer for the following instructions.

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

Solution:



Viewing Registers and Memory with a Simulator

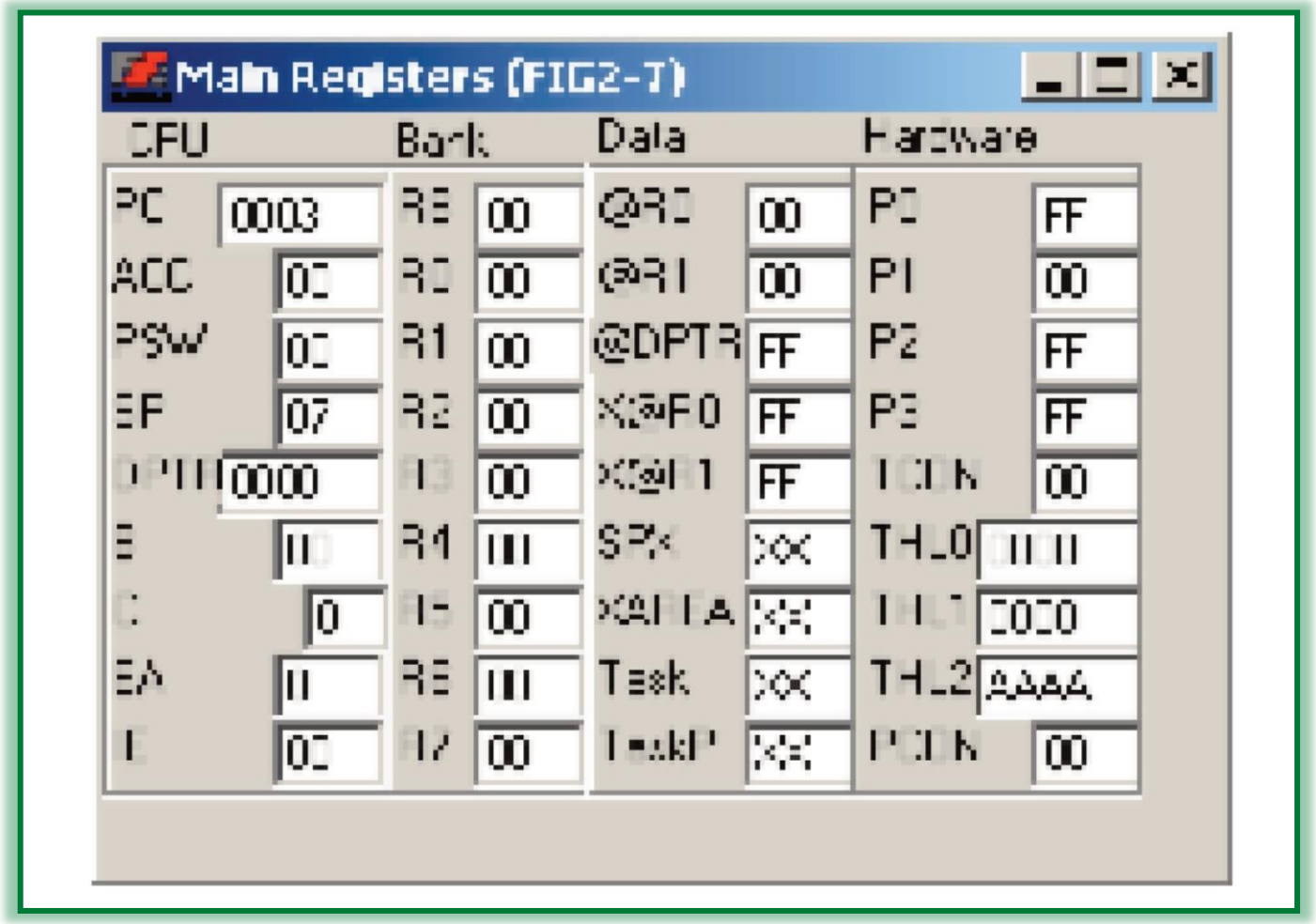


Figure 2-7. Register's Screen from ProView 32 Simulator

Viewing Registers and Memory with a Simulator

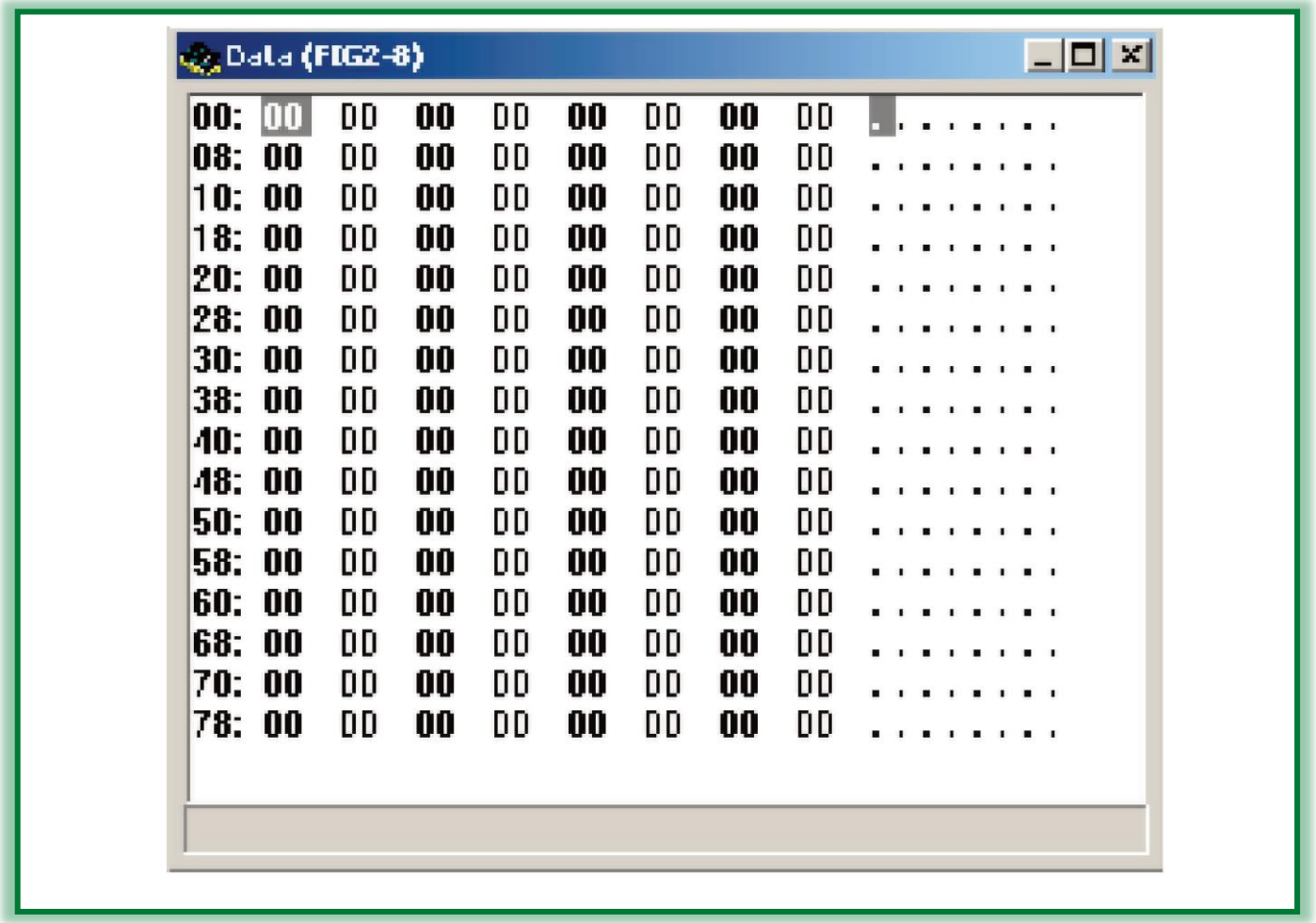


Figure 2-8. 128-Byte Memory Space from ProView 32 Simulator

Viewing Registers and Memory with a Simulator

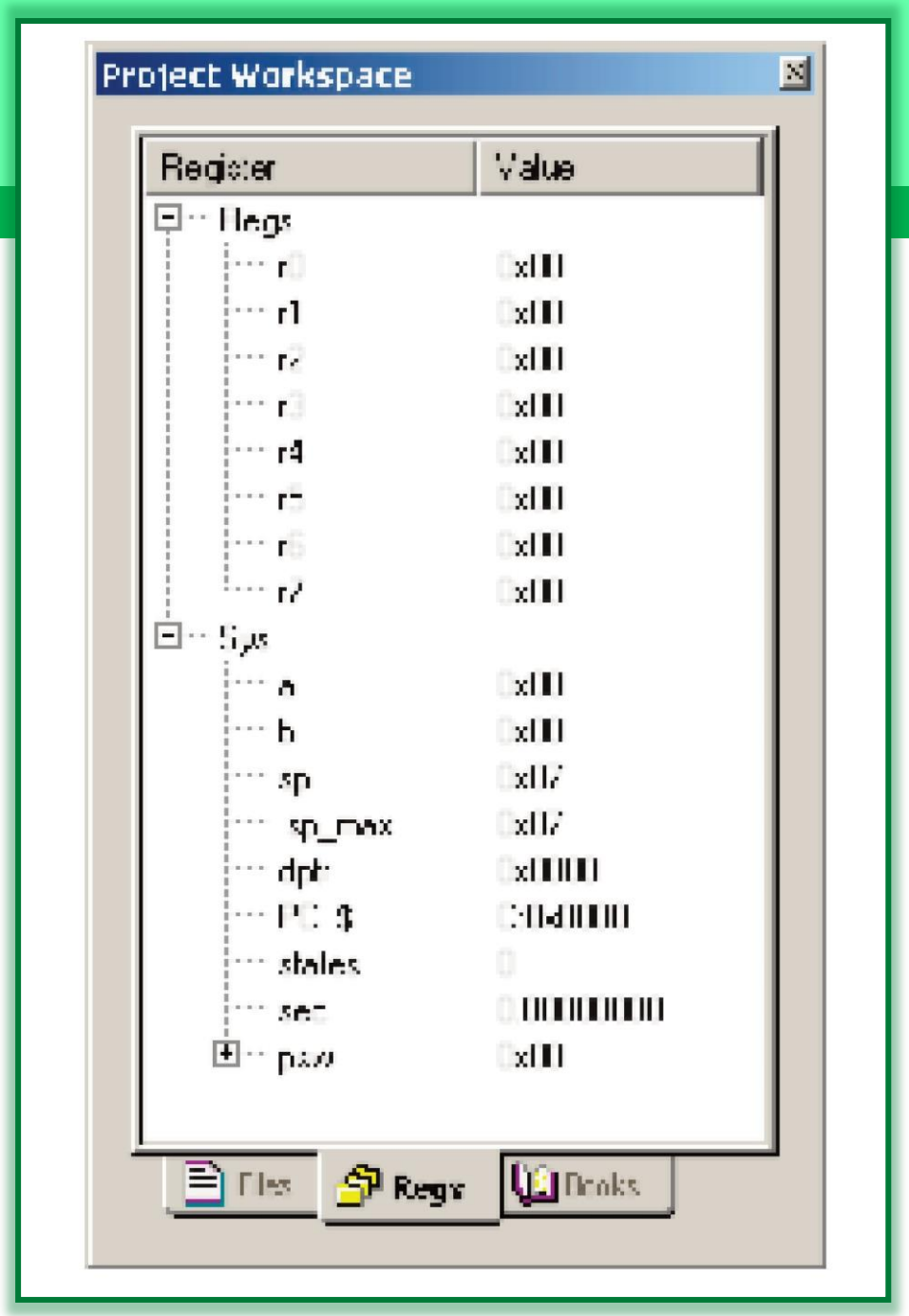


Figure 2-9. Register's Screen from Keil Simulator

Viewing Registers and Memory with a Simulator

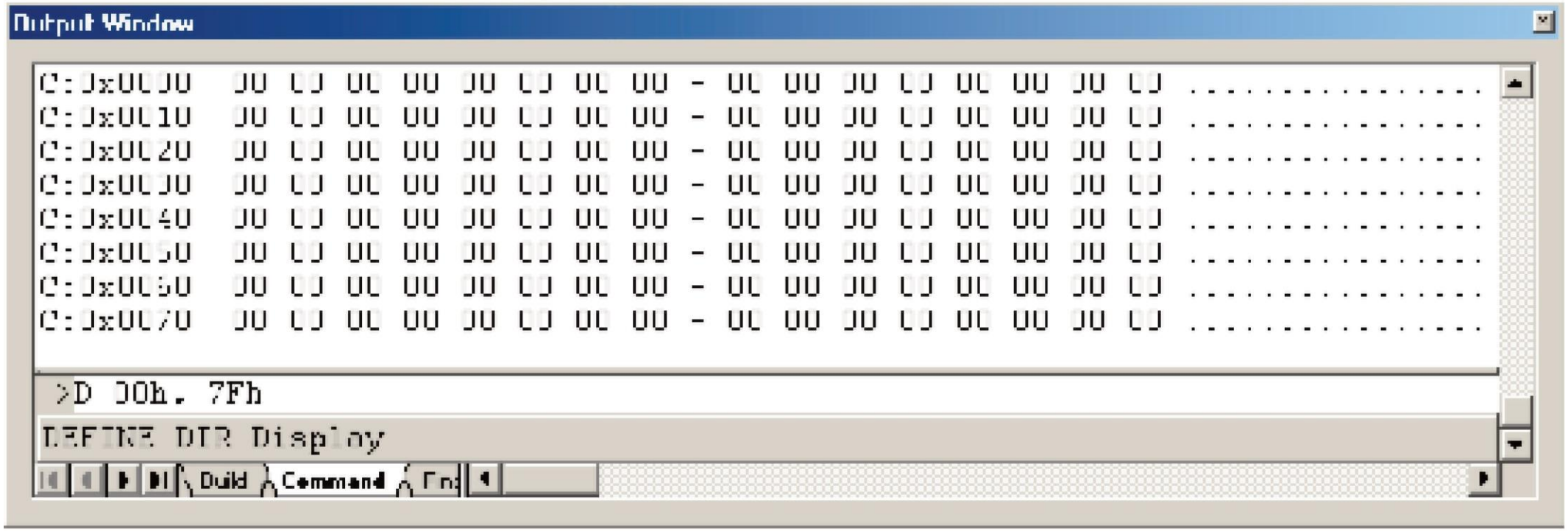


Figure 2-10. 128-Byte Memory Space from Keil Simulator