


[what-when-how](#)

In Depth Tutorials and Information

# 8051 REGISTER BANKS AND STACK

Take advantage of Google's secure infrastructure.



Contact sales

**GROW YOUR BUSINESS GLOBALLY WITH PAYPAL**



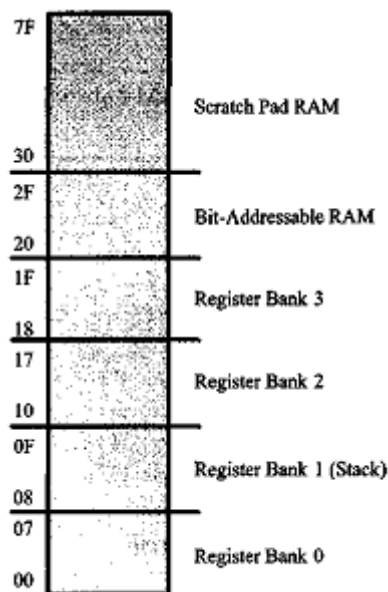
Sign Up for Free

PayPal NOW IN INDIA

## SECTION 2.7: 8051 REGISTER BANKS AND STACK

The 8051 microcontroller has a total of 128 bytes of RAM. In this section we discuss the allocation of these 128 bytes of RAM and examine their usage as registers and stack.

### RAM memory space allocation in the 8051



There are 128 bytes of RAM in the 8051 (some members, notably the 8052, have 256 bytes of RAM). The 128 bytes of RAM inside the 8051 are assigned addresses 00 to 7FH. As we will see in Chapter 5, they can be accessed directly as memory locations. These 128 bytes are divided into three different groups as follows.

**Figure 2-5. RAM Allocation in the 8051**

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. A detailed discussion of bit-address

able memory and instructions is given in Chapter 8.

- 3. A total of 80 bytes from locations 30H to 7FH are used for read and write storage, or what is normally called a *scratch pad*. These 80 locations of RAM are widely used for the purpose of storing data and parameters by 8051 programmers. We will use them in future chapters to store data brought into the CPU via I/O ports. 1

Register banks in the 8051

As mentioned earlier, a total of 32 bytes of RAM are set aside for the register banks and stack. These 32 bytes are divided into 4 banks of registers in which



each bank has 8 registers, R0 – R7. RAM locations 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 location 2, and so on, until memory location 7, which belongs to R7 of bank 0. The second bank of registers R0 – R7 starts at RAM location 08 and goes to location 0FH. The third bank of R0 – R7 starts at memory location 10H and goes to location 17H. Finally, RAM locations 18H to 1FH are set aside for the fourth bank of R0 – R7. The following shows how the 32 bytes are allocated into 4 banks:

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

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

As we can see from Figure 2-5, bank 1 uses the same RAM space as the stack. This is a major problem in programming the 8051. We must either not use register bank 1, or allocate another area of RAM for the stack. This will be discussed below.

#### 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
```

### Default register bank

If RAM locations 00 – 1F are set aside for the four register banks, which register bank of R0 – R7 do we have access to when the 8051 is powered up? The answer is register bank 0; that is, RAM locations 0, 1, 2, 3, 4, 5, 6, and 7 are accessed with the names R0, R1, R2, R3, R4, R5, R6, and R7 when programming the 8051. It is much easier to refer to these RAM locations with names such as R0, R1, and so on, than by their memory locations. Example 2-6 clarifies this concept.

#### Example 2-6

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. See Chapter 5 for a more detailed discussion of addressing modes.

```
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

As stated above, register bank 0 is the default when the 8051 is powered up. We can switch to other banks by use of the PSW (program status word) register. Bits D4 and D3 of the PSW are used to select the desired register bank as

**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

shown in Table 2-2.

The D3 and D4 bits of register PSW are often referred to as PSW.4 and PSW.3 since they can be accessed by the bit-addressable instructions SETB and CLR. For example, “SETB PSW.3” will make PSW.3 = 1 and select bank register 1. See [Example 2-7](#).

**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 the 8051**

The stack is a section of RAM used by the CPU to store information temporarily. This information could be data or an address. The CPU needs this storage area since there are only a limited number of registers.

**How stacks are accessed in the 8051**

If the stack is a section of RAM, there must be registers inside the CPU to point to it. The register used to access the stack is called the SP (stack pointer) register. The stack pointer in the 8051 is only 8 bits wide, which means that it can take values of 00 to FFH. When the 8051 is powered up, the SP register contains value 07. This means that RAM location 08 is the first location used for the stack by the 8051. The storing of a CPU register in the stack is called a PUSH, and pulling the contents off the stack back into a CPU register is called a POP. In other words, a register is pushed onto the stack to save it and popped off the stack to retrieve it. The job of the SP is very critical when push and pop actions are performed. To see how the stack works, let's look at the PUSH and POP instructions.

### Pushing onto the stack

In the 8051 the stack pointer (SP) points to the last used location of the stack. As we push data onto the stack, the stack pointer (SP) is incremented by one. Notice that this is different from many microprocessors, notably x86 processors in which the SP is decremented when data is pushed onto the stack. Examining Example 2-8, we see that as each PUSH is executed, the contents of the register are saved on the stack and SP is incremented by 1. Notice that for every byte of data saved on the stack, SP is incremented only once. Notice also that to push the registers onto the stack we must use their RAM addresses. For example, the instruction "PUSH 1" pushes register R1 onto the stack.

<b>Example 2-8</b>				
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				
<b>Solution:</b>	After PUSH 6	After PUSH 1	After PUSH 4	
0B	0B	0B	0B	
0A	0A	0A	0A F3	
09	09	09 12	09 12	
08	08 25	08 25	08 25	
Start SP = 07	SP = 08	SP = 09	SP = 0A	

### Popping from the stack

Popping the contents of the stack back into a given register is the opposite process of pushing. 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. Example 2-9 demonstrates the POP instruction.

**Example 2-9**

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

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

**Solution:**

	After POP 3	After POP 5	After POP 2
<u>0B 54</u>	<u>0B</u>	<u>0B</u>	<u>0B</u>
<u>0A F9</u>	<u>0A F9</u>	<u>0A</u>	<u>0A</u>
<u>09 76</u>	<u>09 76</u>	<u>09 76</u>	<u>09</u>
<u>08 6C</u>	<u>08 6C</u>	<u>08 6C</u>	<u>08 6C</u>
Start SP = 0B	SP = 0A	SP = 09	SP = 08

**The upper limit of the stack**

As mentioned earlier, locations 08 to 1F in the 8051 RAM can be used for the stack. This is because locations 20 – 2FH of RAM are reserved for bit-addressable memory and must not be used by the stack. If in a given program we need more than 24 bytes (08 to 1FH = 24 bytes) of stack, we can change the SP to point to RAM locations 30 – 7FH. This is done with the instruction “MOV SP, #xx”.

**CALL instruction and the stack**

In addition to using the stack to save registers, 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. More information on this will be given in Chapter 3 when we discuss the CALL instruction.



**Example 2-10**

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
PUSH 4

```

**Solution:**

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

**Stack and bank 1 conflict**

Recall from our earlier discussion that the stack pointer register points to the current RAM location available for the stack. As data is pushed onto the stack, SP is incremented. Conversely, it is decremented as data is popped off the stack into the registers. The reason that the SP is incremented after the push is to make sure that the stack is growing toward RAM location 7FH, from lower addresses to upper addresses. If the stack pointer were decremented after push instructions, we would be using RAM locations 7, 6, 5, etc., which belong to R7 to R0 of bank 0, the default register bank. This incrementing of the stack pointer for push instructions also ensures that the stack will not reach location 0 at the bottom of RAM, and consequently run out of space for the stack. However, there is a problem with the default setting of the stack. Since SP = 07 when the 8051 is powered up, the first location of the stack is RAM location 08, which also belongs to register R0 of register bank 1. In other words, register bank 1 and the stack are using the same memory space. If in a given program we need to use register banks 1 and 2, we can reallocate another section of RAM to the stack. For example, we can allocate RAM locations 60H and higher to the stack as shown in Example 2-10.

**Viewing registers and memory with a simulator**

Many assemblers and C compilers come with a simulator. Simulators allow us to view the contents of registers and memory after executing each instruction (single-stepping). We strongly recommend that you use a simulator to single-step some of the programs in this chapter and future chapters. Single-stepping a program with a simulator gives us a deeper understanding of microcontroller

architecture, in addition to the fact that we can use it to find errors in our programs. Figures 2-

Main Registers (FIG2-7)

CPU	Bank	Data	Hardware
PC	0003	R0	@R0 00 P0 FF
ACC	00	R1	@R1 00 P1 00
PSW	00	R2	@DPTR FF P2 FF
SP	07	R3	X@R0 FF P3 FF
DPTR	0000	R4	X@R1 FF TCON 00
B	00	R5	SPX XX THL0 0000
C	0	R6	XAREA XX THL1 0000
EA	0	R7	Task XX THL2 AAAA
IE	00		TaskP XX PCON 00

Figure 2-7. Register’s Screen from Pro View 32 Simulator

Data (FIG2-8)

00:	00	00	00	00	00	00	00	00	.....
08:	00	00	00	00	00	00	00	00	.....
10:	00	00	00	00	00	00	00	00	.....
18:	00	00	00	00	00	00	00	00	.....
20:	00	00	00	00	00	00	00	00	.....
28:	00	00	00	00	00	00	00	00	.....
30:	00	00	00	00	00	00	00	00	.....
38:	00	00	00	00	00	00	00	00	.....
40:	00	00	00	00	00	00	00	00	.....
48:	00	00	00	00	00	00	00	00	.....
50:	00	00	00	00	00	00	00	00	.....
58:	00	00	00	00	00	00	00	00	.....
60:	00	00	00	00	00	00	00	00	.....
68:	00	00	00	00	00	00	00	00	.....
70:	00	00	00	00	00	00	00	00	.....
78:	00	00	00	00	00	00	00	00	.....

JDtxl

Figure 2-8.128-Byte Memory Space from Pro View 32 Simulator



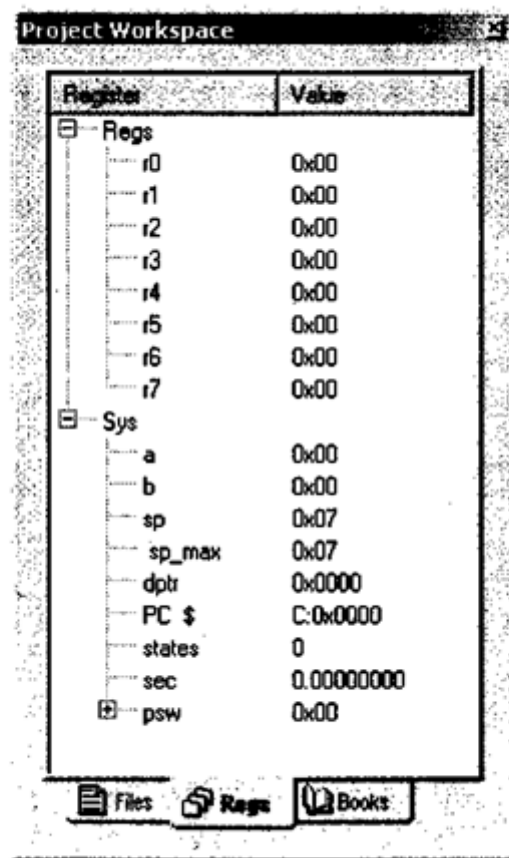


Figure 2-9. Register's Screen from Keil Simulator

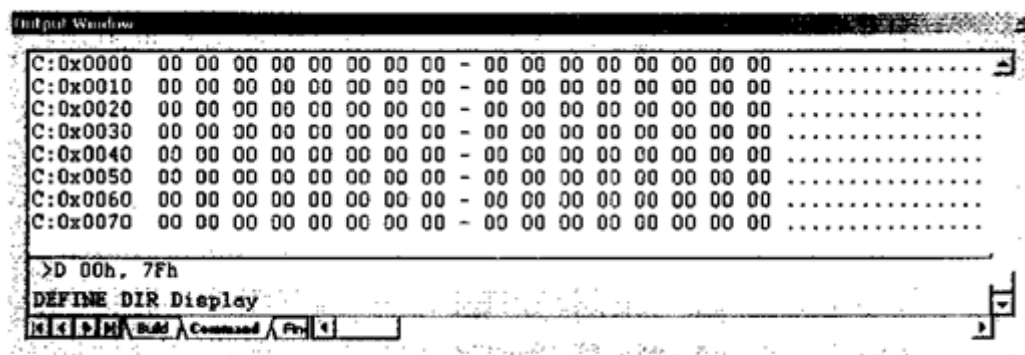


Figure 2-10. 128-Byte Memory Space from Keil Simulator

## SUMMARY

This chapter began with an exploration of the major registers of the 8051, including A, B, RO, R1, R2, R3, R4, R5, R6, R7, DPTR, and PC. The use of these registers was demonstrated in the context of programming examples. The process of creating an Assembly language program was described from writing the source file, to assembling it, linking, and executing the program. The PC (program counter) register always points to the next instruction to be executed. The way the 8051 uses program ROM space was explored because 8051 Assembly language programmers must be aware of where programs are placed in ROM, and how much memory is available.

An Assembly language program is composed of a series of statements that are either instructions or pseudo-instructions, also called *directives*. Instructions are translated by the assembler into machine code. Pseudo-instructions are not translated into machine code: They direct the assembler in how to translate instructions into

machine code. Some pseudo-instructions, called *data directives*, are used to define data. Data is allocated in byte-size increments. The data can be in binary, hex, decimal, or ASCII formats.

Flags are useful to programmers since they indicate certain conditions, such as carry or overflow, that result from execution of instructions. The stack is used to store data temporarily during execution of a program. The stack resides in the RAM space of the 8051, which was diagrammed and explained. Manipulation of the stack via POP and PUSH instructions was also explored.

SPONSORED SEARCHES



SPONSORED SEARCHES



A Computer CPU

Computer Data

20 Storage

1 0

Next post: [JUMP, LOOP, AND CALL INSTRUCTIONS](#)

Previous post: [8051 FLAG BITS AND THE PSW REGISTER](#)

## • Related Links

- [8051 Microcontroller](#)
  - [8051 MICROCONTROLLERS](#)
  - [MICROCONTROLLERS AND EMBEDDED PROCESSORS](#)
  - [OVERVIEW OF THE 8051 FAMILY](#)
  - [8051 ASSEMBLY LANGUAGE PROGRAMMING](#)
  - [INSIDE THE 8051](#)

## • :: Search WWH ::

Google Custom Search

[Help Unprivileged Children](#) ¶ [Careers](#) ¶ [Privacy Statement](#) ¶ [Copyright Information](#)