



RAM MEMORY SPACE ALLOCATION & ADDRESSING MODES

8051 – Microcontroller

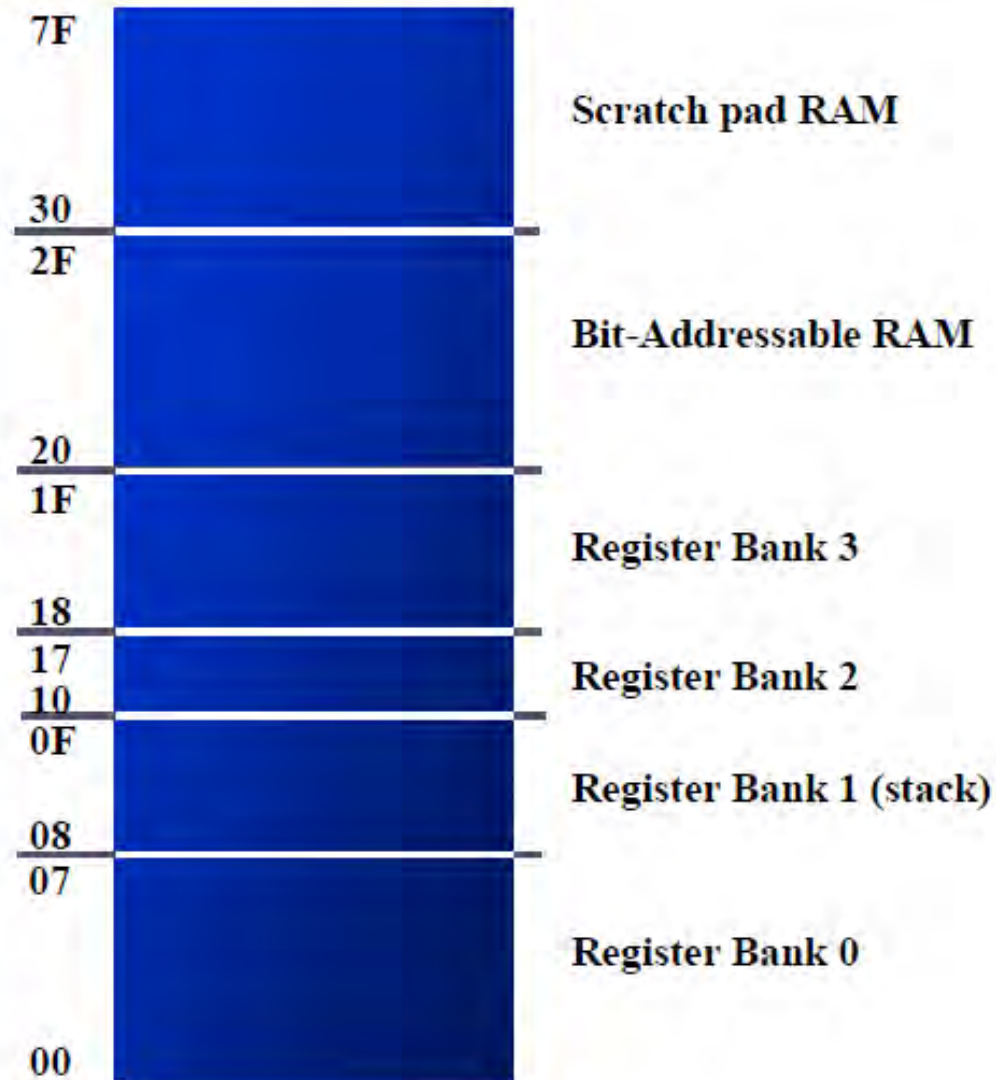
RAM ALLOCATION

- ❑ There are 128 bytes of RAM in the 8051
 - Assigned addresses 00 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 & STACK

RAM Allocation in 8051



REGISTER BANKS

- ❑ 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

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 & ROLE OF PSW

- ❑ 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 BANK SELECTION MNEMONIC

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
```



STACK

- ❑ The stack is a section of RAM used by the CPU to store information temporarily
 - This information could be data or an address
- ❑ 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 00 to FFH
 - When the 8051 is powered up, the SP register contains value 07
 - RAM location 08 is the first location begin used for the stack by the 8051



PUSH AND POP MNEMONIC

- ❑ 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



PUSHING ONTO 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



POPPING FROM 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			
0A	F9		
09	76	76	
08	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"



CALL INSTRUCTION

- ❑ 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
- ❑ 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



INCREMENTING STACK POINTER

- ❑ 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



STACK AND BANK 1

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																																
<table border="1"><tr><td>63</td><td></td></tr><tr><td>62</td><td></td></tr><tr><td>61</td><td></td></tr><tr><td>60</td><td></td></tr></table>	63		62		61		60		<table border="1"><tr><td>63</td><td></td></tr><tr><td>62</td><td></td></tr><tr><td>61</td><td></td></tr><tr><td>60</td><td>25</td></tr></table>	63		62		61		60	25	<table border="1"><tr><td>63</td><td></td></tr><tr><td>62</td><td></td></tr><tr><td>61</td><td>12</td></tr><tr><td>60</td><td>25</td></tr></table>	63		62		61	12	60	25	<table border="1"><tr><td>63</td><td></td></tr><tr><td>62</td><td>F3</td></tr><tr><td>61</td><td>12</td></tr><tr><td>60</td><td>25</td></tr></table>	63		62	F3	61	12	60	25
63																																			
62																																			
61																																			
60																																			
63																																			
62																																			
61																																			
60	25																																		
63																																			
62																																			
61	12																																		
60	25																																		
63																																			
62	F3																																		
61	12																																		
60	25																																		
Start SP = 5F	SP = 60	SP = 61	SP = 62																																



ADDRESSING MODES

- ❑ The CPU can access data in various ways, which are called *addressing modes*

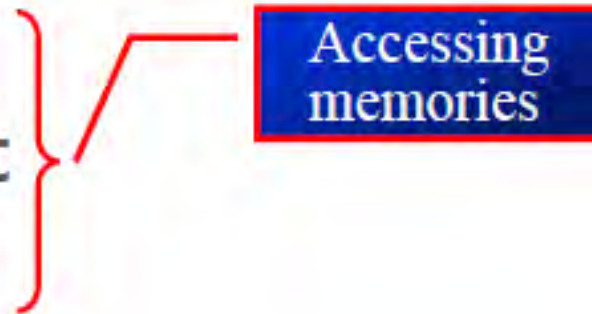
- Immediate

- Register

- Direct

- Register indirect

- Indexed



IMMEDIATE ADDRESSING MODE

- ❑ The source operand is a constant
 - The immediate data must be preceded by the pound sign, “#”
 - Can load information into any registers, including 16-bit DPTR register
 - DPTR can also be accessed as two 8-bit registers, the high byte DPH and low byte DPL

```
MOV A, #25H      ;load 25H into A
MOV R4, #62      ;load 62 into R4
MOV B, #40H      ;load 40H into B
MOV DPTR, #4521H ;DPTR=4512H
MOV DPL, #21H    ;This is the same
MOV DPH, #45H    ;as above

;illegal!! Value > 65535 (FFFFH)
MOV DPTR, #68975
```



IMMEDIATE ADDRESSING MODE

- ❑ We can use EQU directive to access immediate data

```
Count EQU 30
...
MOV R4, #COUNT ;R4=1EH
MOV DPTR, #MYDATA ;DPTR=200H

ORG 200H
MYDATA: DB "America"
```

- ❑ We can also use immediate addressing mode to send data to 8051 ports

```
MOV P1, #55H
```



REGISTER ADDRESSING MODE

- ❑ Use registers to hold the data to be manipulated

```
MOV A,R0      ;copy contents of R0 into A
MOV R2,A      ;copy contents of A into R2
ADD A,R5      ;add contents of R5 to A
ADD A,R7      ;add contents of R7 to A
MOV R6,A      ;save accumulator in R6
```

- ❑ The source and destination registers must match in size

- `MOV DPTR,A` will give an error

```
MOV DPTR,#25F5H
MOV R7,DPL
MOV R6,DPH
```

- ❑ The movement of data between Rn registers is not allowed

- `MOV R4,R7` is invalid



DIRECT ADDRESSING MODE

- ❑ It is most often used the direct addressing mode to access RAM locations 30 – 7FH
 - The entire 128 bytes of RAM can be accessed
 - The register bank locations are accessed by the register names

Direct addressing mode

```
MOV A,4 ;is same as  
MOV A,R4 ;which means copy R4 into A
```

- ❑ Contrast this with immediate addressing mode
 - There is no “#” sign in the operand

Register addressing mode

```
MOV R0,40H ;save content of 40H in R0  
MOV 56H,A ;save content of A in 56H
```



SPECIAL FUNCTION REGISTER

- ❑ The SFR (*Special Function Register*) can be accessed by their names or by their addresses

```
MOV 0E0H,#55H    ;is the same as  
MOV A,#55h       ;load 55H into A  
  
MOV 0F0H,R0      ;is the same as  
MOV B,R0         ;copy R0 into B
```

- ❑ The SFR registers have addresses between 80H and FFH
 - Not all the address space of 80 to FF is used by SFR
 - The unused locations 80H to FFH are reserved and must not be used by the 8051 programmer



Special Function Register (SFR) Addresses

Symbol	Name	Address
ACC*	Accumulator	0E0H
B*	B register	0F0H
PSW*	Program status word	0D0H
SP	Stack pointer	81H
DPTR	Data pointer 2 bytes	
DPL	Low byte	82H
DPH	High byte	83H
P0*	Port 0	80H
P1*	Port 1	90H
P2*	Port 2	0A0H
P3*	Port 3	0B0H
IP*	Interrupt priority control	0B8H
IE*	Interrupt enable control	0A8H
...

Example 5-1

Write code to send 55H to ports P1 and P2, using
(a) their names (b) their addresses

Solution :

(a) MOV A, #55H ; A=55H
 MOV P1, A ; P1=55H
 MOV P2, A ; P2=55H

(b) From Table 5-1, P1 address=80H; P2 address=A0H
 MOV A, #55H ; A=55H
 MOV 80H, A ; P1=55H
 MOV 0A0H, A ; P2=55H

- ❑ Only direct addressing mode is allowed for pushing or popping the stack
 - `PUSH A` is invalid
 - Pushing the accumulator onto the stack must be coded as `PUSH 0E0H`

Example 5-2

Show the code to push R5 and A onto the stack and then pop them back them into R2 and B, where $B = A$ and $R2 = R5$

Solution:

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



REGISTER INDIRECT ADDRESSING MODE

- ❑ A register is used as a pointer to the data
 - Only register R0 and R1 are used for this purpose
 - R2 – R7 cannot be used to hold the address of an operand located in RAM
- ❑ When R0 and R1 hold the addresses of RAM locations, they must be preceded by the "@" sign

```
MOV A,@R0 ;move contents of RAM whose  
           ;address is held by R0 into A  
MOV @R1,B ;move contents of B into RAM  
           ;whose address is held by R1
```



Example 5-3

Write a program to copy the value 55H into RAM memory locations 40H to 41H using

(a) direct addressing mode, (b) register indirect addressing mode without a loop, and (c) with a loop

Solution:

(a)

```
MOV A, #55H    ;load A with value 55H
MOV 40H, A     ;copy A to RAM location 40H
MOV 41H, A     ;copy A to RAM location 41H
```

(b)

```
MOV A, #55H    ;load A with value 55H
MOV R0, #40H   ;load the pointer. R0=40H
MOV @R0, A     ;copy A to RAM R0 points to
INC R0        ;increment pointer. Now R0=41h
MOV @R0, A     ;copy A to RAM R0 points to
```

(c)

```
MOV A, #55H    ;A=55H
MOV R0, #40H   ;load pointer.R0=40H,
MOV R2, #02    ;load counter, R2=3
AGAIN: MOV @R0, A ;copy 55 to RAM R0 points to
INC R0        ;increment R0 pointer
DJNZ R2, AGAIN ;loop until counter = zero
```



REGISTER INDIRECT ADDRESSING MODE

- ❑ The advantage is that it makes accessing data dynamic rather than static as in direct addressing mode
 - Looping is not possible in direct addressing mode

Example 5-4

Write a program to clear 16 RAM locations starting at RAM address 60H

Solution:

```
        CLR A           ;A=0
        MOV R1,#60H    ;load pointer. R1=60H
        MOV R7,#16     ;load counter, R7=16
AGAIN:  MOV @R1,A      ;clear RAM R1 points to
        INC R1         ;increment R1 pointer
        DJNZ R7,AGAIN ;loop until counter=zero
```



Example 5-5

Write a program to copy a block of 10 bytes of data from 35H to 60H

Solution:

```
        MOV R0, #35H    ;source pointer
        MOV R1, #60H    ;destination pointer
        MOV R3, #10     ;counter
BACK:   MOV A, @R0      ;get a byte from source
        MOV @R1, A      ;copy it to destination
        INC R0          ;increment source pointer
        INC R1          ;increment destination pointer
        DJNZ R3, BACK   ;keep doing for ten bytes
```



- ❑ R0 and R1 are the only registers that can be used for pointers in register indirect addressing mode
- ❑ Since R0 and R1 are 8 bits wide, their use is limited to access any information in the internal RAM
- ❑ Whether accessing externally connected RAM or on-chip ROM, we need 16-bit pointer
 - In such case, the DPTR register is used



INDEXED ADDRESSING MODE

- ❑ Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM
- ❑ The instruction used for this purpose is `MOVC A, @A+DPTR`
 - Use instruction `MOVC`, "C" means code
 - The contents of A are added to the 16-bit register DPTR to form the 16-bit address of the needed data



- ❑ In many applications, the size of program code does not leave any room to share the 64K-byte code space with data
 - The 8051 has another 64K bytes of memory space set aside exclusively for data storage
 - This data memory space is referred to as *external memory* and it is accessed only by the `MOVX` instruction
- ❑ The 8051 has a total of 128K bytes of memory space
 - 64K bytes of code and 64K bytes of data
 - The data space cannot be shared between code and data



NEXT – JUMP, LOOP AND CALL INSTRUCTIONS

