

# Lab 01



## 8051 Assembly Programming

---



# Today's Goal

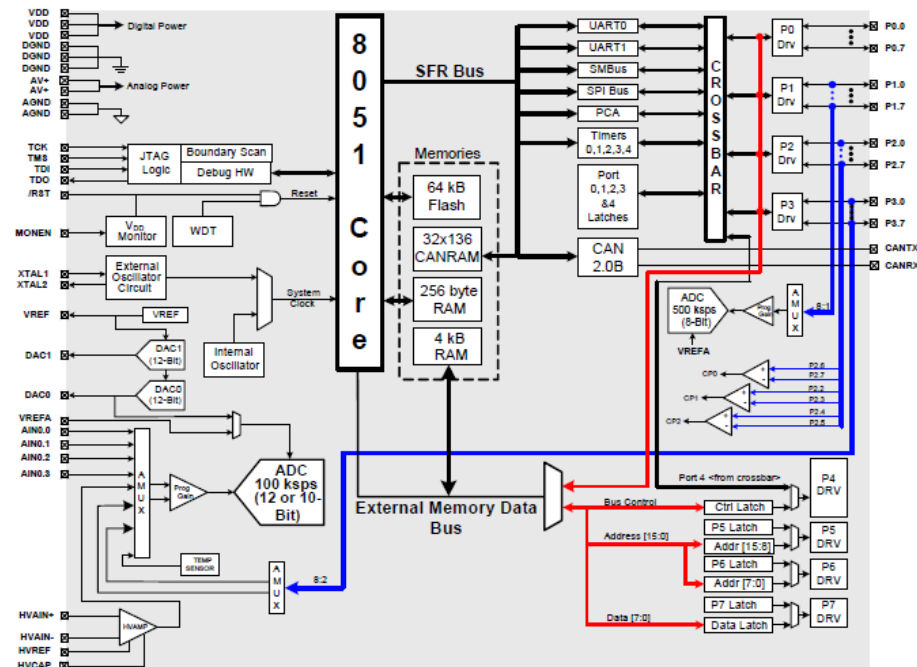
---

- Guide you to write your first 8051 assembly program
  - and perhaps your first assembly program
- Your work: write a program to compute

$$S = \sum_{i=0}^{N-1} A[i] * B[i]$$

- where  $A[i]$ ,  $B[i]$  are integer arrays (8-bit) in 8051's internal memory

- 





Fundamental: von Neumann  
model in assembly level

---



# The Von Neumann Model

---

$D = A + B * C;$   
 $E = D + F;$   
 $A = A - 1;$

CPU

PC: program counter

memory

A 10

B 20

C 30

D 40

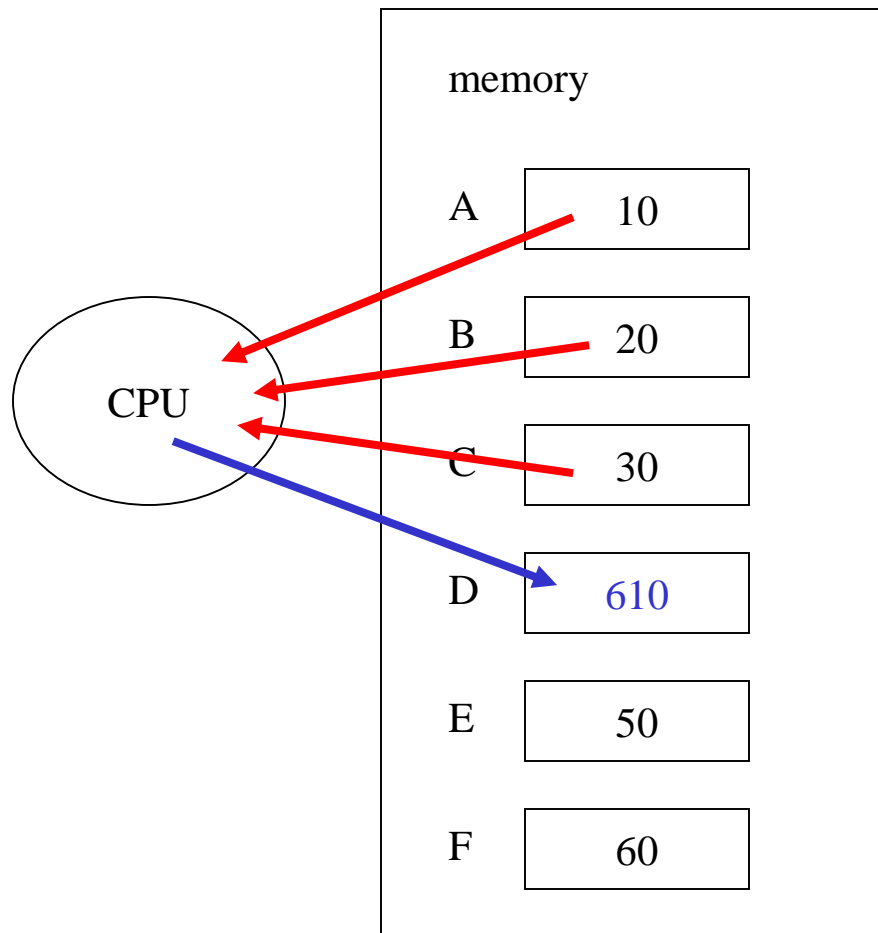
E 50

F 60

# The Von Neumann Model

PC →  $D = A + B * C;$   
 $E = D + F;$   
 $A = A - 1;$

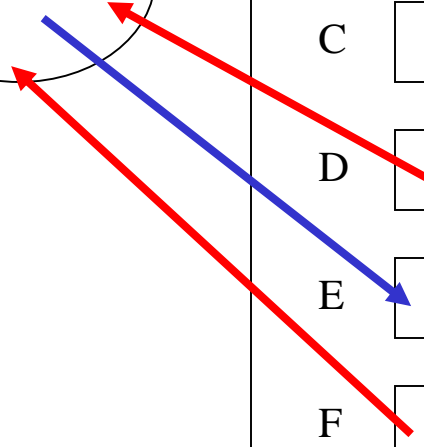
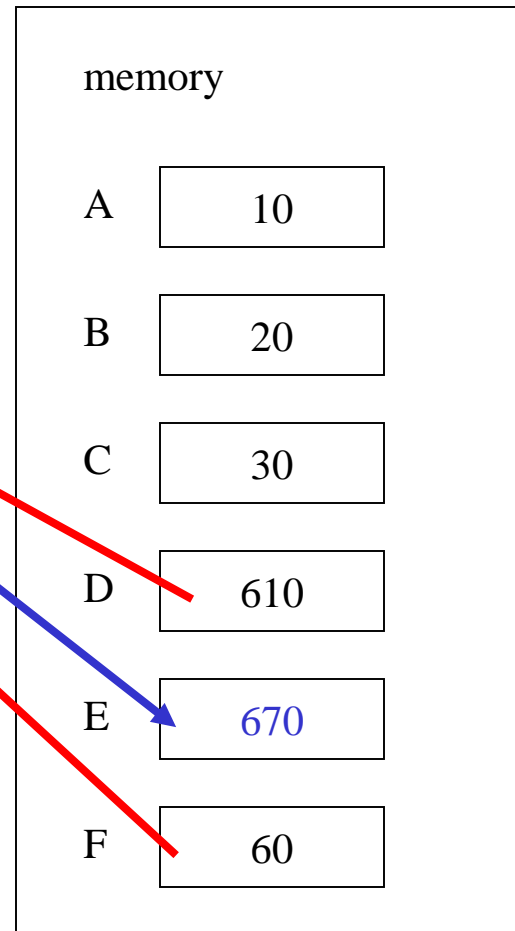
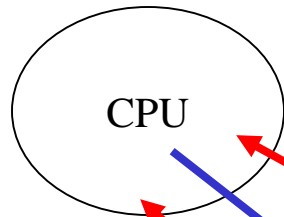
PC: program counter



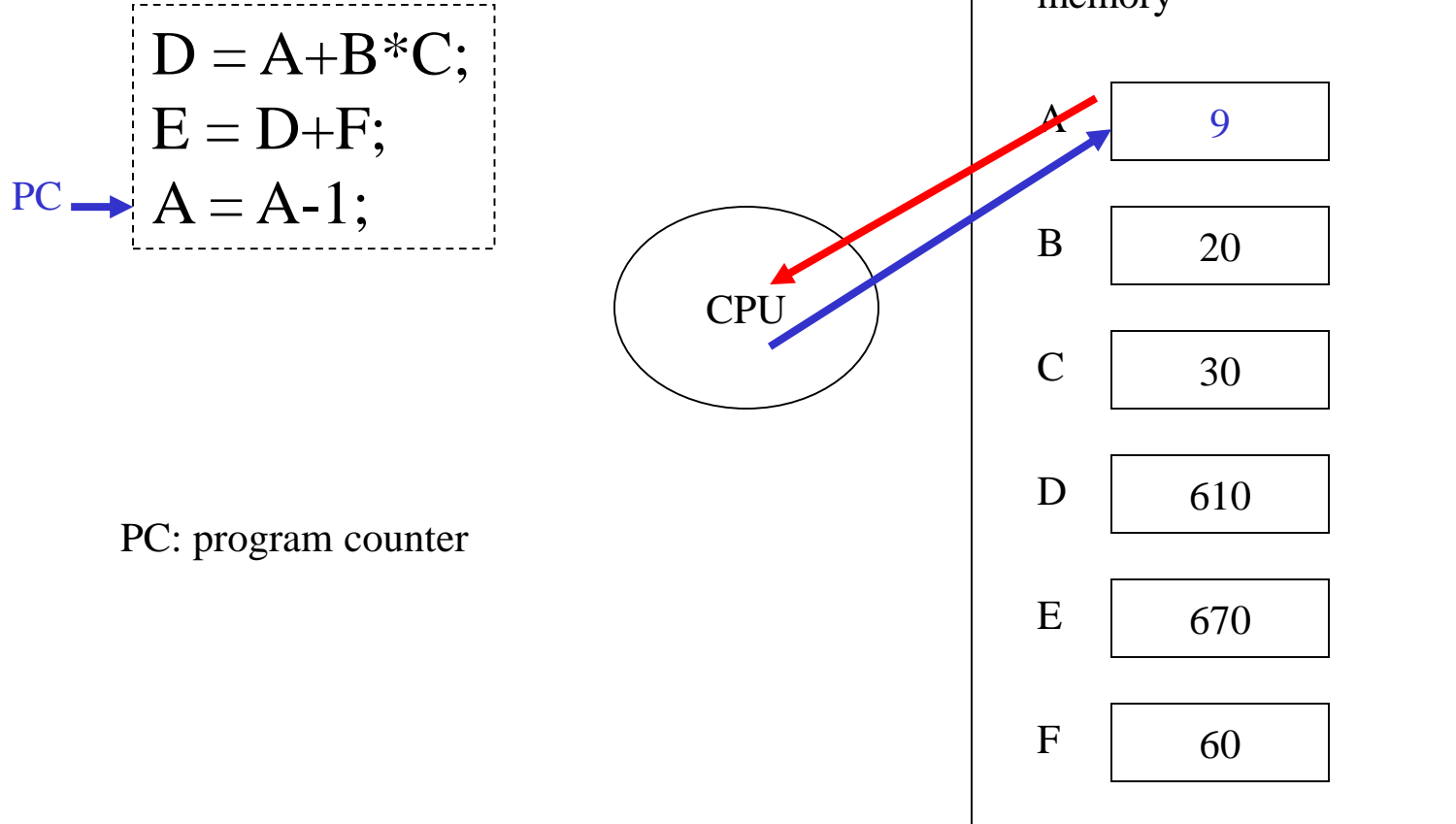
# The Von Neumann Model

PC →  $D = A + B * C;$   
 $E = D + F;$   
 $A = A - 1;$

PC: program counter



# The Von Neumann Model





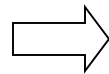


# How a CPU works?

---

- follow the von Neumann model
  - step by step
  - one *instruction* per step
- but decompose operations into primitive and regular ones

$D = A + B * C;$   
 $E = D + F;$   
 $A = A - 1;$



## assembly program

```
load R1, A;      //R1 = mem[A];
load R2, B;      //R2 = mem[B];
load R3, C;      //R3 = mem[C];
mult R4, R2, R3;  //R4=R2*R3;
add R5, R1, R4;   //R5=R1+R4;
store D, R5;      //mem[D] = R5;
load R6, F;      //R6 = mem[D];
add R7, R5, R6;   //R7 = R5+R6;
store E, R7;      //mem[E] = R7;
sub R1, R1, 1;    //R1 = R1-1;
store A, R1;      //mem[A] = R1;
```



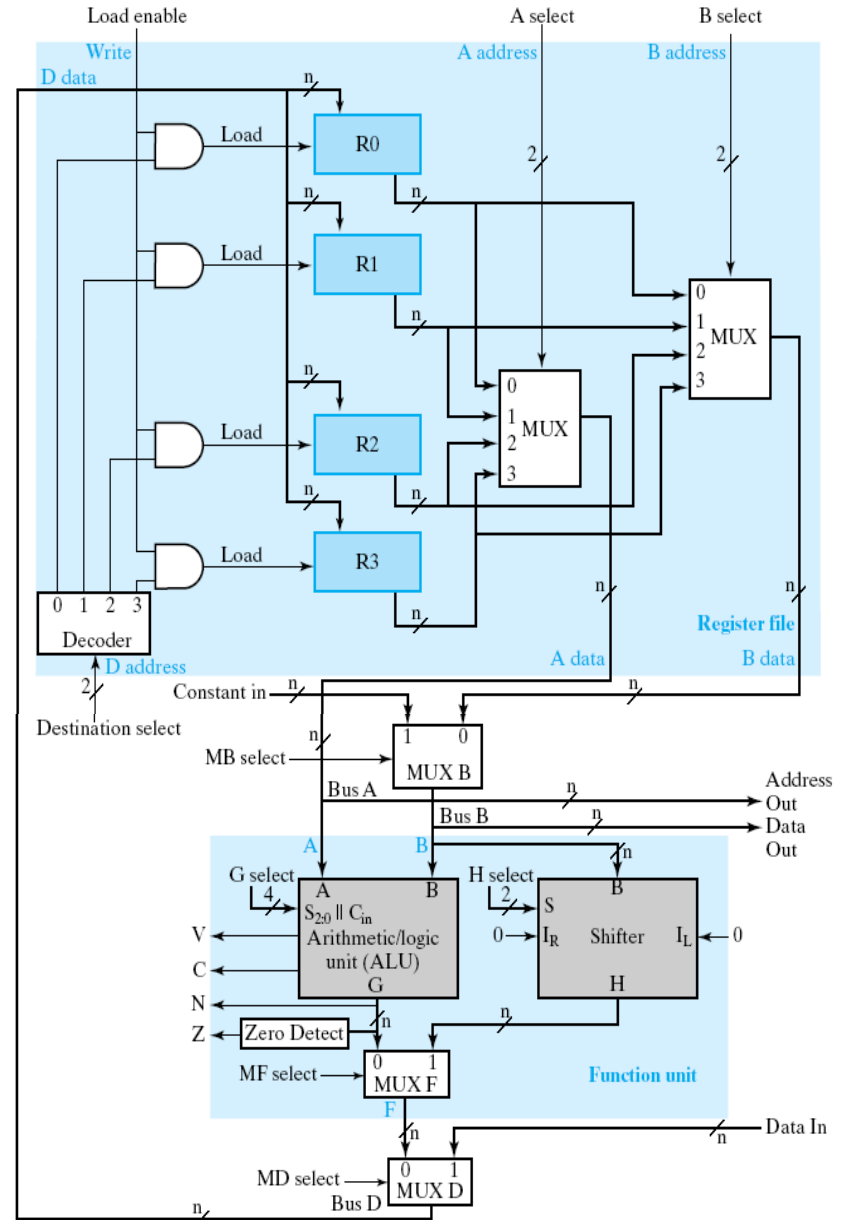
# Question

---

- Q: what's the major difference between assembly programming and high-level language programming?
- A: you have to imagine how hardware works!

# How a CPU works?

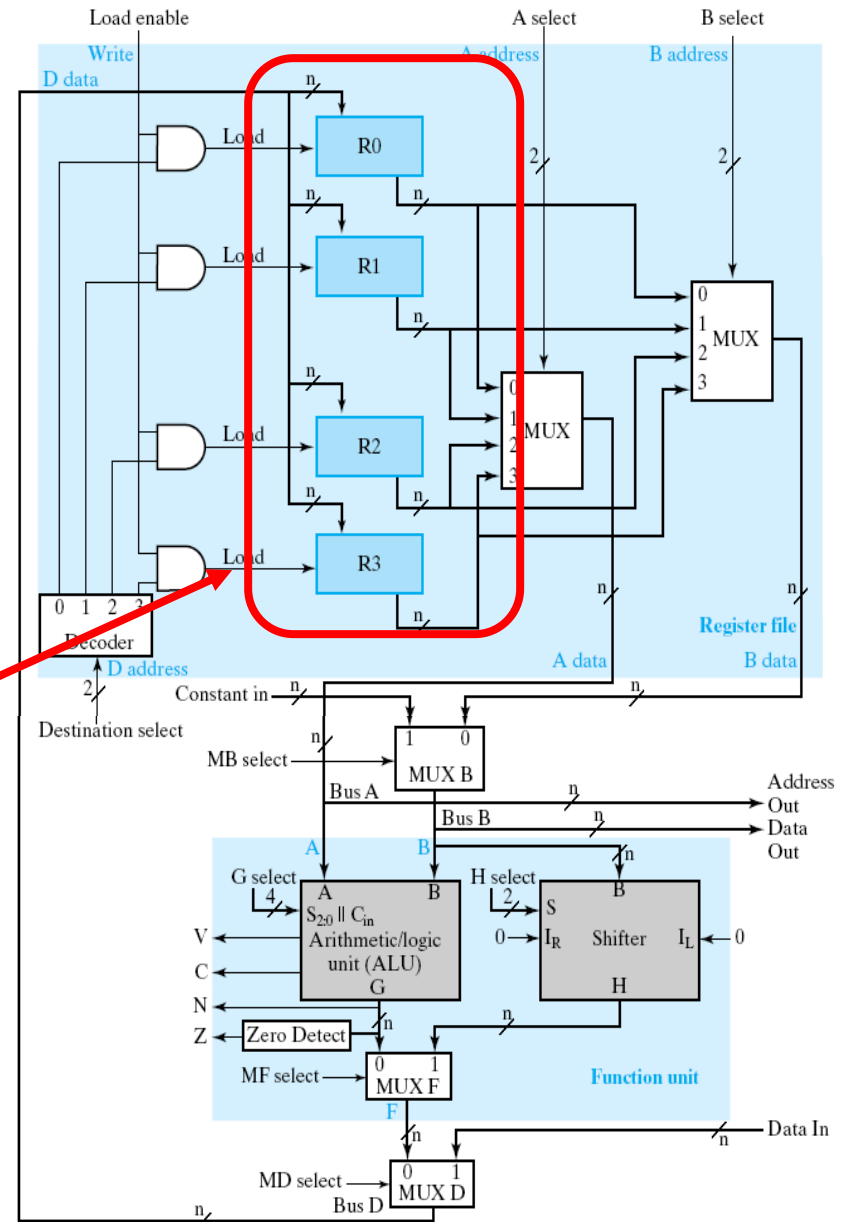
- from hardware design perspective
- the data path:



# How a CPU works?

- registers to store *variables*

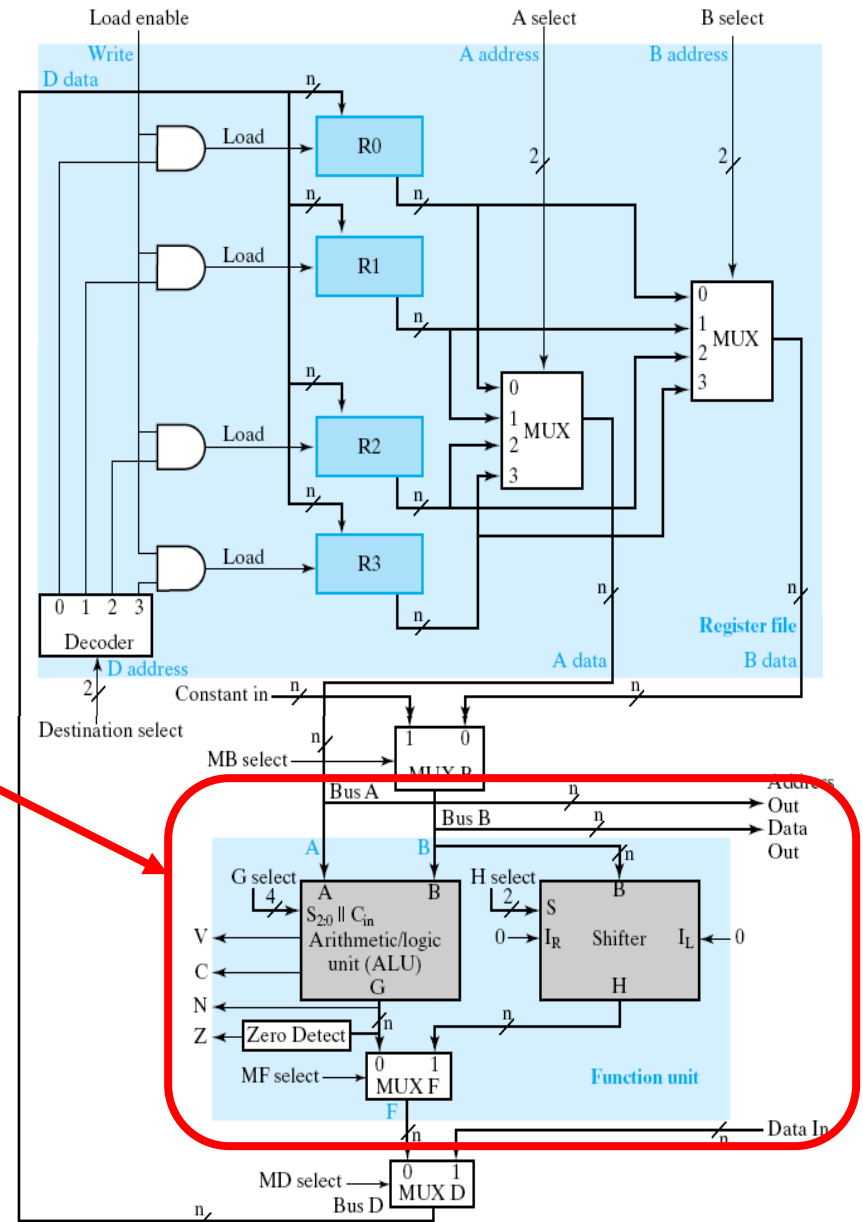
registers



# How a CPU works?

- function units to perform computation

function units

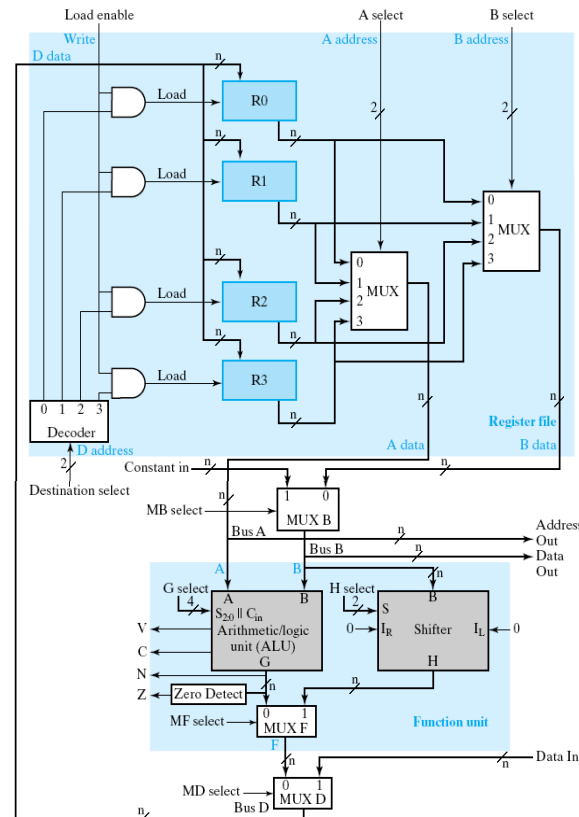


# How a CPU works?

- still follows von Neumann model

$A = B + C;$

load R1, B;      //R1 = mem[B];  
 load R2, C;      //R2 = mem[C];  
 add R3, R1, R2;    //R3 = R1+R2;  
 store A, R3;      //mem[A] = R3;



memory

A

B

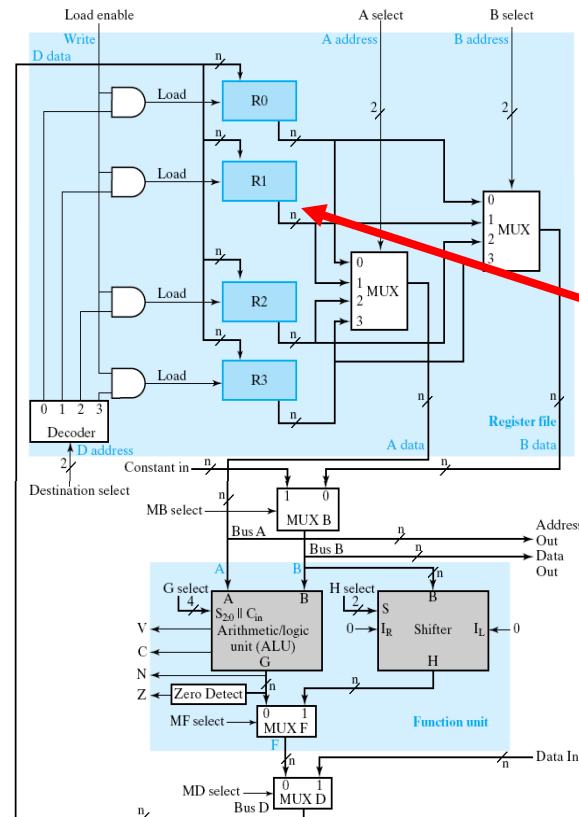
C

# How a CPU works?

- still follows von Neumann model

$A = B + C;$

→ load R1, B;      //R1 = mem[B];  
 load R2, C;      //R2 = mem[C];  
 add R3, R1, R2;    //R3 = R1+R2;  
 store A, R3;      //mem[A] = R3;



memory

A

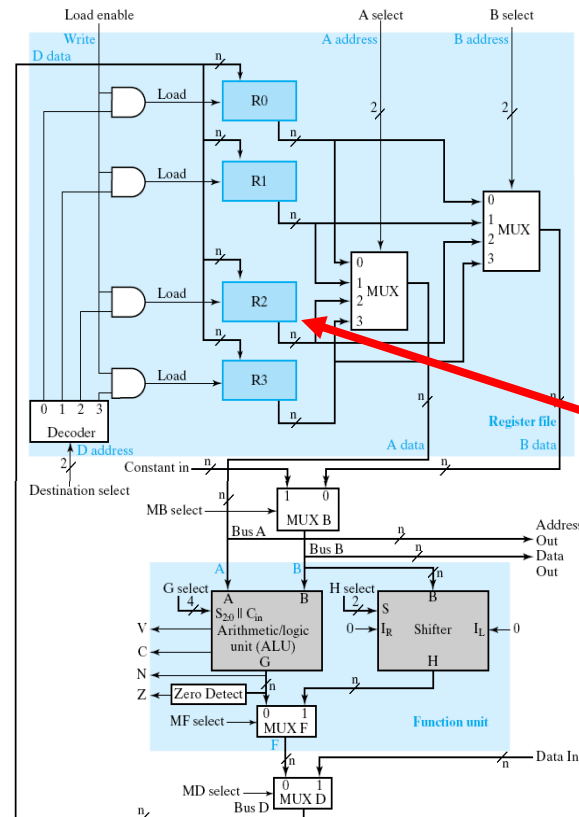
B

C

- still follows von Neumann model

**A = B+C;**

```
load R1, B;           //R1 = mem[B];
load R2, C;           //R2 = mem[C];
add R3, R1, R2;        //R3 = R1+R2;
store A, R3;          //mem[A] = R3;
```



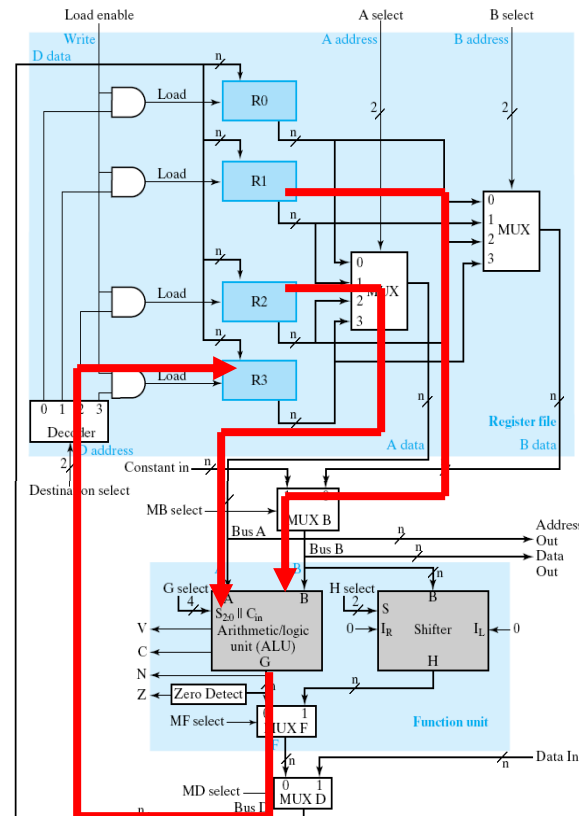


# How a CPU works?

- still follows von Neumann model

$A = B + C;$

load R1, B;      //R1 = mem[B];  
 load R2, C;      //R2 = mem[C];  
 → add R3, R1, R2;    //R3 = R1+R2;  
 store A, R3;      //mem[A] = R3;



memory

A

B

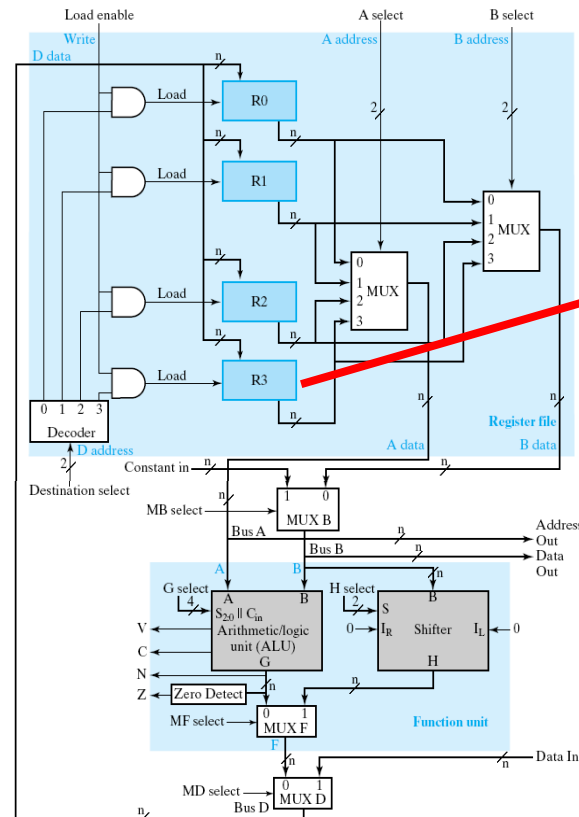
C

# How a CPU works?

- still follows von Neumann model

$A = B + C;$

load R1, B;      //R1 = mem[B];  
 load R2, C;      //R2 = mem[C];  
 add R3, R1, R2;    //R3 = R1+R2;  
 store A, R3;      //mem[A] = R3;



memory

A

B

C

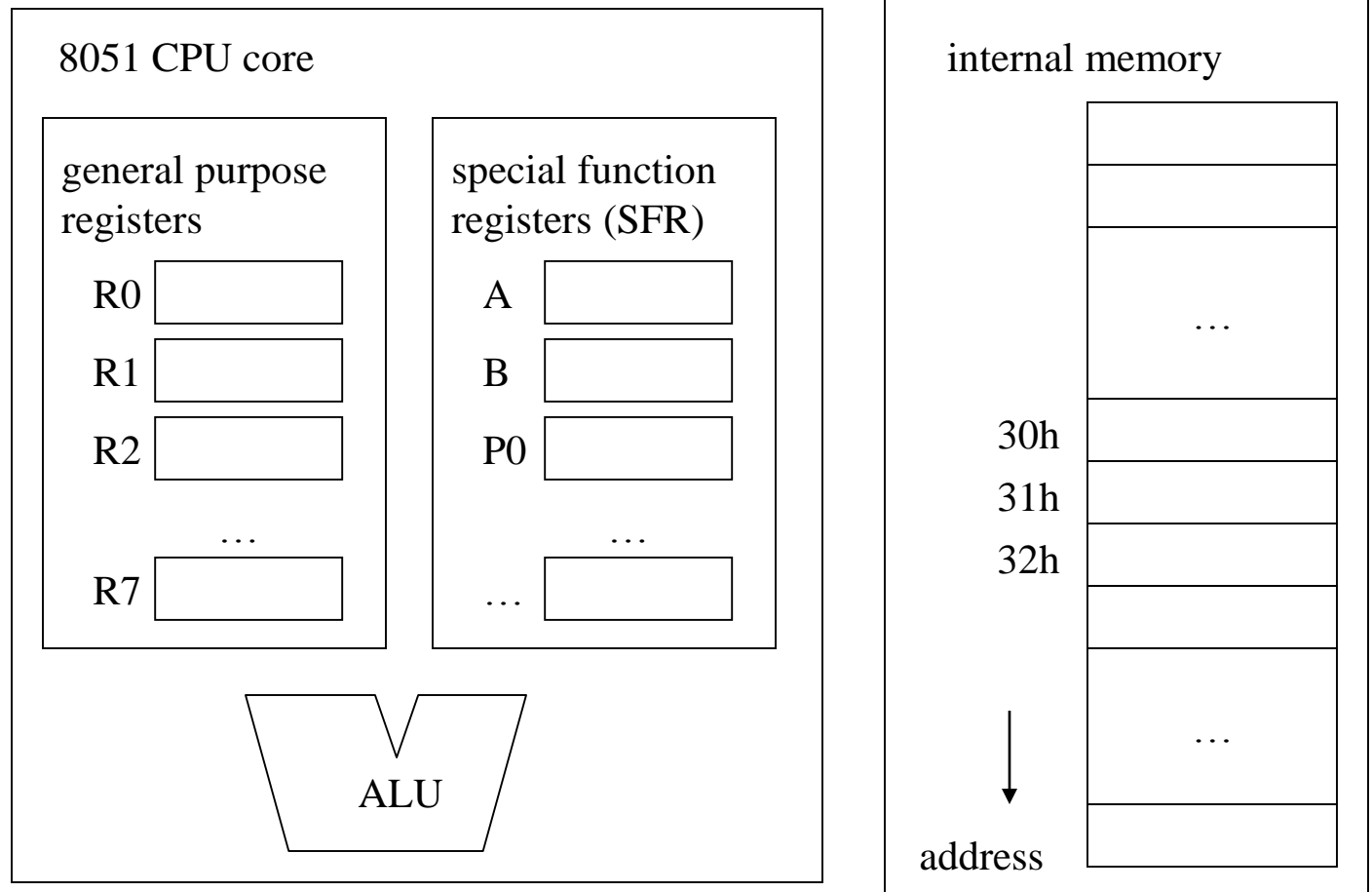


# 8051 Architecture Model

---

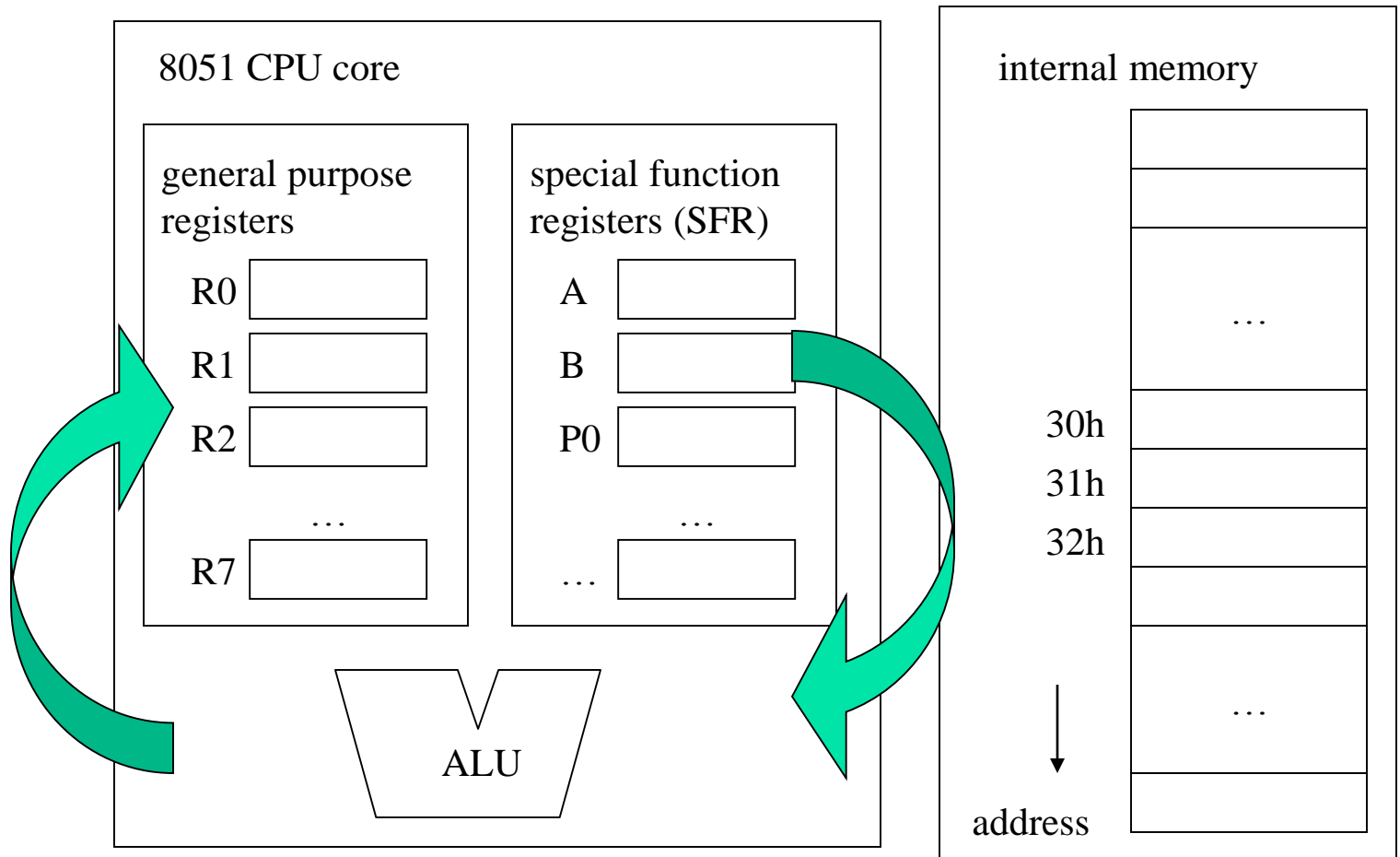
# Imagination on 8051 architecture

- Imagine how data flow in the architecture!



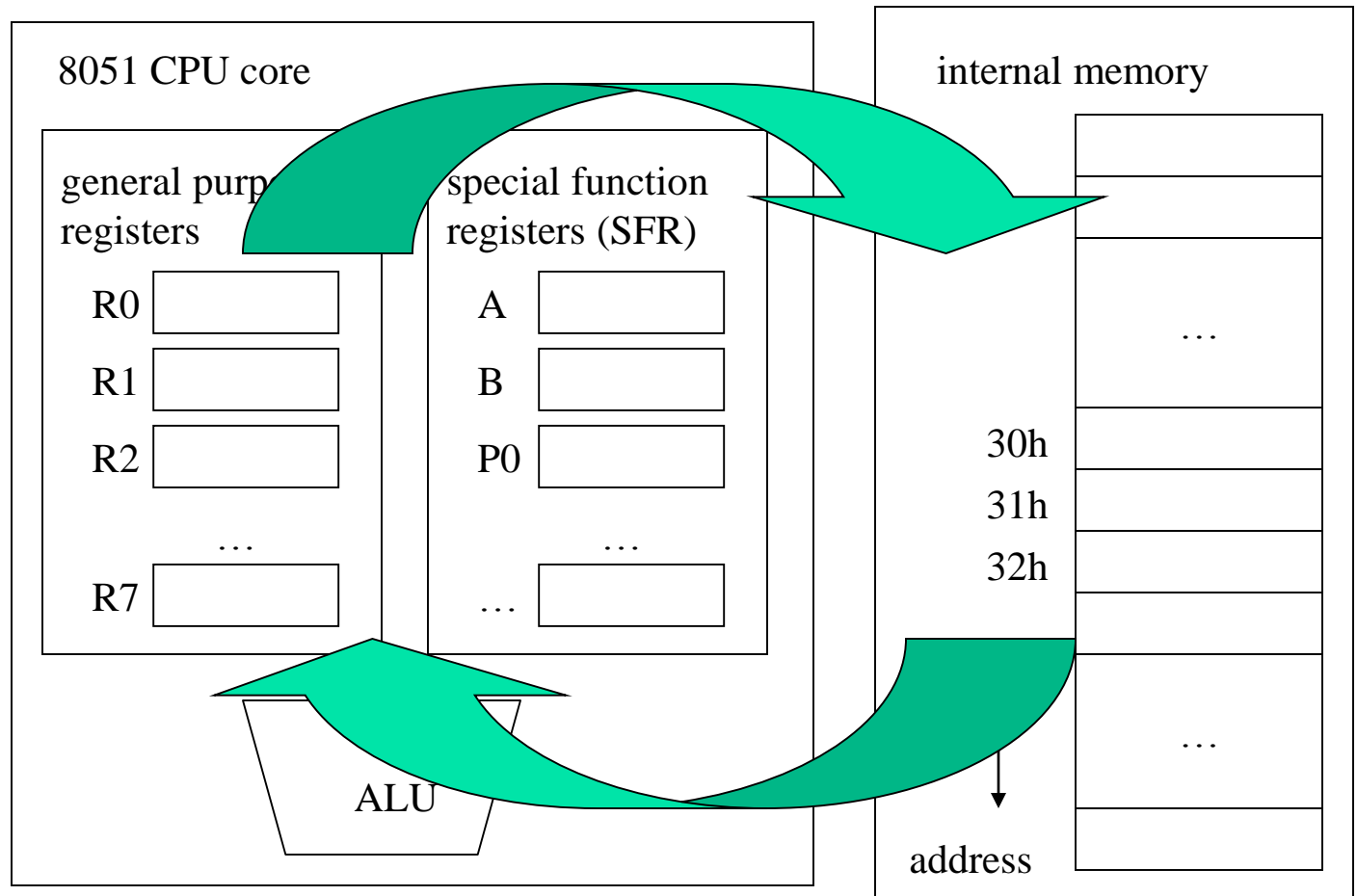
# Imagination on 8051 architecture

- flow of an arithmetic instruction



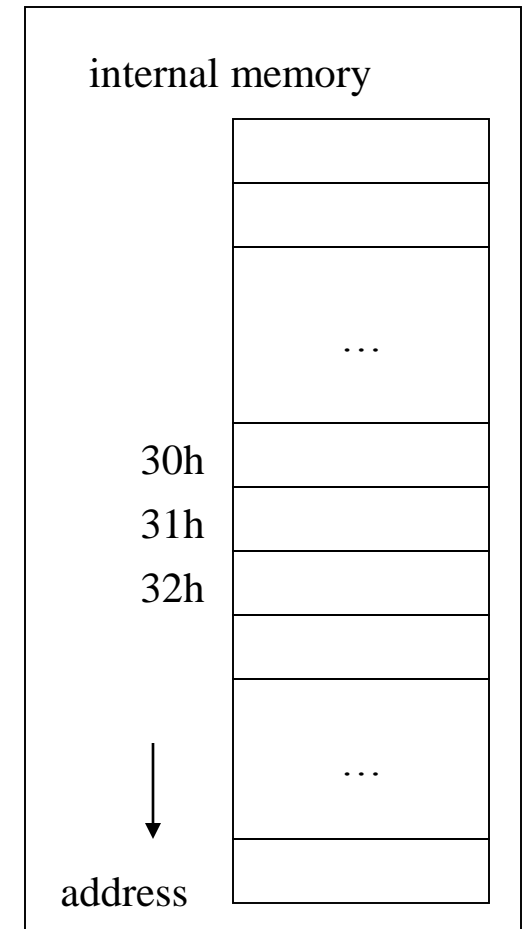
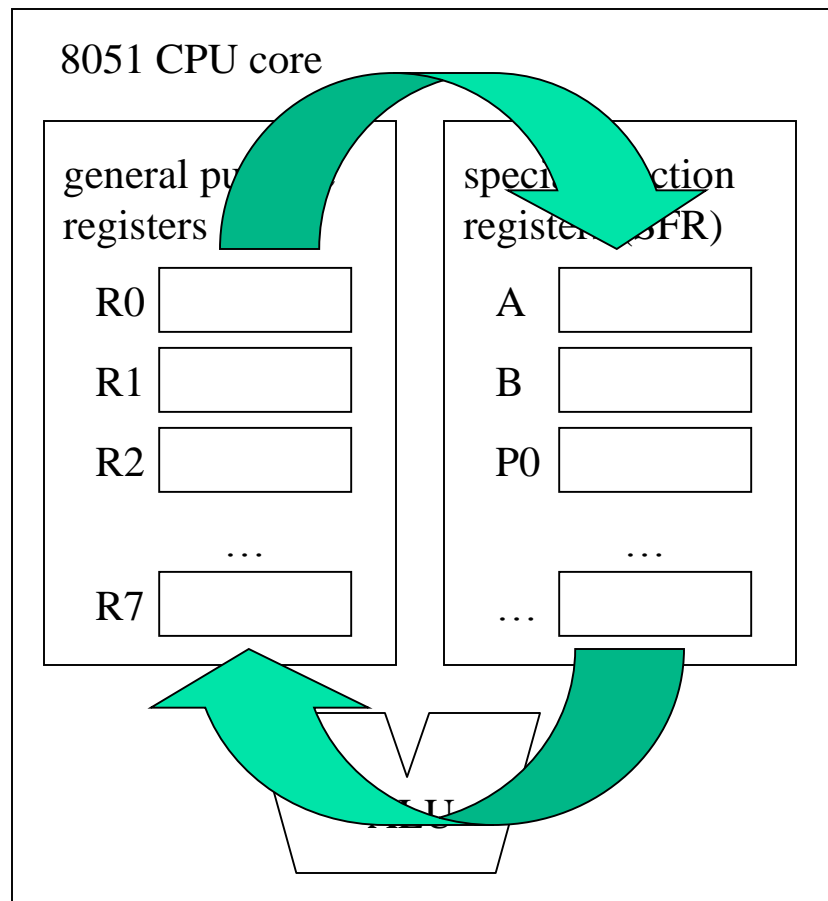
# Imagination on 8051 architecture

- data movement between memory and registers
- the MOV instruction



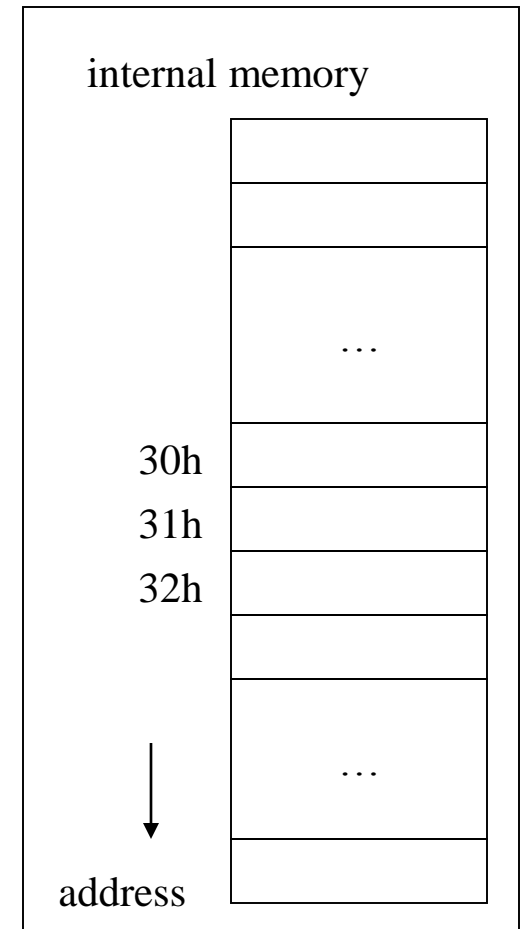
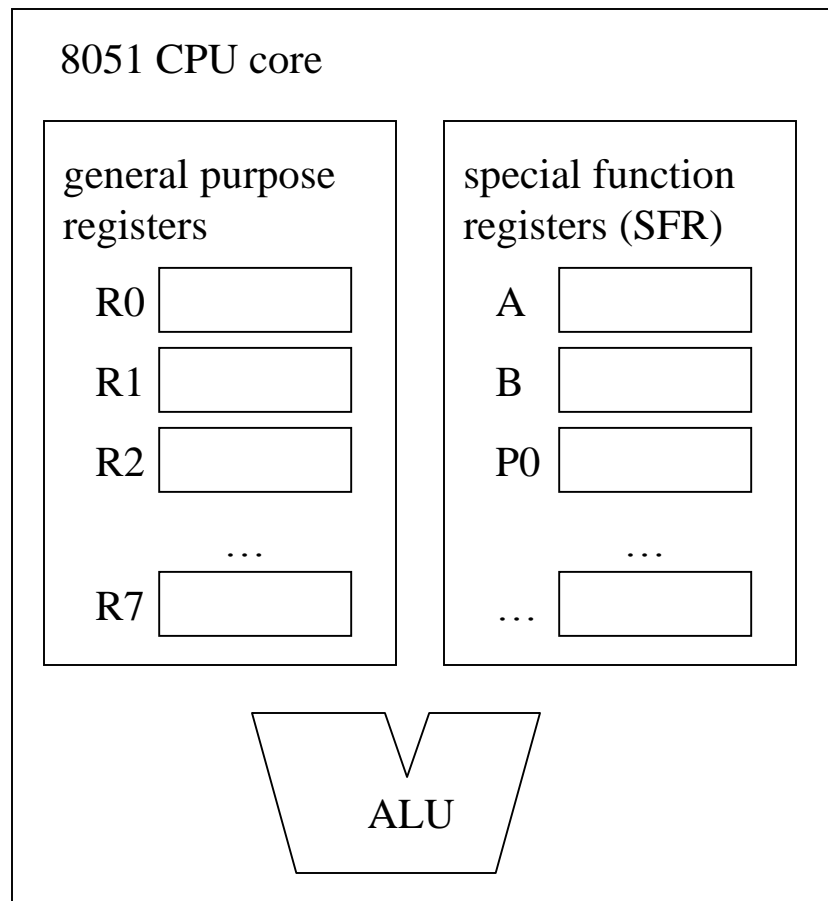
# Imagination on 8051 architecture

- the MOV also for registers



# Imagination on 8051 architecture

- Feature: most of instructions have limited use on registers
- Example:
  - ADD uses ACC
  - MUL (multiply) uses A and B





# Special Function Registers (SFR)

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8								CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH			PCON	87

↑  
Bit-addressable Registers

# A first look on 8051 assembly program



---



# Features of 8051 CPU and assembly

---

- 8-bit data operation
- two-operand assembly instruction
  - Example: `ADD A, R1` //  $A = A + R1$



# Classification of instructions

---

- Arithmetic (ADD, SUBB, MUL, etc.)
  - Notice: the use of register
  - e.g. ADD A, R0
  - e.g. MUL AB
- Branch (AJMP, ACALL, RET)
  - Notice: the jump range
- Data Transfer (MOV)
  - direct/indirect addressing mode
- Logical
  - bit-addressible instruction

# 8051 assembly language looks like

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

```
ORG 0H                ;start (origin) at location 0
    MOV R5,#25H        ;load 25H into R5
    MOV R7,#34H        ;load 34H into R7
    MOV A,#0           ;load 0 into A
    ADD A,R5           ;add contents of R5 to A
                        ;now A = A + R5
    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                ;end of asm source file
```

# 8051 assembly language looks like

general form

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

```
ORG 0H                ;start (origin) at location 0
    MOV R5,#25H        ;load 25H into R5
    MOV R7,#34H        ;load 34H into R7
    MOV A,#0           ;load 0 into A
    ADD A,R5           ;add contents of R5 to A
                        ;now A = A + R5
    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                ;end of asm source file
```

# 8051 assembly language looks like

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

```
ORG 0H                ;start (origin) at location 0
MOV R5,#25H           ;load 25H into R5
MOV R7,#34H           ;load 34H into R7
MOV A,#0              ;load 0 into A
ADD A,R5              ;add contents of R5 to A
                     ;now A = A + R5
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                   ;end of asm source file
```

**R7=0x34**

# 8051 assembly language looks like

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

```
ORG 0H                ;start (origin) at location 0
    MOV R5,#25H        ;load 25H into R5
    MOV R7,#34H        ;load 34H into R7
    MOV A,#0           ;load 0 into A
    ADD A,R5           ;add contents of R5 to A
                        ;now A = A + R5
    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                ;end of asm source file
```

**ACC=ACC+R5**



# 8051 assembly language looks like

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

```
ORG 0H                ;start (origin) at location 0
    MOV R5,#25H        ;load 25H into R5
    MOV R7,#34H        ;load 34H into R7
    MOV A,#0           ;load 0 into A
    ADD A,R5           ;add contents of R5 to A
                        ;now A = A + R5
    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               ;end of asm source file
```

goto HERE



Example 1:  $d = a * b + c$

---



# In-Class Exercise

---

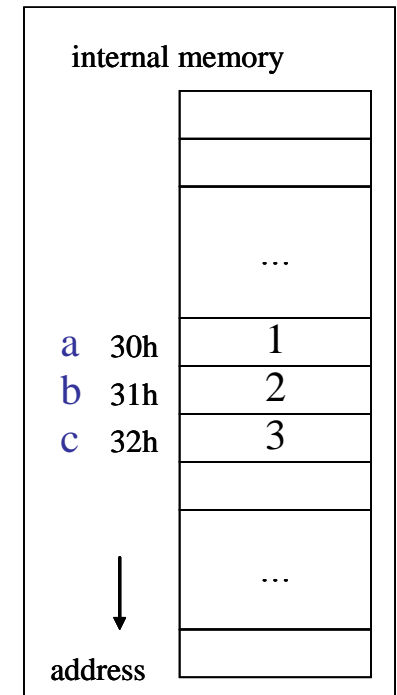
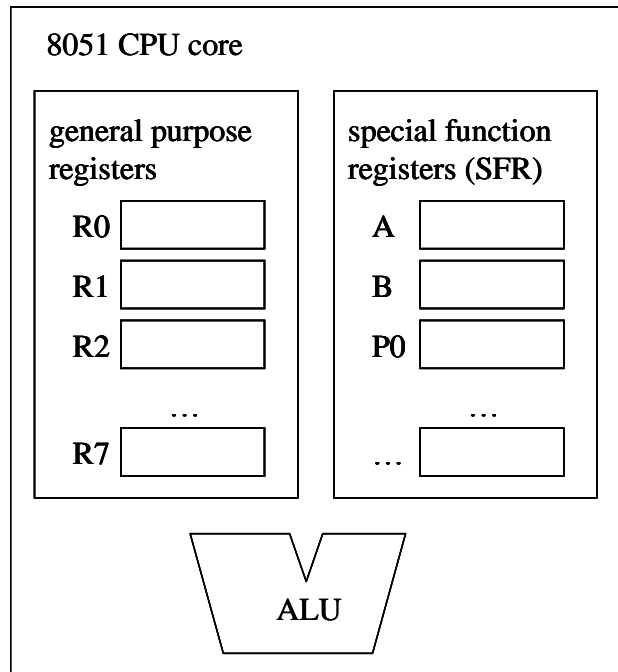
- write the program  $d = a * b + c$ 
  - where a, b, c are originally in the memory
- 8051 instructions you need:
  - add
  - mul (multiplication)
  - inc (increment)
  - mov (move)
- Check the instruction reference manual for **restrictions on using register operands!**

# Example: $d = a * b + c$

→  
mov R1, #30h ; R1=30  
mov A, @R1 ; A=mem[R1]  
inc R1 ; R1++  
mov B, @R1 ; B=mem[R1]  
inc R1 ; R1++  
mov R0, @R1 ; R0=mem[R1]=C

mul AB ; {A,B} = A\*B  
add A, R0 ; A=A+R0

wait:  
sjmp wait



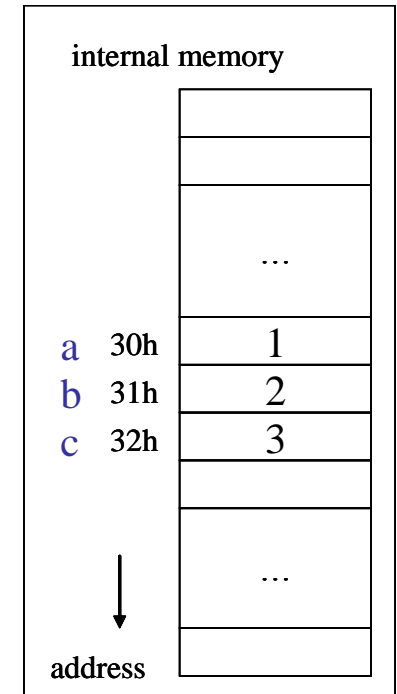
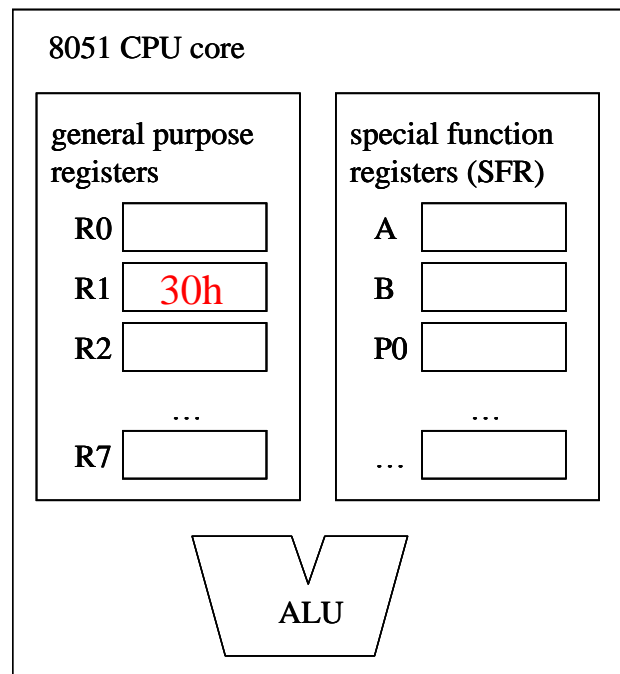
# Example: $d = a * b + c$

→

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
mov    B, @R1      ; B=mem[R1]
inc    R1          ; R1++
mov    R0, @R1     ; R0=mem[R1]=C

mul    AB          ; {A,B} = A*B
add    A, R0       ; A=A+R0

wait:  sjmp        wait
```

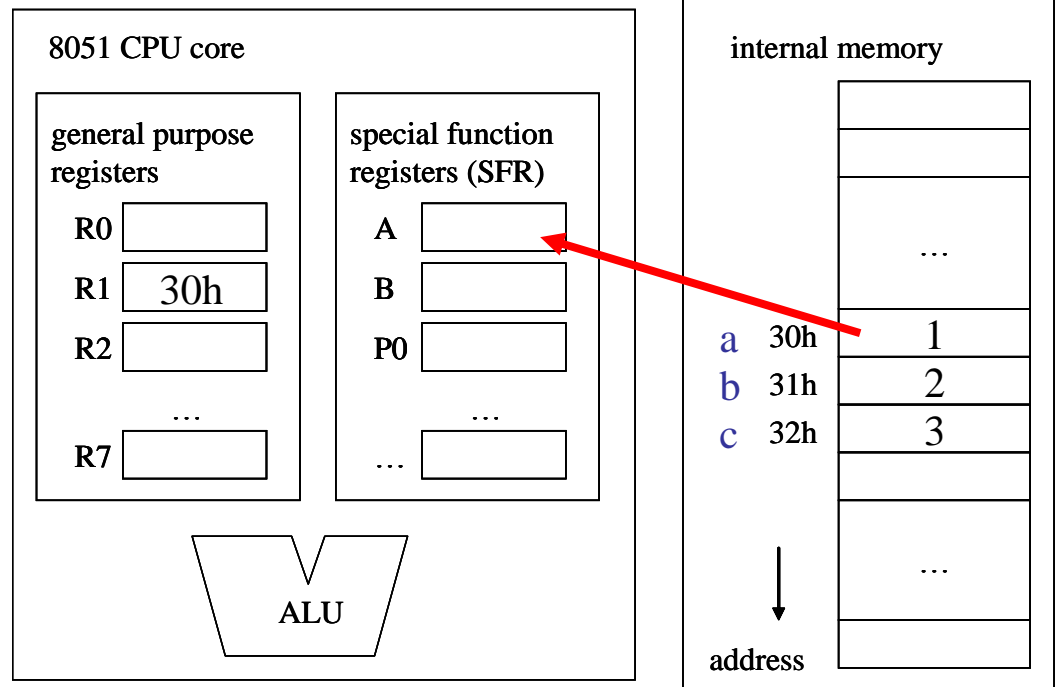


# Example: $d = a * b + c$

→  
mov R1, #30h ; R1=30  
mov A, @R1 ; A=mem[R1]  
inc R1 ; R1++  
mov B, @R1 ; B=mem[R1]  
inc R1 ; R1++  
mov R0, @R1 ; R0=mem[R1]=C

mul AB ; {A,B} = A\*B  
add A, R0 ; A=A+R0

wait:  
sjmp wait

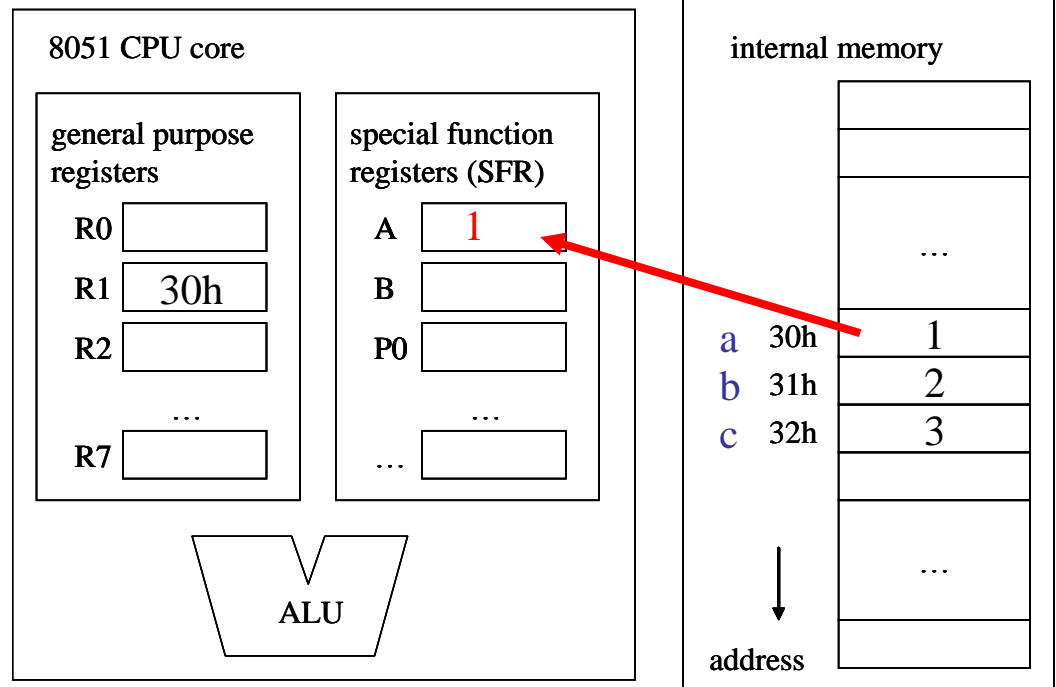


# Example: $d = a * b + c$

→  
mov R1, #30h ; R1=30  
mov A, @R1 ; A=mem[R1]  
inc R1 ; R1++  
mov B, @R1 ; B=mem[R1]  
inc R1 ; R1++  
mov R0, @R1 ; R0=mem[R1]=C

mul AB ; {A,B} = A\*B  
add A, R0 ; A=A+R0

wait:  
sjmp wait

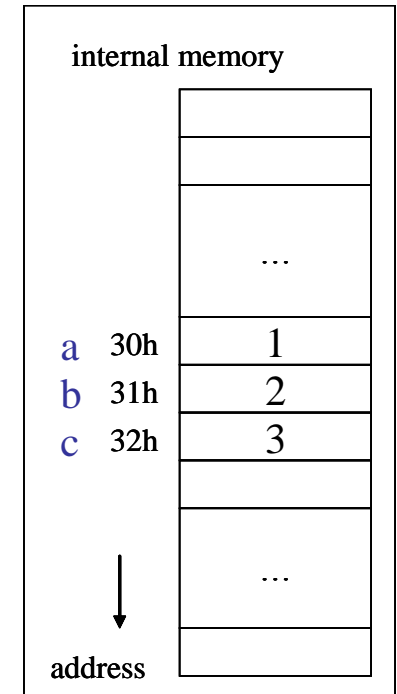
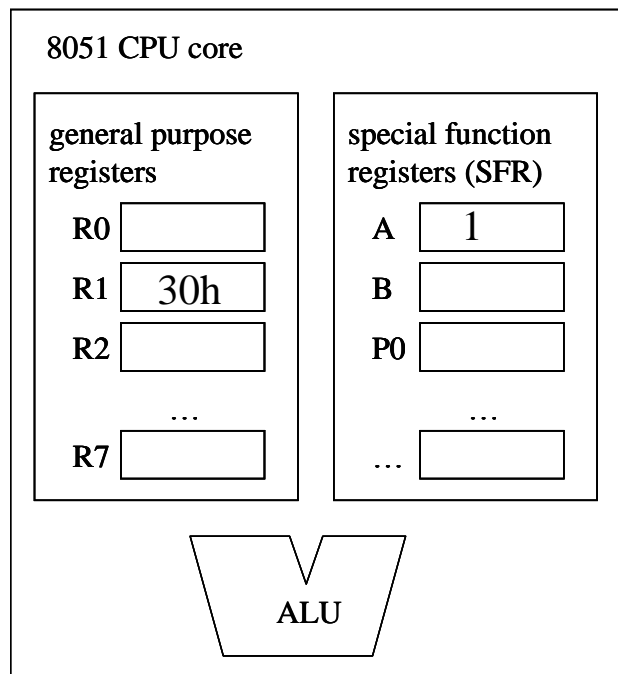


# Example: $d = a * b + c$

→  
mov R1, #30h ; R1=30  
mov A, @R1 ; A=mem[R1]  
inc R1 ; R1++  
mov B, @R1 ; B=mem[R1]  
inc R1 ; R1++  
mov R0, @R1 ; R0=mem[R1]=C

mul AB ; {A,B} = A\*B  
add A, R0 ; A=A+R0

wait:  
sjmp wait



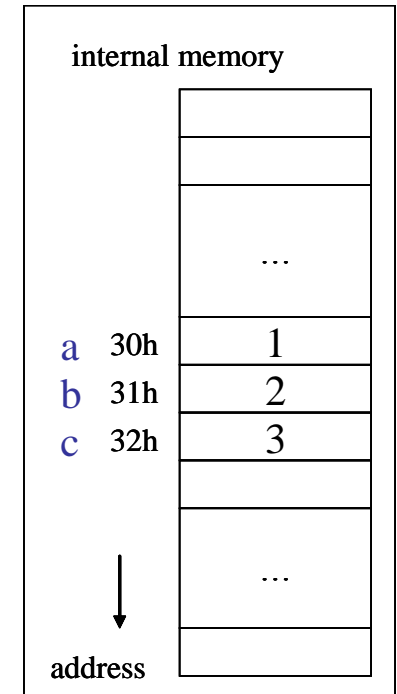
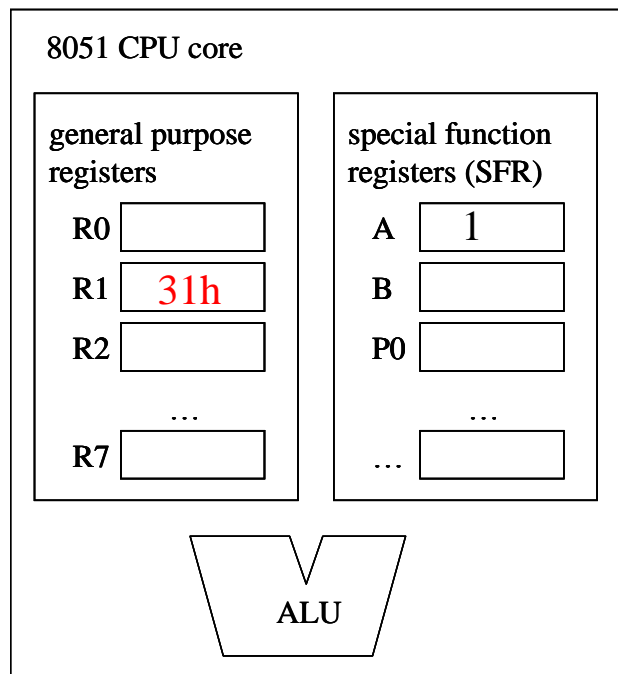


# Example: $d = a * b + c$

→  
mov R1, #30h ; R1=30  
mov A, @R1 ; A=mem[R1]  
inc R1 ; R1++  
mov B, @R1 ; B=mem[R1]  
inc R1 ; R1++  
mov R0, @R1 ; R0=mem[R1]=C

mul AB ; {A,B} = A\*B  
add A, R0 ; A=A+R0

wait:  
sjmp wait

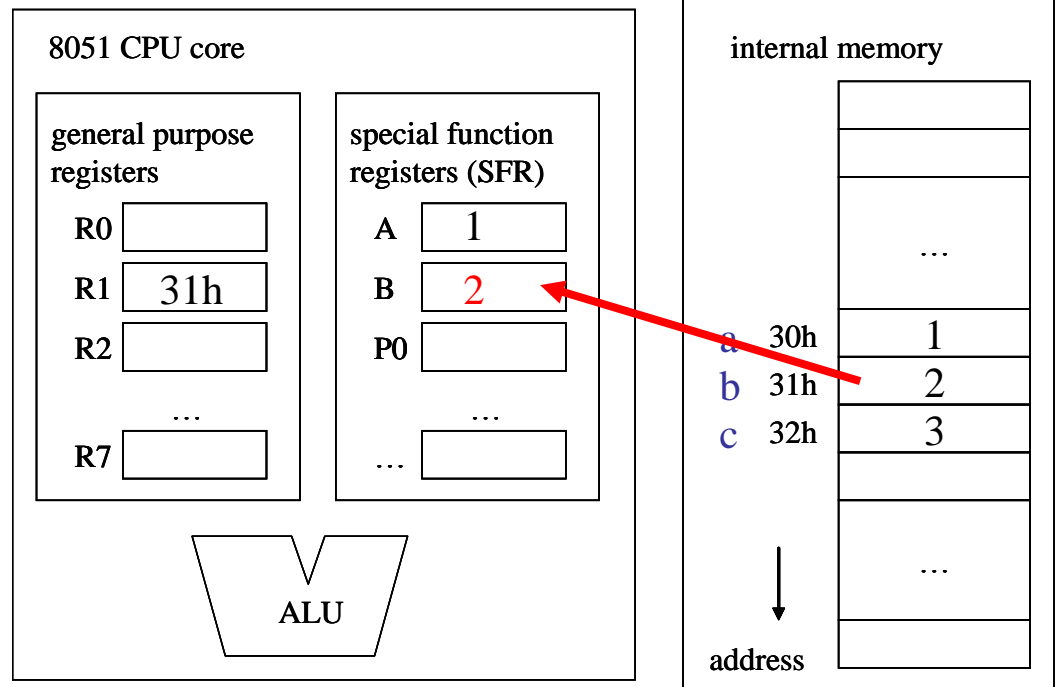


# Example: $d = a * b + c$

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
→ mov    B, @R1     ; B=mem[R1]
inc    R1          ; R1++
mov    R0, @R1     ; R0=mem[R1]=C
```

```
mul    AB          ; {A,B} = A*B
add    A, R0        ; A=A+R0
```

```
wait:  sjmp        wait
```

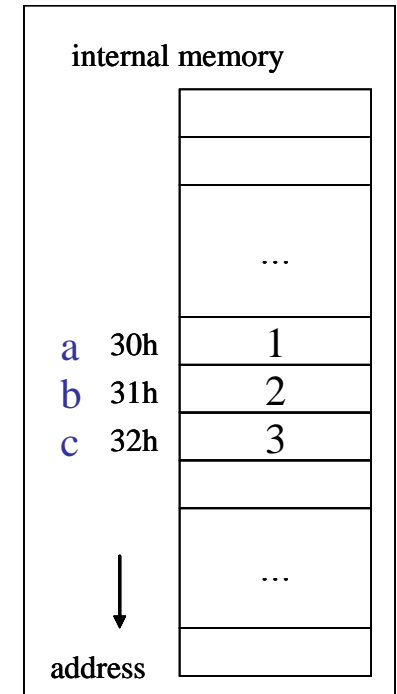
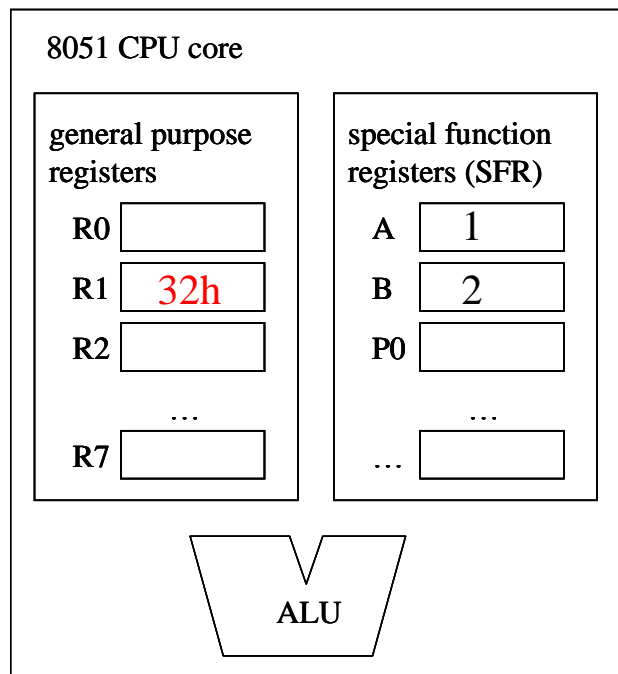


# Example: $d = a * b + c$

mov R1, #30h ; R1=30  
mov A, @R1 ; A=mem[R1]  
inc R1 ; R1++  
→ mov B, @R1 ; B=mem[R1]  
inc R1 ; R1++  
mov R0, @R1 ; R0=mem[R1]=C

mul AB ; {A,B} = A\*B  
add A, R0 ; A=A+R0

wait:  
sjmp wait

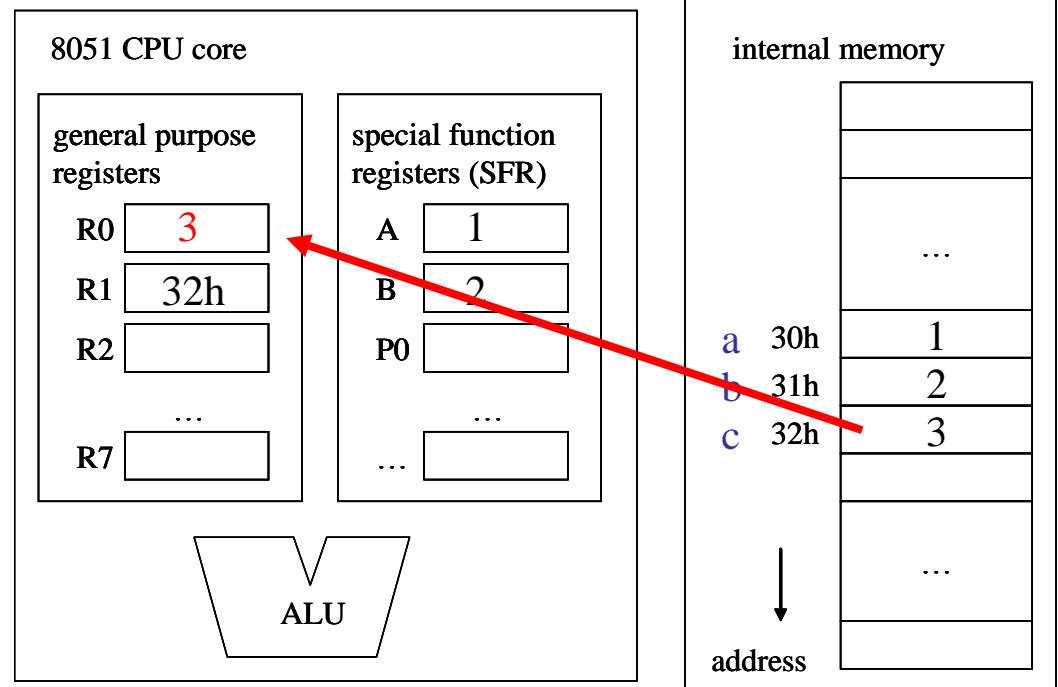


# Example: $d = a * b + c$

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
mov    B, @R1      ; B=mem[R1]
inc    R1          ; R1++
→ mov   R0, @R1     ; R0=mem[R1]=C
```

```
mul    AB          ; {A,B} = A*B
add    A, R0        ; A=A+R0
```

```
wait:  sjmp         wait
```



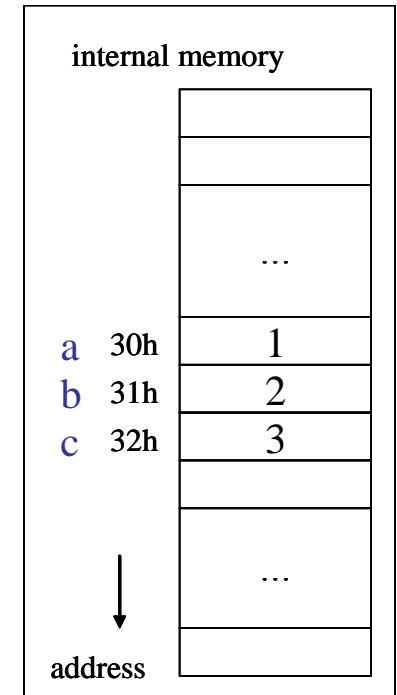
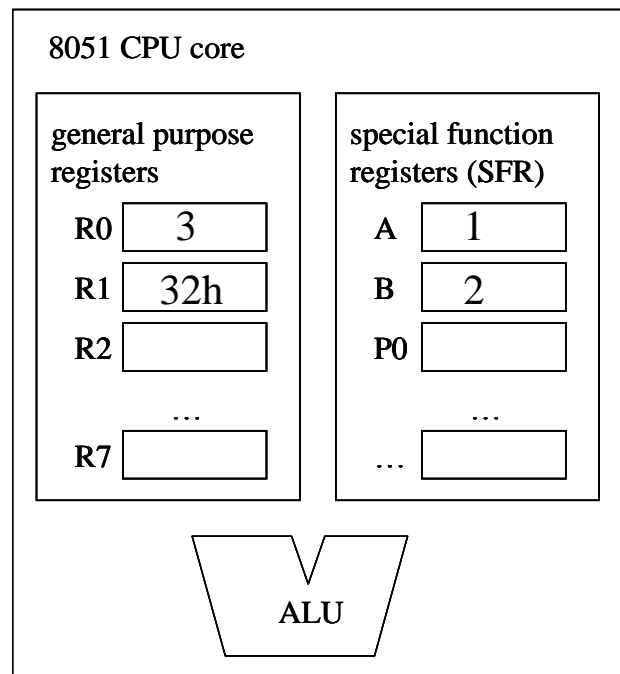
# Example: $d = a * b + c$

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
mov    B, @R1      ; B=mem[R1]
inc    R1          ; R1++
mov    R0, @R1     ; R0=mem[R1]=C
```

→

```
mul    AB          ; {A,B} = A*B
add    A, R0       ; A=A+R0

wait:  sjmp        wait
```

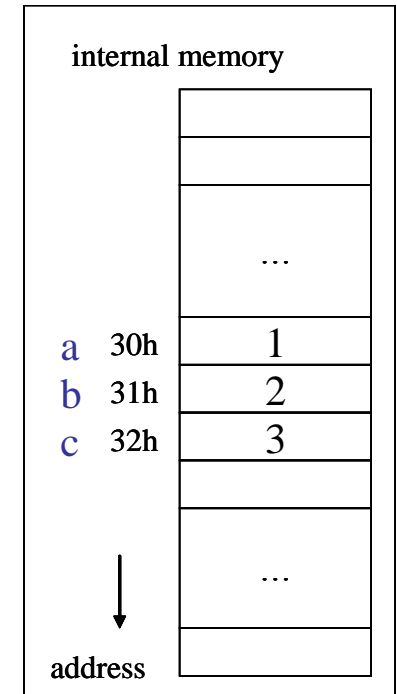
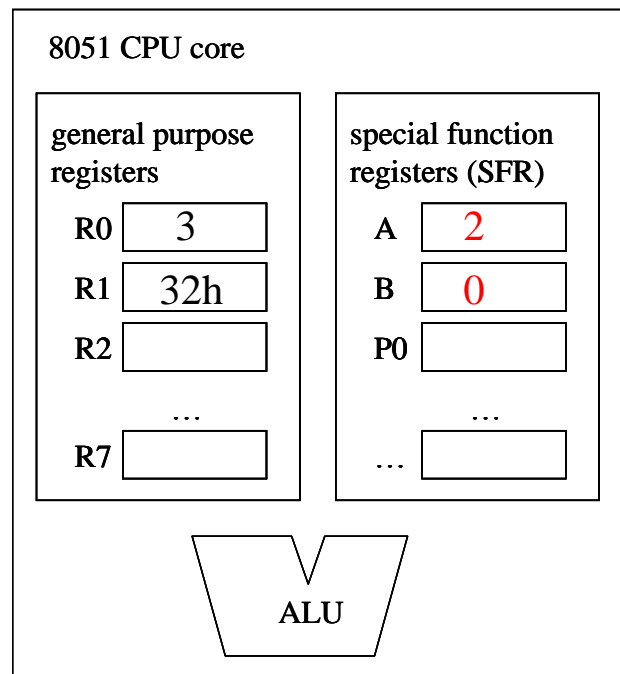


# Example: $d = a * b + c$

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
mov    B, @R1      ; B=mem[R1]
inc    R1          ; R1++
mov    R0, @R1     ; R0=mem[R1]=C
```

```
mul    AB          ; {A,B} = A*B
add    A, R0        ; A=A+R0
```

```
wait:  sjmp        wait
```



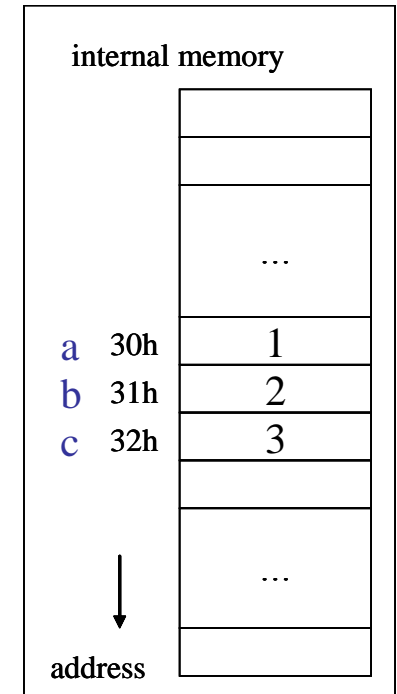
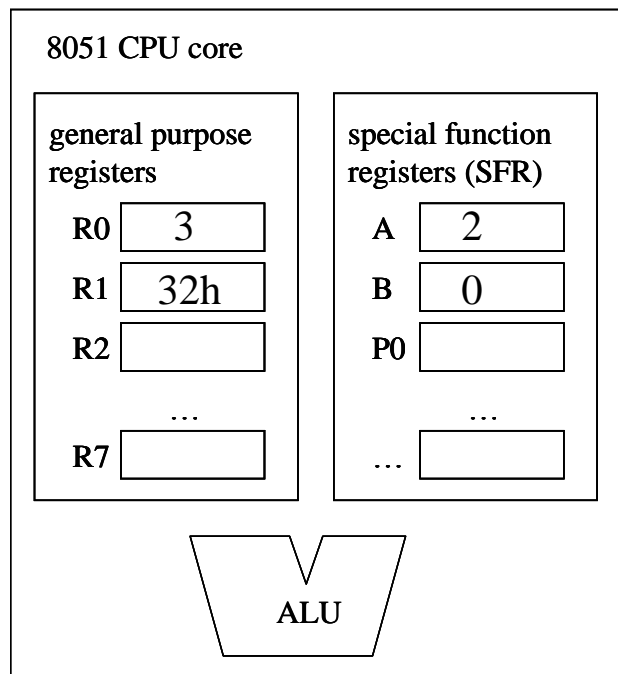
# Example: $d = a * b + c$

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
mov    B, @R1      ; B=mem[R1]
inc    R1          ; R1++
mov    R0, @R1     ; R0=mem[R1]=C
```

```
mul    AB          ; {A,B} = A*B
add    A, R0        ; A=A+R0
```

wait:

```
sjmp   wait
```



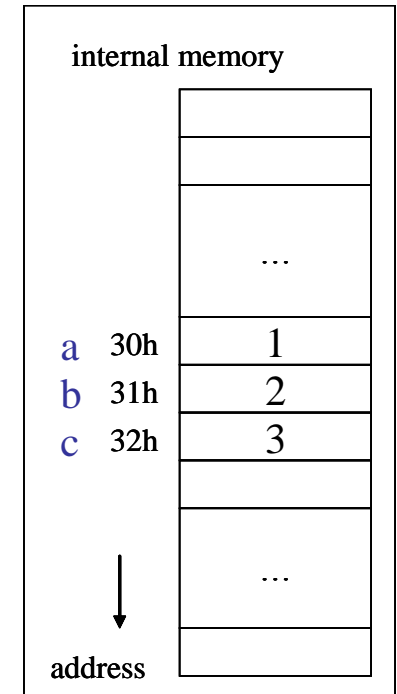
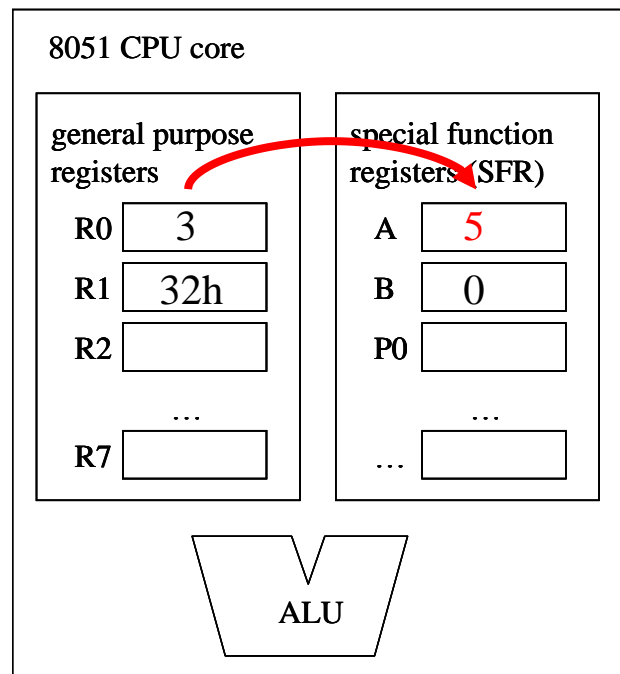
# Example: $d = a * b + c$

```
mov    R1, #30h    ; R1=30
mov    A, @R1      ; A=mem[R1]
inc    R1          ; R1++
mov    B, @R1      ; B=mem[R1]
inc    R1          ; R1++
mov    R0, @R1     ; R0=mem[R1]=C
```

```
mul    AB          ; {A,B} = A*B
add    A, R0       ; A=A+R0
```

wait:

```
sjmp   wait
```







# Branch (Jump) Instruction

---

# What is a branch/jump instruction

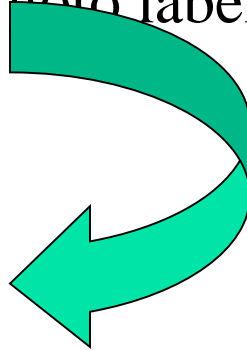
- the “goto” to control program execution path

the mul instruction  
won't be executed

```
add A, R1  
sjmp label_1; goto label_1  
mul AB
```

...

```
label_1:  
add A, R2
```





# Frequently used branch instructions of 8051

---

LJMP addr16	Long jump	3	4
SJMP rel	Short jump (from -128 to +127 locations relative to the following instruction)	2	3
JC rel	Jump if carry flag is set. Short jump.	2	3
JNC rel	Jump if carry flag is not set. Short jump.	2	3
JB bit,rel	Jump if direct bit is set. Short jump.	3	4
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3	4
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if the accumulator is zero. Short jump.	2	3
JNZ rel	Jump if the accumulator is not zero. Short jump.	2	3

for more, check <http://www.mikroe.com/en/books/8051book/ch3/>



# Conditional Branch Instructions

---

- JC: Jump if Carry=1
- JNC: Jump if Carry=0
- JZ: Jump if A=0
- JNZ: Jump if A!=0
- DJNZ Rn, location
  - $R_n = R_n - 1$
  - jump if  $R_n \neq 0$

# How conditional branch works in 8051?



---

- an arithmetic instruction sets bits in **PSW**
- the conditional branch checks bits in PSW to determine whether to jump or not

# How conditional branch works in 8051?

- an arithmetic instruction sets bits in **PSW**
- the conditional branch checks bits in PSW to determine whether to jump or not

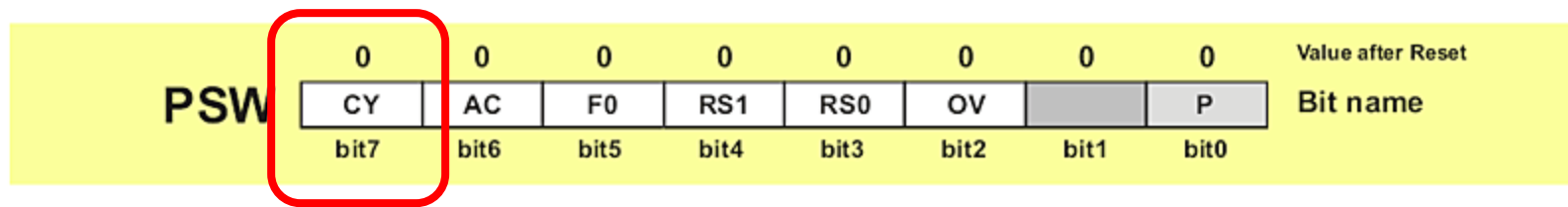
a register in SFR

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8									CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

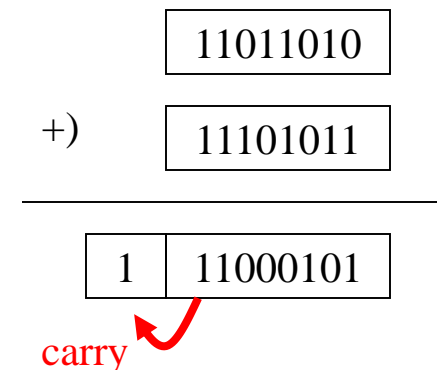
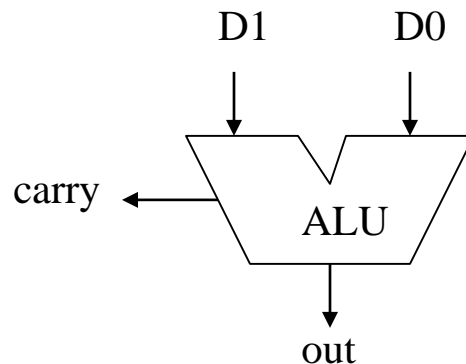
Bit-addressable Registers

# How conditional branch works in 8051?

- an arithmetic instruction sets bits in **PSW**
- the conditional branch checks bits in **PSW** to determine whether to jump or not



carry flag set by ALU





# Example of conditional branch instruction

---

- *JC label*
  - if (C==1) goto *label*
- *JNC label*
  - if (C==0) goto *label*





# In-Class Exercise

---

- write the program:

```
if (R0+R1>0xff)
    A = 0xff;
else
    A = R0+R1
```



# Example: using JNC

---

```
if (R0+R1>0xff)
    A = 0xff;
else
    A = R0+R1
```



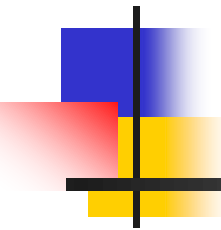
```
A = R0;
A = A+R1; //implicitly set C
if (C==0) goto exit;
A = 0xff;

exit:
...
```



```
mov    A, R0
add     A, R1
JNC     exit
mov     A, #ffh

exit:
...
```



# Example: a simple for loop

---



# A useful instruction

---

- DJNZ Rn, location
  - $R_n = R_n - 1$
  - jump if  $R_n \neq 0$



# Loop example

---

```
sum = 0;  
  
for (i=10;i>0;i--)  
    sum = sum+i;
```



```
A = 0;    //A is sum  
R0 = 10;  //R0 is i  
  
loop_start:  
    A = A+R0;  
    if (--R0) goto loop_start;
```



```
mov      A, #0  
mov      R0, #10  
  
loop_start:  
    add      A, R0  
    djnz     R0, loop_start
```



# Now you should be able to do your work

---

- write a program to compute

$$S = \sum_{i=0}^{N-1} A[i] * B[i]$$

- where  $A[i]$ ,  $B[i]$  are integer array (8-bit) in 8051's internal memory
- instructions you may use:
  - ADD (addition)
  - MUL (multiply)
  - MOV (move data)
  - DJNZ (decrement and jump if not zero)
- Check the instruction reference manual!



# How to build an executable program

---

Recap on the system programming  
before the practical guide



# The Next Lab: LED lights

---