.:: Architecture ::.

semaine n°3
Modes d'adressages et utilisation de la pile

Le MCS251- C => ASM

Passage du C à l'assembleur :

Si Condition		If1: CMP Condition
		J« cond. inverse » Else1
Alors		Then1:
Action1	==>	Action1
		JMP EndIf1
Sinon		Else1:
Action2		Action2
Finsi		EndIf1:

Le MCS251- C => ASM

Passage du C à l'assembleur :

Si Condition		If1:	CMP Condition J« cond. inverse » EndIf1
Alors		Then1:	
Action1	==>		Action1
Finsi		EndIf1:	

Le MCS251- C => ASM

Passage du C à l'assembleur :

Tantque Condition Faire		While: CMP Condition J« Cond. Inverse » EWhile
Action	=>	Action JMP While
Fait		EWhile:

Le MCS251-Modes Adressage

Interrogation écrite !

DR4	R4 00	R5 00	R6 05	R7 00
DR0	R0 00	R1 00	R2 00	R3 00

```
MOV   R0, 400Bh
MOV   WR2,TAB2
MOV   WR4,#TAB2
MOVZ WR6,R5

CMP   R0,R7
```

CY	OV	N	Z

NB :	05h	00:4000h
TAB1 :	40h	00:4001h
	00h	00:4002h
	00h	00:4003h
	01h	00:4004h
	FFh	00:4005h
	F8h	00:4006h
	00h	00:4007h
	19h	00:4008h
TAB2 :	01h	00:4009h
	03h	00:400Ah
	09h	00:400Bh
	05h	00:400Ch
	00h	00:400Dh
	00h	00:400Eh
	01h	00:400Fh
	03h	00:4010h

Le MCS251-Modes Adressage

Interrogation écrite !

DR4	R4 40h	R5 09h	R6 00h	R7 09h
DR0	R0 09h	R1 00h	R2 01h	R3 03h

```
MOV   R0, 400Bh
MOV   WR2,TAB2
MOV   WR4,#TAB2
MOVZ WR6,R5

CMP   R0,R7
```

CY	OV	N	Z
0	0	0	1

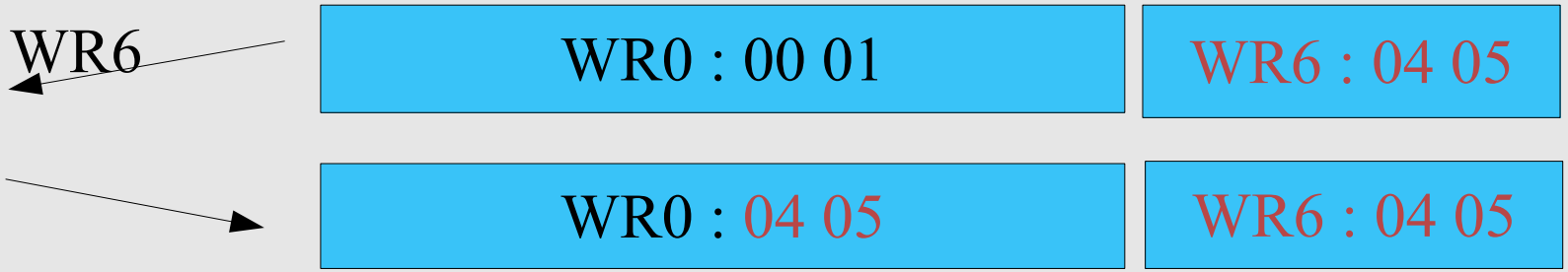
NB :	05h	00:4000h
TAB1 :	40h	00:4001h
	00h	00:4002h
	00h	00:4003h
	01h	00:4004h
	FFh	00:4005h
	F8h	00:4006h
	00h	00:4007h
	19h	00:4008h
TAB2 :	01h	00:4009h
	03h	00:400Ah
	09h	00:400Bh
	05h	00:400Ch
	00h	00:400Dh
	00h	00:400Eh
	01h	00:400Fh
	03h	00:4010h

Le MCS251-Modes Adressage

Le mode implicite : Registre à registre

- La source et la destination sont des registres
  - MOV R0, R1
    ADD R2, R4

- MOV WR0, WR6



Le MCS251-Modes Adressage

Le mode direct : Registre à mémoire

- Charger une variable mémoire dans un registre :
  - MOV R0, index

R0	R1	R2	R3
00	C8	61	1A

  - MOV WR2, tableau

R0	R1	R2	R3
03	C8	00	00

WR2

	??	
	??	
	??	
tableau2	??	00:010B
	04	
	00	
	03	
	00	
	02	
	00	
	01	
	00	
	00	00:0102
tableau	00	00:0101
index	00	00:0100

Le MCS251-Modes Adressage

Le mode indirect par registre : Registre à mémoire

• Par rapport au mode direct, l'adresse mémoire n'est pas une donnée statique (étiquette) mais une donnée dynamique : un registre.

Lectures	Ecritures
Mov Rx,@WRx	Mov @WRx, Rx
Mov WRx,@WRx	Mov @WRx, WRx
~~Mov DRx,@WRx~~	~~Mov @WRx,DRx~~

- Dans ce mode, le registre précédé d'un « @ » correspond à un pointeur, comme en C.
- **Il faut distinguer la taille du pointeur ici (WR – 16b) de la taille des données lues/écrites, ici 8 puis 16 puis 32b.**

Le MCS251-Modes Adressage

Le mode indirect par registre : Registre à mémoire

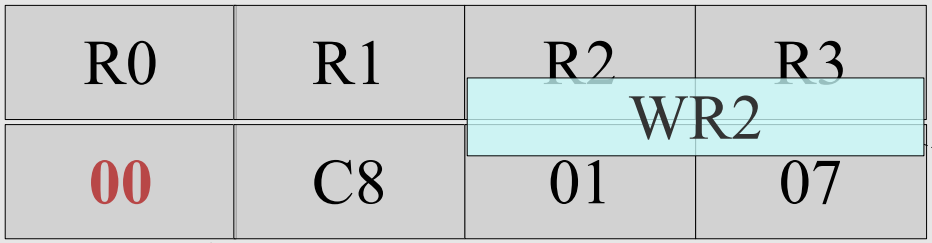
- **Un pointeur sera généralement de 16 bits (taille standard d'une adresse sur le MCS251)**

- Il existe toutefois des pointeurs 32b (puisque la mémoire totale peut faire 24bits) et des pointeurs courts 8b (pour l'accès aux registres SFR de configuration).

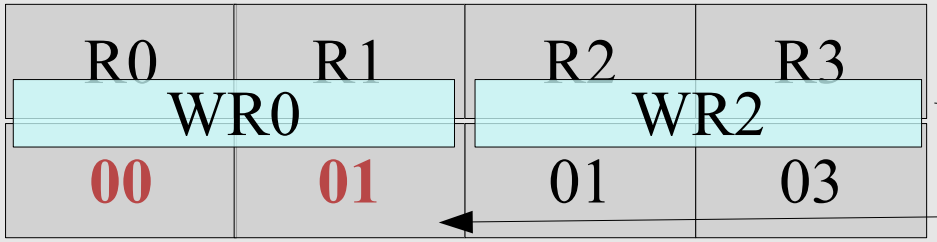
Le MCS251-Modes Adressage

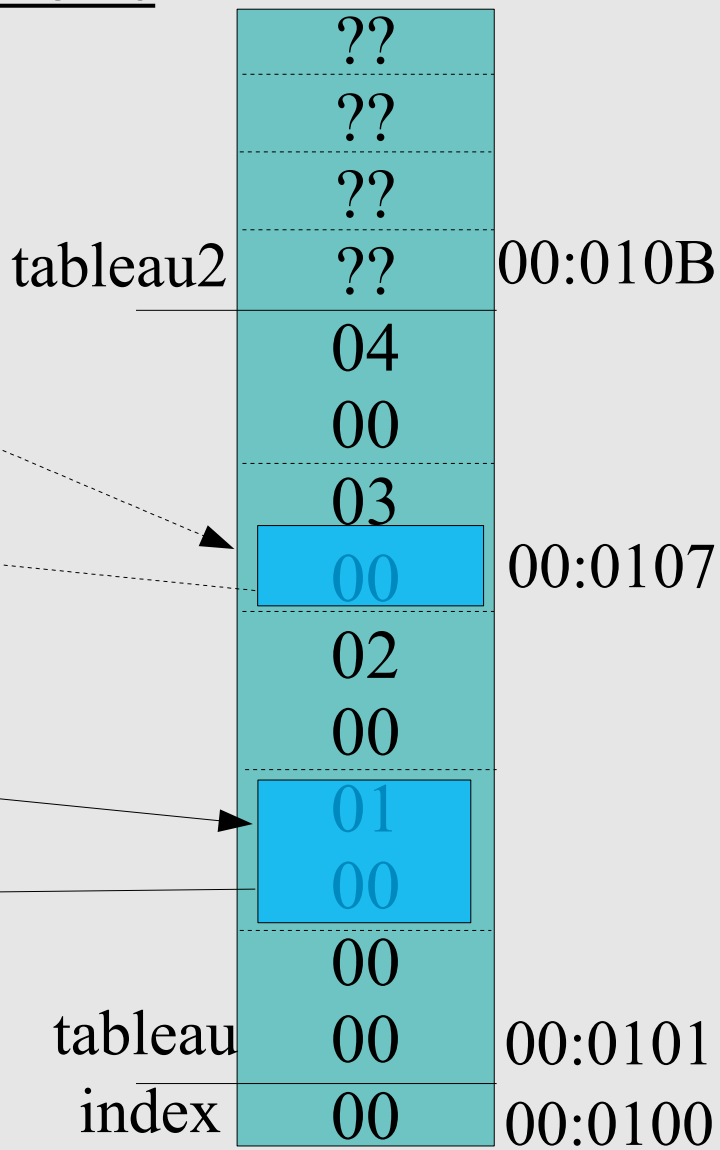
Le mode indirect par registre : Registre à mémoire

- MOV R0, @WR2



- MOV WR0, @WR2





Le MCS251-Modes Adressage

Le mode indexé par registre : Registre à mémoire

- Comme le mode Indirect par registre, mais il est possible en plus d'ajouter un offset (adresse fixe de départ)

LECTURES	ECRITURES
• Mov Rx, @WRx+*direct16*	Mov @WRx+*direct16*, Rx
• Mov WRx, @WRx+*direct16*	Mov @WRx+*direct16*, WRx
• ~~Mov DRx, @WRx+*direct16*~~	~~Mov @WRx+*direct16*, Drx~~

- Mode très utile pour le parcours d'un tableau : direct16 est alors l'adresse de début du tableau et WRx représente l'élément accédé dans le tableau.

Le MCS251-Modes Adressage

Le mode adressage bit – voir cours.

Le MCS251- C => ASM

Lire une donnée dans un tableau :

char tab1[4] = { 0,0,0,0 };

Deux façons en C :

1 – v = tab1[1];

2 – pointeur = tab;
pointeur++;
v = *pointeur;

Le MCS251- C => ASM

Lire une donnée dans un tableau :

char tab1[4] = { 0,0,0,0 };
L'adresse de l'élément 1 (le second) est tab1+1

Deux façons en C :

1 – v = tab1[1];

```
MOV   WR0,#1
MOV   R2,@WR0+tab1
```

2 – pointeur = tab;
pointeur++;
v = *pointeur;

Le MCS251- C => ASM

Lire une donnée dans un tableau :

char tab1[4] = { 0,0,0,0 };
L'adresse de l'élément 1 (le second) est tab1+1

Deux façons en C :

1 – v = tab1[1];

```
MOV   WR0,#1
MOV   R2,@WR0+tab1
```

2 – pointeur = tab;
pointeur++;
v = *pointeur;

```
MOV   WR0,#tab
ADD   WR0,#1
MOV   R2,@WR0
```

Rq : Dans les deux cas on utilise un pointeur qui indique soit l'adresse dans le tableau, soit le décalage à faire dans le tableau : WR0 (toujours un WR !!)

Le MCS251- C => ASM

Que se passe-t-il si on change la taille des éléments :

int tab1[4] = { 0x505,0x300,0x1FF, 0xF800 };
L'adresse est cette fois : TAB1+2*index

Deux façons en C :

1 – v = tab1[1];

2 – pointeur = tab;
pointeur++;
v = *pointeur;

TAB1 :

05h	00:4000h
05h	00:4001h
03h	00:4002h
00h	00:4003h
01h	00:4004h
FFh	00:4005h
F8h	00:4006h
00h	00:4007h
19h	00:4008h
01h	00:4009h
03h	00:400Ah
0Fh	00:400Bh
05h	00:400Ch
00h	00:400Dh
00h	00:400Eh
01h	00:400Fh
03h	00:4010h

Le MCS251- C => ASM

Que se passe-t-il si on change la taille des éléments :

int tab1[4] = { 0x505,0x300,0x1FF, 0xF800 };
L'adresse est cette fois : TAB1+2*index

Deux façons en C :

```
1 – v = tab1[1];        MOV   WR0,#1
                        ADD   WR0,WR0 ; 2*index
                        MOV   WR2,@WR0+tab1

2 – pointeur = tab;     MOV   WR0,#1
    pointeur++;         ADD   WR0,WR0; 2*index
    v = *pointeur;      ADD   WR0,#tab1
                        MOV   WR2,@WR0
```

TAB1 :

05h	00:4000h
05h	00:4001h
03h	00:4002h
00h	00:4003h
01h	00:4004h
FFh	00:4005h
F8h	00:4006h
00h	00:4007h
19h	00:4008h
01h	00:4009h
03h	00:400Ah
0Fh	00:400Bh
05h	00:400Ch
00h	00:400Dh
00h	00:400Eh
01h	00:400Fh
03h	00:4010h

Le MCS251- C => ASM

Généralisons :

Accéder à l'élément « i » tel que tab[i], se traduit ainsi

=> l'adresse est tab1 + (i*la taille d'un élément du tableau).
=> La multiplication est remplacée par des additions ou des décalages.

=> On utilisera plutôt la forme v=tab[i] lorsque l'index « i » est utilisé pour autre chose. Par exemple quand on a tab[min]

=> On utilisera plutôt la forme v = *pointeur lorsque le programme devra faire un parcours du tableau type *for (int i = 0 ; i < NB ; i++) v = tab[i] ...*

Remarque l'adresse correspondante à tab1 est aussi appelée **Offset du tableau**.

Le MCS251- La Pile

La pile :

=> Sert à passer sauvegarder les variables locales

=> Sert à sauvegarder les registres avant de les modifier dans une fonction

=> Sert à passer des paramètres à une fonction

=> Sert à stocker l'adresse de retour des fonctions (CALL)

=> La pile fonctionne comme une pile d'assiettes : seul le dernier élément ajouté à la pile peut être retiré.

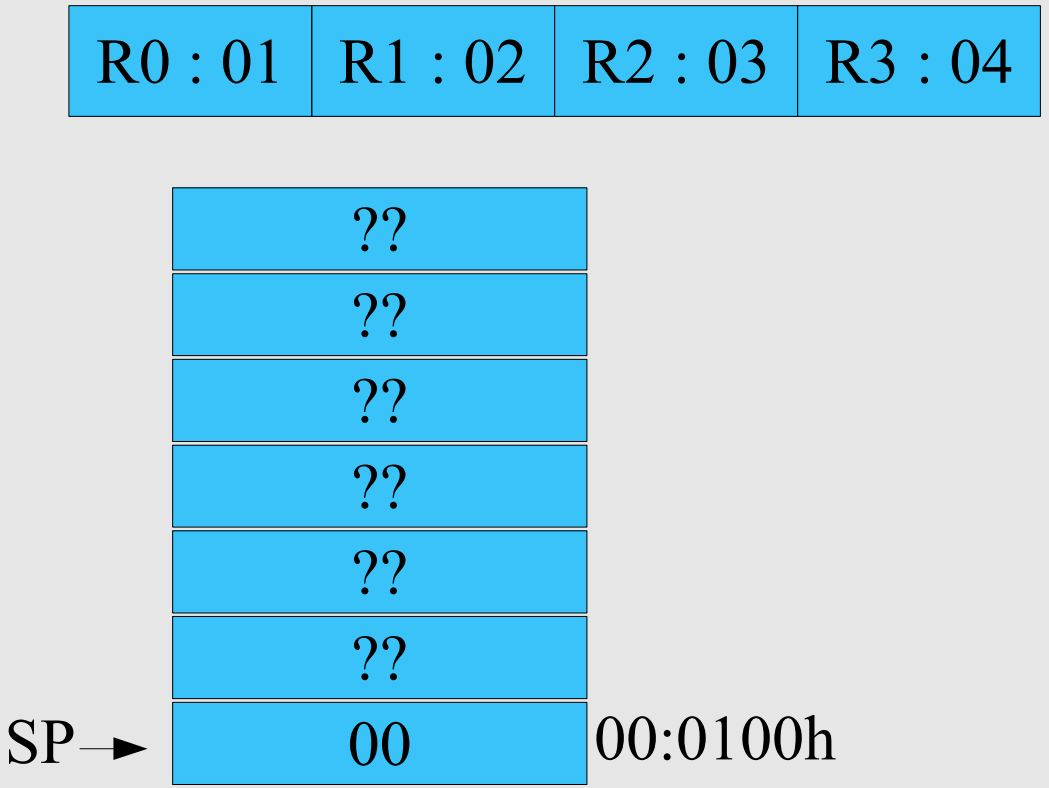
=> La pile est une zone de mémoire identifiée par un pointeur de pile : le registre SP. Ce registre indique le sommet de la pile.

Le MCS251- La Pile

La pile :

=> Ajouter des éléments dans la pile : PUSH

=> PUSH R0

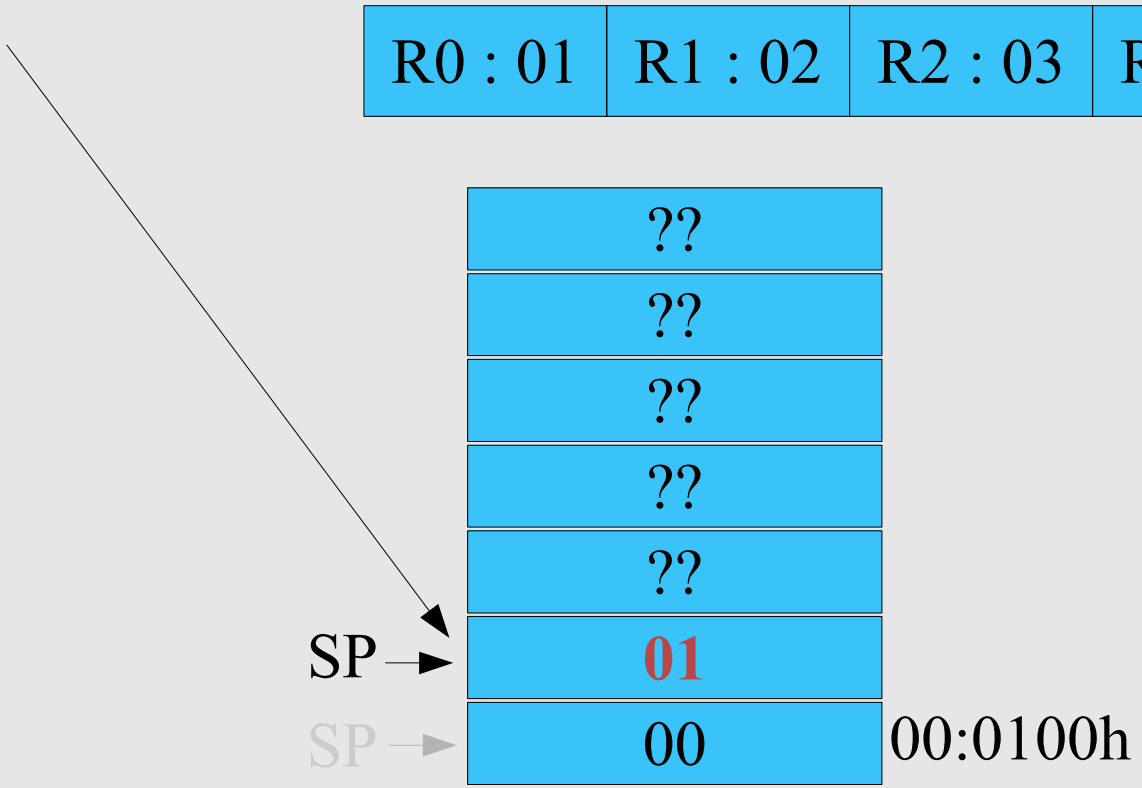


Le MCS251- La Pile

La pile :

=> Ajouter des éléments dans la pile : PUSH

=> PUSH R0

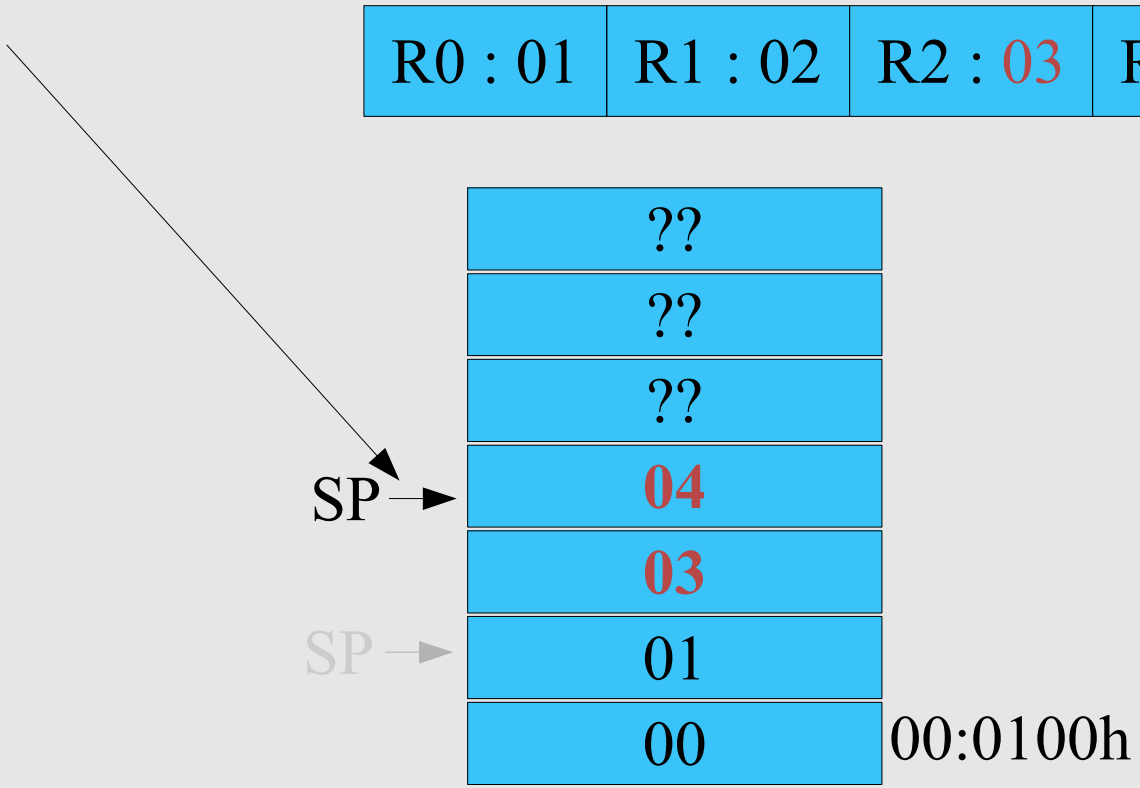


Le MCS251- La Pile

La pile :

=> Ajouter des éléments dans la pile : PUSH

=> PUSH WR2

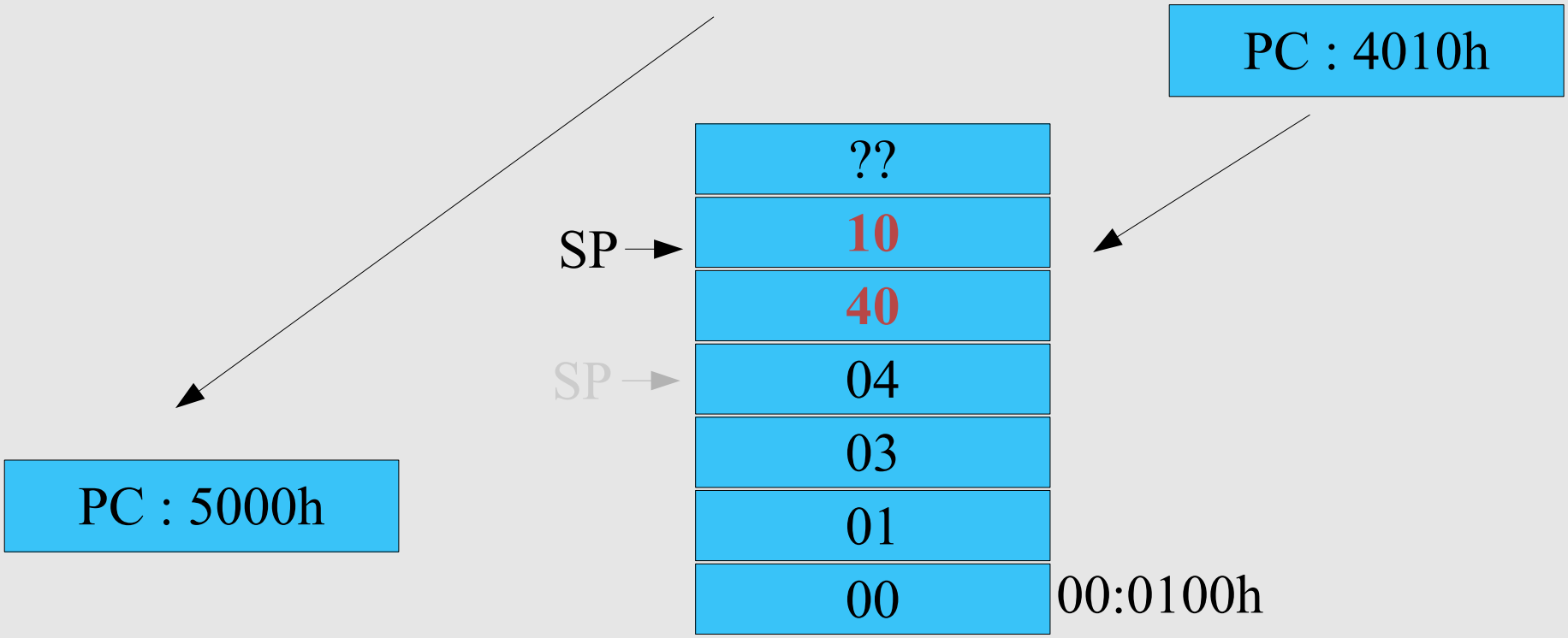


Le MCS251- La Pile

La pile :

=> Appel de fonctions

=> CALL fonction (adresse 5000h)

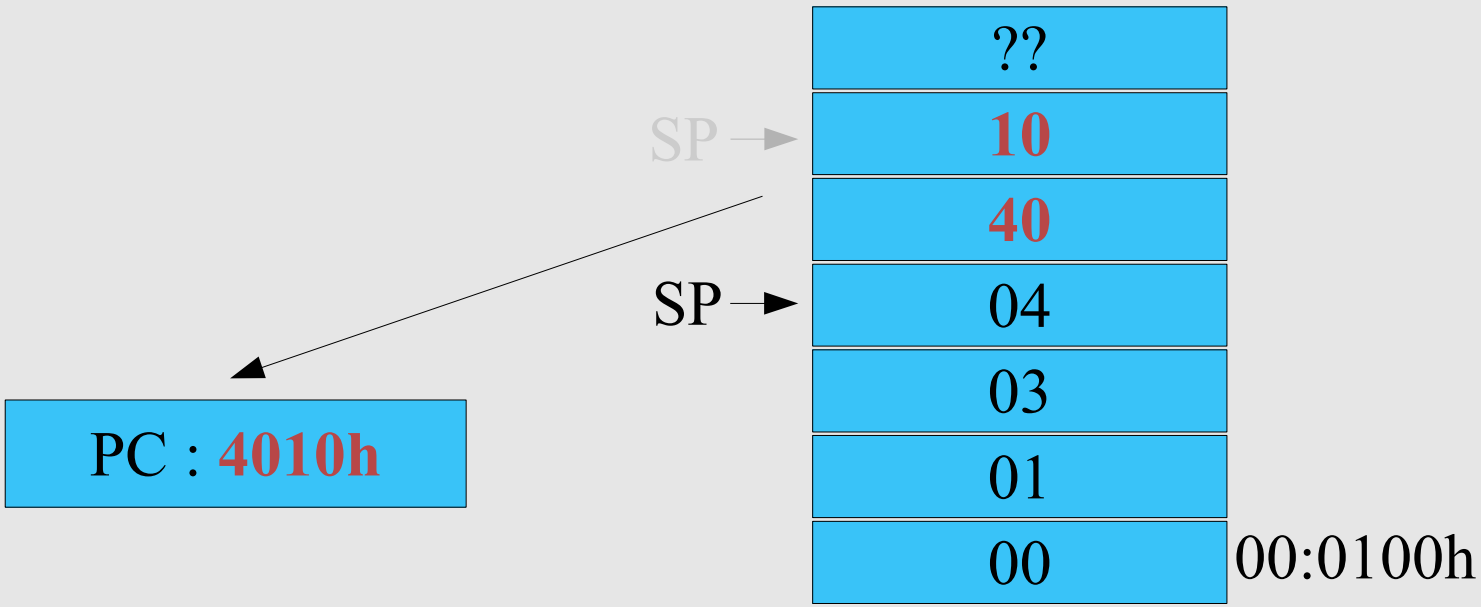


Le MCS251- La Pile

La pile :

=> Retour d'une fonction

=> RET

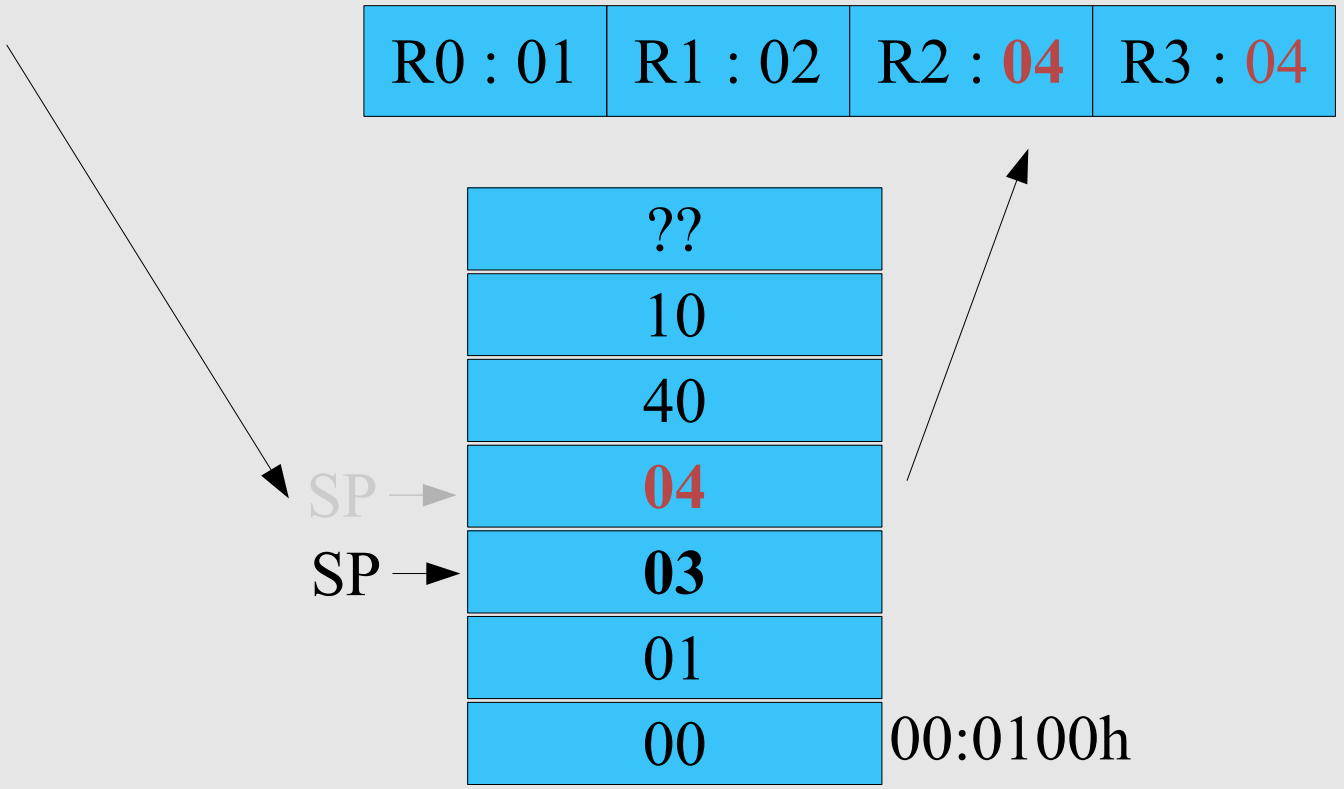


Le MCS251- La Pile

La pile :

=> Retirer des éléments dans la pile : POP

=> POP R2

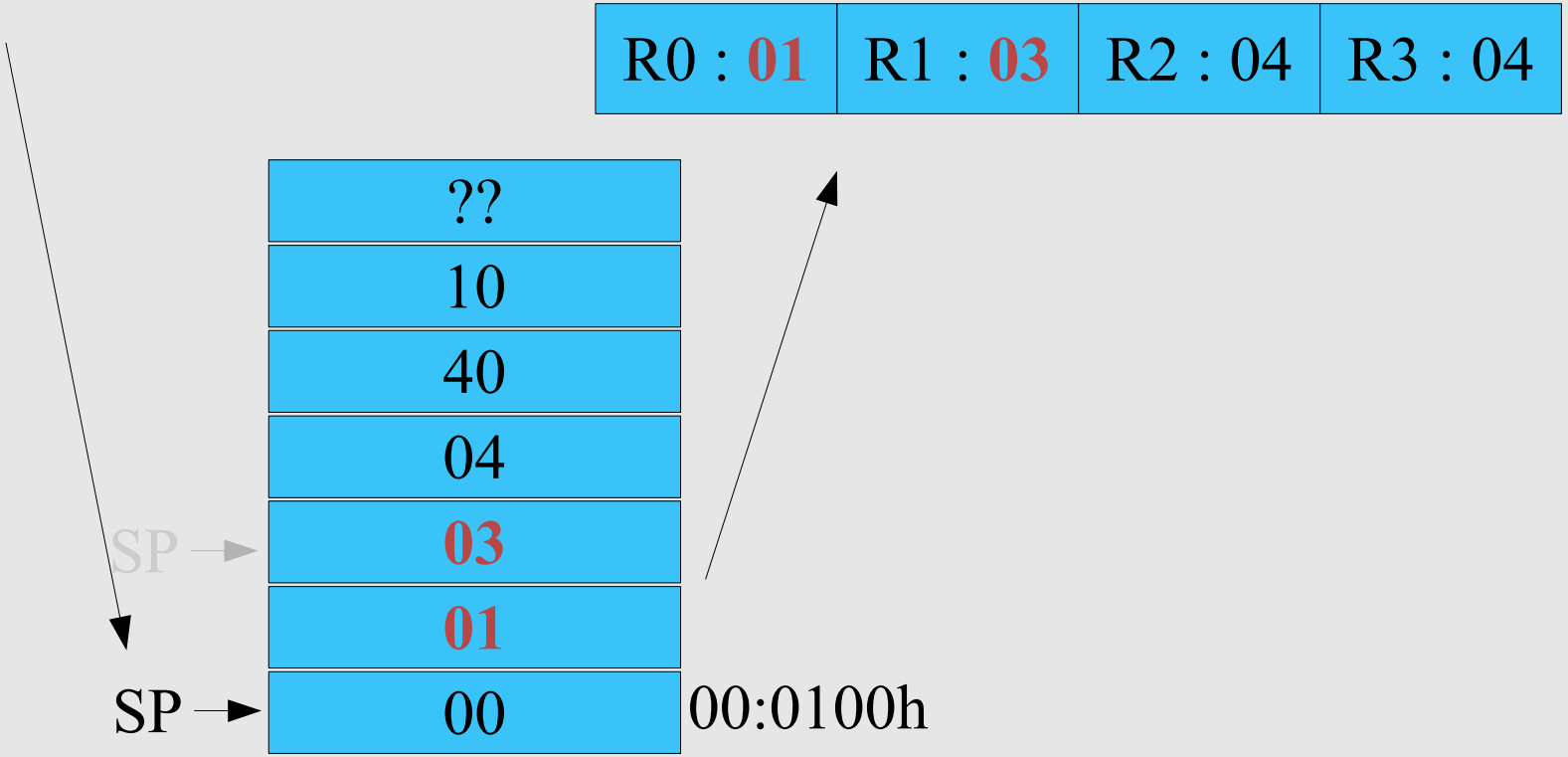


Le MCS251- La Pile

La pile :

=> Retirer des éléments dans la pile : POP

=> POP WR0



Le MCS251- C => ASM

Revenons sur la methode à employer pour accéder à un tableau

Le MCS251- C => ASM

Généralisons :

Accéder à tab[i]
pour char tab[];
R4 = tab[WR0]

```
MOV WR0,i
PUSH WR0

ADD WR0,#tab

MOV R4,@WR0

POP  WR0
```

Accéder à tab[i]
pour int tab[];
WR4=tab[WR0]

(variante)

```
MOV   WR0,i
PUSH  WR0

ADD   WR0,WR0

ADD   WR0,#tab

MOV   WR4,@WR0

POP   WR0
```

Accéder à tab[i]
pour long tab[];
DR4=tab[WR0]

```
MOV   WR0,i
PUSH  WR0

ADD   WR0,WR0
ADD   WR0,WR0
ADD   WR0,#tab

MOV   WR4,@WR0
MOV   WR6,@WR0+2
POP   WR0
```

Le MCS251- C => ASM

Généralisons avec l'utilisation des tab[i]:

Parcours de tab pour char tab[];

```
        MOV WR0,#0
WH:  CMP   WR0,NB
        JE     EWH

        PUSH  WR0

        ADD WR0,#tab

        MOV   R4,@WR0
        POP    WR0

        INC    WR0
        JMP    WH
EWH:
```

Parcours de tab pour int tab[];

```
        MOV WR0,#0
WH:  CMP  WR0,NB
        JE     EWH

        PUSH WR0
        ADD WR0,WR0

        ADD WR0,#tab

        MOV   WR4,@WR0
        POP   WR0

        INC    WR0
        JMP    WH
EWH:
```

Parcours de tab pour long tab[];

```
        MOV WR0,#0
WH:  CMP  WR0,NB
        JE    EWH

        PUSH  WR0
        ADD  WR0,WR0
        ADD  WR0,WR0

        ADD WR0,#tab

        MOV   WR4,@WR0
        MOV   WR6,@WR0+2
        POP   WR0

        INC    WR0
        JMP    WH
EWH:
```

Le MCS251- C => ASM

Généralisons avec l'utilisation des pointeurs:

Parcours de tab pour char tab[];

```
        MOV WR0,#tab
        XRL   R10,R10 ; R10 = 0
WH: CMP   R10,NB
        JE     EWH

        MOV   R4,@WR0
        INC     WR0,#1

        INC     R10
        JMP     WH
EWH:
```

Parcours de tab pour int tab[];

```
        MOV WR0,#tab
        XRL R10,R10
WH: CMP  R10,NB
        JE     EWH

        MOV   WR4,@WR0
        INC     WR0,#2

        INC     R10
        JMP     WH
EWH:
```

Parcours de tab pour long tab[];

```
        MOV WR0,#tab
        XRL R10,R10
WH: CMP  R10,NB
        JE    EWH

        MOV   WR4,@WR0
        MOV   WR6,@WR0+2
        INC     WR0,#4

        INC     R10
        JMP     WH
EWH:
```

Passage du C à l'assembleur :

Exemple : recherche du minimum d'un tableau

```
TAB1:  DB 3,2,12,1,25,40,18,8,9,4
NB:    DB 10        ; nombre d'éléments
```

```
for ( min = tab[0], i = 1 ; i < NB ; i++ ) if ( tab[i] < min ) min = tab[i];
```

ou plus simplement :

```
min = tab[0];
i = 1
While i < NB faire
    v1 = tab[i];
    Si v1 < min
    Alors min =  v1
    Finsi
    i <= i +1
Fait
```

Passage du C à l'assembleur :

Exemple : recherche du minimum d'un tableau

```
TAB1:  DB 3,2,12,1,25,40,18,8,9,4
NB:    DB 10        ; nombre d'éléments
```

```
for ( min = tab[0], i = 1 ; i < NB ; i++ ) if ( tab[i] < min ) min = tab[i];
```

ou plus simplement :

```
min = tab[0];
i = 1
While i < NB faire
    v1 = tab[i];
    Si v1 < min
    Alors min =  v1
    Finsi
    i <= i +1
Fait
```

Choix des registres

min <=> R0

i <=> WR2 (car il sera utilisé comme pointeur)

v1 <=> R4

Passage du C à l'assembleur :

Exemple : recherche du minimum d'un tableau

```
TAB1:  DB 3,2,12,1,25,40,18,8,9,4
NB:    DB 10        ; nombre d'éléments
```

ou plus simplement :

```
min = tab[0];
i = 1
While i < NB faire
      v1 = tab[i];
      Si v1 < min
      Alors min =  v1
      Finsi
      i <= i +1
Fait
```

Remplacement des variables

```
R0 = tab[0];
WR2 = 1;
While ( WR2 < NB ) {
      R4 = tab[WR2];
      if ( R4 < R0 ) {
            R0 = R4;
      }
      WR2 = WR2 + 1;
}
```

Le MCS251- C => ASM

Passage du C à l'assembleur :

ou plus simplement :

```
R0 = tab[0];
WR2 = 1;
While ( WR2 < NB ) {


    R4 = tab[WR2];




    if ( R4 < R0 ) {

        R0 = R4;
    }
    WR2 = WR2 + 1;

}
```

Traduction

```
        MOV   R0,_tab
        MOV   WR2,#1
WH1:    CMP   WR2,NB
        JGE   EWH1

        PUSH  WR2
        ADD   WR2,#_tab
        MOV   R4,@WR2
        POP   WR2

If1:    CMP   R4,R0
        JSGE  EIf1
        MOV   R0,R4
EIf1:
        INC   WR2
        JMP   WH1
EWH1:
```