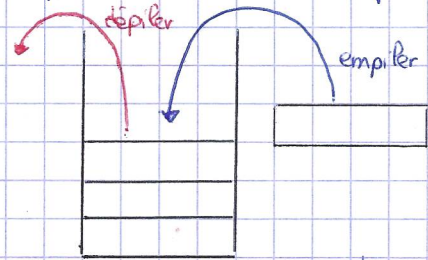


# Archi 1 Architecture des ordinateurs.

## I / Piles.

La pile est une structure de données qui se manipule avec les deux opérations suivantes :


- **empiler** : ajouter un élément au sommet de la pile.
- **dépiler** : enlever le sommet de la pile et le renvoyer



- L'inconvénient de cette structure c'est qu'on ne contrôle pas l'ordre dans lequel on peut accéder aux éléments.
- L'avantage de cette structure est que sa gestion ne nécessite que peu de ressources.
- En CRAPS, la gestion de la pile s'effectue grâce à l'adresse du sommet de la pile, qu'on appelle "pointeur de pile" (stack pointer, sp).

### Régistres en CRAPS.

servent à la prog.  
assembleur.

r0 = 0
r1 - r3

r20 = 1
...
r28 ret
r29 sp
r30 pc
r31 ir

r26 brk break

régistres qui servent  
à l'exécution des  
programmes.

return  
stack pointer  
program counter  
instruction register

### Pile en CRAPS.

	0
...	
	← sp - 1
	← sp
	512 : 0x200

Les instructions CRAPS de gestion de la pile sont :

- **push %r<sub>i</sub>** : empiler le contenu du registre %r<sub>i</sub>.
- **pop %r<sub>i</sub>** : dépile le sommet de la pile et le stocke dans le registre %r<sub>i</sub>.

Instructions équivalentes :

push %r<sub>i</sub>  $\Leftrightarrow$  sub %r29, 1, %r29  
st %r<sub>i</sub>, [%r29]

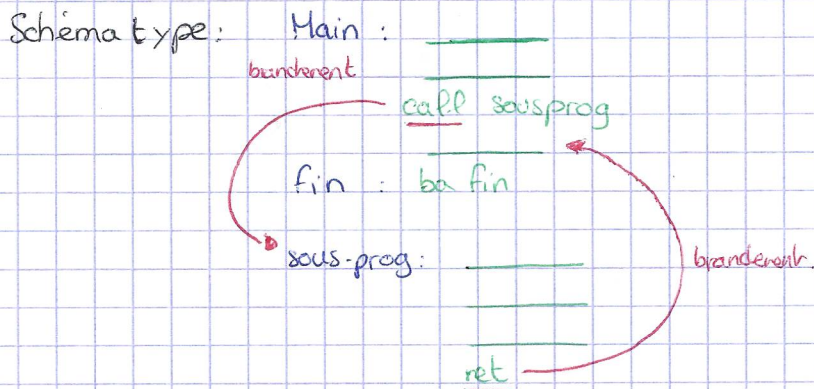
pop %r<sub>i</sub>  $\Leftrightarrow$  ld [%r29], %r<sub>i</sub>  
add %r29, 1, %r29

Note importante : Au début de l'exécution d'un programme, les registres sont initialisés à 0, y compris le %r29, le pointeur de pile.  
Il est donc nécessaire d'initialiser le pointeur de pile à l'adresse 0x200, c'est-à-dire en bas de la mémoire, avec : **set 0x200, %r29**



## II / Sous-programmes

- Il est parfois utile dans un même programme de réutiliser un segment de code sans avoir à le dupliquer. C'est la raison pour laquelle CRAPS implémente un système de sous-programmes.



- Pour gérer l'appel de sous-programmes en CRAPS, on utilise le registre %r28, ret (return) dans lequel on va stocker l'adresse de retour dans le programme appelant.

→ call sousprog  $\Leftrightarrow$  add %r30, %r0, %r28 // Sauvegarde de l'adresse de l'instruction suivante  
ba sousprog

→ ret  $\Leftrightarrow$  add %r28, 1, %r30

N.B.: Aux yeux du programme appelant, les registres ne doivent pas être modifiés dans un sous-programme

- Tous les registres modifiés dans un sous-programme doivent être sauvegardés au préalable et rétablis à la fin du sous-programme.

Pour cela, on utilise la pile:

```
sousprog:  push %r1
           push %r2
           _____
           pop  %r2
           pop  %r1
           ret
```

- Un sous-programme doit être accompagné d'un contrat, c-à-d d'une définition de ses paramètres et de ses valeurs de retour.

```
sousprog:  _____ // In: %r1 N
           _____ // Out: %r2 res.
           _____
           _____
```

- La remarque précédente ne s'applique pas aux registres définis dans le contrat comme contenant des valeurs de retour.



Exercice 1: Ecrire un sous-programme delay 100 ms qui va s'exécuter en 100 ms

La fréquence du processeur est de 12,5 MHz

Les instructions arithmétiques avec 2 registres opérands s'exécutent en 3 cycles  
 ———— 1 registre opérande et une constante en 4 cycles.  
 ———— de branchement s'exécutent en 3 cycles.

### Instructions arithmétiques:

add	
addcc	
sub	
subcc	%r1, %rj, %rk.
mulecc	
and	%r1, cste, %rk.
andcc	
or	
orcc	
xor	
xorcc	

Une instruction se terminant par "cc" modifie les indicateurs (flags).

Negative Zero overflow Carry

### Branchements conditionnels:

nom	Condition
ba	1
be	Z=1
bne	Z=0
bneg, bn	N=1
bpos, bnm	N=0

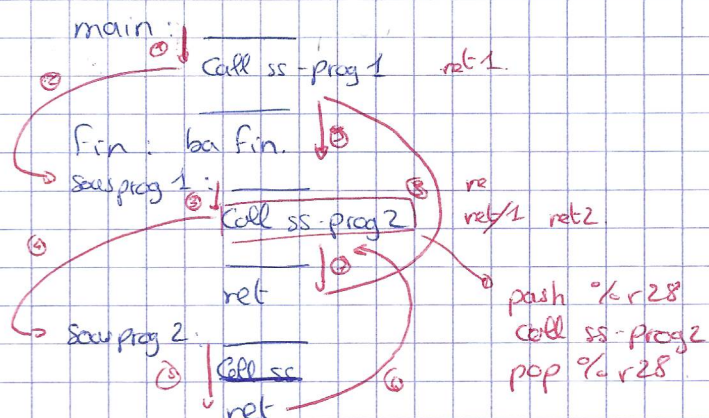
La syntaxe d'une instruction de branchement est:

bX label.

delay 100ms: push %r1  
 set 1250000, %r1.

loop: subcc %r1, 7, %r1  
 bpos loop  
 pop %r1  
 ret.

### ⚠ Aux appels de sous-programmes imbriqués:



Pour composer plusieurs appels de sous-prog il faut au préalable le contenu de %r28

push %r28  
 call ss-prog2  
 pop %r28



Exercice 2: Ecrire un sous-programme récursif qui calcule la factorielle d'un nombre  $N$

```
fact_rec : // In: N, %r1.  
           // Out: N!, %r2  
           push %r1      push %r28.  
           push %r3  
           subcc %r1, 1, %r1 // %r1 <= N-1  
           be  cas_terminel
```

```
           call fact_rec // %r2 <= fact_rec(N-1)  
           add %r1, 1, %r3 // %r3 <= N  
           umulcc %r3, %r2, %r2 // %r2 <= N x %r2  
           ba Fin.
```

```
cas_terminel: add %r0, %r20, %r2 // %r2 <= 1  
Fin: pop %r28  
     pop %r3.  
     pop %r1  
     ret.
```

```
fact_rec(N)  
  Si N=1 Alors  
    return 1  
  Sinon  
    return N x fact_rec(N-1)  
Fin Si
```