Cours 4 : Programmation en assembleur pour RISC

La pile

- Permet de stocker des éléments temporairement en mémoire :
  - Sauvegarde des registres / Restauration des registres
  - (Passage de paramètres)
  - (Sauvegarde de l'adresse de retour des sous-programmes)

Conventions

- Stockage : Adresse décroissante -> Optimisation de la mémoire
  - Représentation de la pile l'espace d'adressage
    - Adresse croissante (vers le bas). Dans ce cas le sommet de la pile est en haut de la représentation.
    - Adresse croissante (vers le haut). Dans ce cas le sommet de la pile est en bas de la représentation.
- Pile pointe
  - Première case vide
  - **Dernière case remplie**

Utilisation dans la pile

- Cf LB : i386
  - (Sauvegarde) PUSH %EAX
    - SP:=SP-1 puis MEM(SP):=EAX
  - (Restitution) POP %EAX
    - EAX:=MEM(SP) puis SP:=SP+1

Utilisation dans la pile

- Et avec le RISC ?
  - Dédier un registre en pointeur de pile ex : R1
  - PUSH %R5 devient
  - POP %R5 devient

Sous programme

- Cf LB : I386
  - Appel à un sous programme

  CALL sousProgramme
    - PUSH %PC (après lecture de l'instruction, donc PC pointe sur l'adresse de l'instruction qui suit CALL)
    - PC<-sousProgramme
  - Retour d'un sous programme

  RET
    - POP %PC

Sous programme

- Et sur le RISC?
  - L'adresse de retour n'est pas stocké dans la pile mais dans une registre (à sauvegarder éventuellement). Exemple R7
  - Appel à un sous programme
    - LI %R2,sousProgramme
    - BAL %R7,%R2
  - Retour d'un sous programme
    - BAINC -, %R7

Appel de fonction

- Exemple

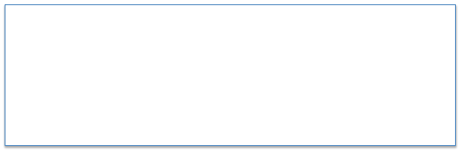
```
short f(short a, short b){
    return 2*a+b;
}

main () {
    short x,y,z;
    …
    x=f(3,4);
    …
    z=f(x,y);
    …
}
```

Traduction

-> Comment passer les paramètres (valeurs)?
Par registre ou par la pile. (Convention à définir)
-> Comment récupérer les résultats?
Par registre. (Convention à définir)
-> Comment matérialiser des variables locales?
Registres (*à sauvegarder avant dans la pile*)
(Zone dans la pile.)

Passage de paramètres

- I386 : dans la pile (Cf. LB)
- RISC : par registres (convention R3 et R4)
  Exercice: x=f(10,40);

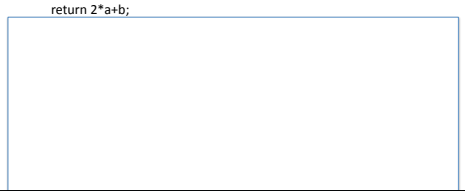


Valeur de retour de fonction

- Registre contient le résultat de la fonction (exemple R2).

Exercice (convention R3 désigne a, R4 désigne b)

```
short f(short a, short b){
    return 2*a+b;
```

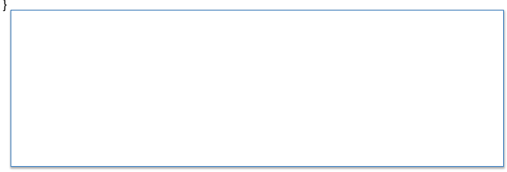


Variable locale

- Si peu de variables : Registres possibles
- Sinon stocké dans la pile.

Exemple (avec variable locale stockée dans la pile):

```
short f1(short n){
    short fn=n;
    …
    return fn;
}
```



Exemple de convention

- R0 : Registre de travail (temporaire)
- R1 : Pointeur de pile
- R2 : Valeur de retour de la fonction
- R3-R4 : Paramètres de la fonction (2 ici)
- R7 : Adresse de retour de la fonction

Condition

if (x!=y) instructions1;
else instructions2;
…

```
// x=R5, y=R6
                SUB R0,R5,R6                    //
                BRleq R0,decalage_instructions2 // if (x!=y)
                instructions1;                  // then instructions1;
                BRinc -,décalage_suite          //
Instructions2:                                  // else instructions2
                instructions2;
suite:
                …
```

Conditions multiples

```
If ((condition1) && (condition2))
    instructions1;
else
    instructions2;
…
```

Conditions multiples

```
If ((condition1) || (condition2))
    instructions1;
else
    instructions2;
...
```

Boucle

```
while (x!=y) {
    instuctions;
    y++;
}
...
```

Exercice

Traduisez en assembleur RISC (du processeur du projet) le programme suivant:

```
short fact(short n){
    short fn=n;
    if (n==1) fn=1;
    else fn=fact(n-1);
    return fn;
}
```