Assembleur : partie 4

Xavier Merrheim

Empilement des environnements

- L'exécution d'un programme commence par l'exécution de la fonction main.
- Ensuite la fonction main appelle d'autres fonctions et ainsi de suite.
- A chaque fonction est associé un environnement.
- On empile les différents environnements.

Contenu des environnements

- Un environnement contient :
- Les paramètres de la fonction
- Les variables locales de la fonction
- Des donnes permettant de créer une liste chainées des environnements (de taille variable) et de les empiler et dépiler.

- Une sauvegarde de l'adresse de retour de la fonction appelante

Gestion de la pile

- SP est le pointeur du haut de la pile.
- Quand on empile, les adresses diminuent.

Variable locale

```
int main()
{
int a,b,c;
a=10;

b=20;

c=a+b;
return 0;
}
```

Traduction arm

```
sub sp,sp,#12

mov r0,#10
str r0,[sp,#8]
mov r0,#20
str r0,[sp,#4]
ldr r0,[sp,#8]
ldr r1,[sp,#4]
add r2,r0,r1
str r2,[sp]
add sp,sp,#12
mov r0,#0
bx lr
```

Qu'est que mov r0,#0 et bx lr ?

- r0 contient la valeur renvoyée par un return
- lr est un registre contient l'adresse de retour de la fonction main
- bx lr continue le programme à cette adresse de retour, c'est à dire rend la mmain au système d'exploitation.

Appel d'une fonction

- Nous n'envisagerons que des fonctions des 4 paramètres maximum.
- Les paramètres sont mis dans les registres r0,r1, r2 et r3 dans l'ordre.
- A l'intérieur de la fonction ces paramètres sont sauvegardés dans la pile.
- On met dans la piile d'abord les pramètres de la fonction et ensuite les variables locales

Sauvegarde dans la pile

- L'adresse de retour lr dans être sauvegardée dans la pile en début de fonction.
- r0 contient la valeur du return de la fonction

- Unee fonction de doit pas modifier la valer des registres autres que r0 : il faut empiler ces registres au début de l'appel et les restaurer à la fin

- stfmd sp!,{r3,lr} empile lr puis r3
  ldfmd sp!,{r3,lr} depile r3 puis lr

Exemple

```
int main()
{
int a,b,c;
a=10;

b=20;

c=f(a,b);
return 0;
}
int f(int x, int y)
{int z;
z=x+y;

return z;
}
```

```
stmfd sp!,{lr}
sub sp,sp,#12

mov r0,#10
str r0,[sp,#8]
mov r0,#20
str r0,[sp,#4]
ldr r0,[sp,#8]
ldr r1,[sp,#4]
bl f
str r0,[sp]
add sp,sp,#12
ldmfd sp!,{lr}
bx lr
```

```
stmfd sp!,{r2,,lr}

sub sp,sp,#12

str r0,[sp,#8]
str r1,[sp,#4]
add r2,r0,r1
str r2,[sp]
mov r0,r2
add sp,sp,#12
ldmfd sp!,{r2,lr}
bx lr
```

Exercice

```
int main()
{ int a,b,c,d;
a=10;b=20;
c=f(a,b+3);
d=d+c-a;
}
```

```
int f(int u, int v}
{ int i,s;
s=0;
for(i=0;i<10;i++)
 {
s=s+i+u;
u=u+v;
}
return s;
}
```

Passage de pointeur

```
int main()
{
int a,b,c;
a=10;
b=20;
f(a,b,&c)
return 0;
}
```

```
void f(int x, int y, int * z)
{
int u;
u=x+y;
*z=u;
return u;
}
```

```
stmfd sp!,{lr}
sub sp,sp,#12

mov r0,#10
str r0,[sp,#8]
mov r0,#20
str r0,[sp,#4]
ldr r0,[sp,#8]
ldr r1,[sp,#4]
mov r2,sp
bl f
add sp,sp,#12
ldmfd sp!,{lr}
bx lr
```

```
stmfd sp!,{r3,r4,lr}
sub sp,sp,#16

str r0,[sp,#12]
str r1,[sp,#8]
str r2,[sp,#4]
add r3,r0,r1

str r3,[sp]
ldr r3,[sp,#4]
ldr r4,[sp]
str r4,[r3]

add sp,sp,#16
ldmfd sp!,{r3,r4lr}
bx lr
```

EXERCICE

```
int main()
{ int a,b,c,d;
a=90;b=70;
minmax(a,b,&c,&d);
return 0;
}
```

```
void minmax(int x,int y,int
*min,int * max)
{
if(a>b)
   {*min=b;*max=a;}
else
   {*min=a;*max=b;}
return 0;
}
```