L'ASSEMBLEUR – PARTIE 1

Xavier Merrheim

Programmation de haut niveau

- Aujourd'hui la plupart des programmeurs programment dans des langages de haut niveau comme le langage C, le java ou le C++
- Le processeur utilise un langage très rudimentaire appelé assembleur
- Il faut traduire les programmes dans un langage de haut niveau en assembleur pour que le processeur puisse les exécuter.

Compilation

- Le compilateur va traduire un programme source en assembeur puis en programme exécutable.
- La compilation est une science extrêmement complexe.

De nombreux assembleurs

- Il existe autant d'assembleurs que de familles de processeurs : x86, arm, 68 000 ...
- Le langage assembleur est très complexe avec de nombreux détails extrêmement techniques.
- Il est difficile de programmer en assembleur

Pour étudier l'assembleur

- Il ne s'agit pas de faire de vous des programmeurs assemblleurs opérationnels.
- L'objectif est une bonne culture générale permettant de mieux comprendre l'informatique
- Nous allons notamment apprendre à traduire des programmes en C en assembleur ARM

Architecture cible

- Nous allons étudier l'assembleur ARM 32 bits
- Le processeur contient 16 registres de 32 bits notés r0,r1, r2, ..., r15
- A coté du processeur se trouve un RAM mémorisant des cases mémoires de 32 bits.
- Chaque case mémoire porte un numéro appelé adresse

Exemple 1

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

Un programme bien étrange

- Variables globales : cet exemple en comporte car il est plus facile de gérer des variables globales que locales.

- Le main n'est pas une fonction : c'est volontaire car les fonctions sont complexes à traduire
- Ne faites pas cela en cours de C !!!

Traduction de l'exemple 1

```
L1:
.word a
.word b
.word c
.comm a,4,4
.comm b, 4,4
.comm c, 4,4
```

Traduction de l'exemple 1

```
mov r0,#10

ldr r1,L1
str r0,[r1]
mov r0,#20

ldr r1,L1+4
str r0,[r1]
ldr r0,L1
ldr r1,[r0]
ldr r0,L1+4
ldr r2,[r0]
add r1,r1,r2
ldr r0,L1+8
str r1,[r0]
```

mov

- mov r0,#10 : mets la constante 10 dans le registre r0 du processeur.

- Autre utilisation : mov r0,r1 : copie le contenu du registre r1 dans r0

ldr

- ldr r1,L1 : met dans r1 le contenu de la case mémoire L1 (lecture de la RAM)
- On peut ajouter une contante à l'adresse L1+4 ou L1+8
- ldr r1,[r0] met dans r1 le contenu de la case mémoire r0

Qui choisit la valeur de L1 ?

- Le système d'exploitation va charger le programme qui est sur le disque dur dans la RAM.

- A ce moment, il choisit la valeur de L1 et calcule toutes les adresses comme L1+4

str

- str r0,[r1] copie r0 dans la case mémoire numéro r1
- Il s'agit d'une écriture de la ram

Add et sub

- add r1,r2,r3 met dans r1 la somme de r2 et de r3
- Sub r1,r2,r3 : met dans r1 la valeur de r2-r3
- On peut écrire
  add r1,r1,#4
  sub r4,r5,#89

Différents modes d'adressage

- Par registre : r2
- Constante : #10
- Par adresse : L1
- Indirect [r0]
- Il y en a d'autres !

Exercice

Traduire en assembleur ARM le programme C

```
int a,b,c,d;
main()
{
a=10;b=65;
c=b-a+12;
d=c+b-98;
d=d+a-87;
}
```

Solution

```
L1:
.word a
.word b
.word c
.word d
.comm a,4,4
.comm b, 4,4
.comm c, 4,4
.comm d,4,4
```

```
mov r0,#10
ldr r1,L1
str r0,[r1]
mov r0,#65

ldr r1,L1+4
str r0,[r1]
ldr r0,L1+4
ldr r1,[r0]
ldr r0,L1
ldr r2,[r0]
sub r1,r1,r2
add r1,r1,#12
ldr r0,L1+8

str r1,[r0]
```

```
ldr r0,L1+8
ldr r1,[r0]
ldr r0,L1+4

ldr r2,[r0]
add r1,r1,r2
sub r1,r1,#98

ldr r0,L1+12
str r1,[r0]
ldr r0,L1+12
ldr r1,[r0]
ldr r2,L1
ldr r3,[r2]
add r1,r1,r3
sub r1,r1,#87
str r1,[r0]
```

Etiquette

- Une étiquette peut être placée devant n'importe quelle instruction en assembleur

  toto: mov r0,#10

- L'étiquette correspond à l'adresse en RAM où est codée l'instruction.

cmp

- cmp permet de comparer 2 éléments
- cmp r1,r2 calcule r1-r2 et effectue toutes les comparaisons possibles avec 0 : >0,<0,>=0,<=0,=0,!=0
- Le résultat des comparaisons est stocké dans le registe d'état.
- Le registre d'état est un registre du processeur que le programmeur assembleur ne peut pas modifier

Branchement non conditionnel : bra

- bra etiquette
- Cette instruction continue le programme à cette étiquette.

Branchement conditionnel

- Beq etiquette
  Si le résultat de la dernière comparaison =0 est vrai on va à l'étiquette sinon on continue à l'instruction suivante.
- Branch if equal

Autres branchements

bne branch if not equal
bgt branch if greater than
bge branch if greater or equal
blt branch if less than
ble branch if less or equal

Traduire un if

```
int a,b,c;
main()
{
a=10;b=20;
if(a+b>29)a=a+3;
c=b-a;
}
```

Traduction en assembleur ARM

```
L1:
.word a
.word b
.word c
.comm a,4,4
.comm b,4,4
.comm c,4,4
```

```
mov r0,#10
ldr r1,L1
str r0,[r1]
mov r0,#20
ldr r1,L1+4
str r0,[r1]
ldr r0, L1
ldr r1,[r0]
ldr r0,L1+4
ldr r2[r0]
add r1,r1,r2
cmp r1,#29
ble fin_if
ldr r0,L1
ldr1,[r0]
add r1,r1,#3
str r1,[r0]
fin_if : ldr r0,L1+4
ldr r1,[r0]
ldr r0,L1
ldr r2,[r0]
sub r1,r1,r2
ldr r0,L1+8
str r1,[r0]
```

Exercice

```
int a,b,c,d;
main()
{
a=10;b=67;c=12;
if(b-a<c){c=c+98;a=a+1;}
d=d+a-b;
}
```