

Résumé

Un plan de cours pour l'initiation à la programmation en langage d'assemblage en utilisant un processeur PowerPC. Le cours a été inauguré sur Bull Escala sous AIX, puis transposé sur un Mac, et enfin sur une machine virtuelle simulant un PowerPC.

Initiation à la programmation en langage d'assemblage

M Billaud

9 mai 2000

Table des matières

1 Objectifs du cours	2	A.2.5 Mouvement de registre registre	9
2 Etude d'un processeur : le PowerPC 604	2	A.3 Branchements et conditions . . .	10
3 Etude d'exemples simples	3	A.4 Opérations arithmétiques et logiques	10
3.1 Un calcul simple	3		
3.1.1 Le source C++	3		
3.1.2 Traduction en langage d'assemblage	3		
3.1.3 Le code en langage d'assemblage	3		
3.1.4 Analyse du code	3		
3.1.5 Exercices	3		
3.2 Si-alors-sinon	3		
3.2.1 Source	3		
3.2.2 Traduction	4		
3.2.3 Explications	4		
3.2.4 Exercices	4		
3.3 Boucles	4		
3.3.1 Le code	4		
3.3.2 Analyse	4		
4 Tableaux	5		
5 Exercices	5		
6 Passage de paramètre par valeur	5		
7 Conventions de passage de paramètres	6		
8 Utilisation de la pile	6		
8.1 Un autre exemple	6		
A Annexe : Jeu d'instructions	7		
A.1 Notations	7		
A.2 Mouvements de données	7		
A.2.1 Par octet	7		
A.2.2 Par demi-mot	8		
A.2.3 Par mot	9		
A.2.4 Chargement de constantes	9		

1 Objectifs du cours

- Montrer les différents éléments : jeu d'instruction, registres, pointeurs, adressages, etc.
- Réalisation des opérations de haut niveau (boucles, décisions, sous-programmes) au moyen des instructions élémentaires de la machine.
- Conventions de passage des paramètres
- Aperçu des techniques d'optimisation (droulage de boucle, etc.). Méthodologie de l'optimisation (recherche des parties coûteuses, mesures et comparaisons).

Approche basée sur l'étude du code fabriqué par les compilateurs.

2 Etude d'un processeur : le PowerPC 604

- Processeur RISC,
- 32 registres généralisés de 32 bits (+ 32 registres flottants de 64 bits et registres spéciaux)
- Registre de condition de 32 bits, découpé en 8 sous-registres de 4 bits.
- La plupart des instructions arithmétiques et logiques se réfèrent à 3 registres. Par exemple `add R4, R3, R8` additionne les contenus de `R4` et `R3` et met le résultat dans `R8`.

Signification : $R_8 \leftarrow (R_4) + (R_3)$

3 tude d'exemples simples

On ne regarde que des *fonctions feuille*, qui n'appellent pas d'autres fonctions.

3.1 Un calcul simple

3.1.1 Le source C++

```
int foo(int a,int b)
{
    return a-b+42;
}
```

3.1.2 Traduction en langage d'assemblage

On demande la traduction de ce programme `foo.cc` en langage d'assemblage, gr ce l'option `-S` de `g++`, avec les optimisations maximum.

```
$ g++ -S -O9 foo.cc
$
```

La traduction est mise dans `foo.s`

3.1.3 Le code en langage d'assemblage

```
.file "foo.cc"
.toc
.csect .text[PR]
gcc2_compiled.:
__gnu_compiled_cplusplus:
    .align 2
    .globl foo__Fii
    .globl .foo__Fii
.csect foo__Fii[DS]
foo__Fii:
    .long .foo__Fii, TOC[t0], 0
.csect .text[PR]
.foo__Fii:
    subf 4,4,3
    addi 3,4,42
    blr
LT..foo__Fii:
    .long 0
    .byte 0,9,32,64,0,0,2,0
    .long 0
    .long LT..foo__Fii-.foo__Fii
    .short 8
    .byte "foo__Fii"
_section_.text:
.csect .data[RW]
    .long _section_.text
    .file "foo.cc"
.toc
```

3.1.4 Analyse du code

La partie intéressante se limite en fait aux trois instructions qui sont apr s l'tiquette d'entr e de la fonction :

```
.foo__Fii:
    subf 4,4,3
    addi 3,4,42
    blr
```

Le reste se compose de *directives* que nous n'étudierons pas.

Commentons par la fin

- L'instruction `blr` (*Branch to link register*) fait revenir la fonction appelante. Celle-ci, lors de l'appel de `foo`, a mis dans l'adresse de retour dans le registre de liens. `blr` copie cette adresse dans le registre pointeur de programme (compteur ordinal).
- L'instruction `addi 3,4,42` additionne la valeur 42 au contenu du registre R4, et place le résultat dans R3. Par convention, c'est dans le registre R3 qu'une fonction doit placer la valeur retournée. C'est une addition avec une "valeur immédiate" : en effet dans l'instruction 42 n'est pas le numéro d'un registre, mais une valeur qui est utilisée telle quelle.
- `subf` est une soustraction (*subtract from*) entre registres " $R4 \leftarrow (R3) - (R4)$ " (attention l'ordre). l'entr e de la fonction `foo` les paramètres `a` et `b` sont donc reus respectivement dans R3 et R4.

3.1.5 Exercices

Essayez de traduire vous-même les deux fonctions qui suivent, puis comparez avec ce que donne le compilateur :

```
int plus (int n) { return n+1; }
int moins (int n) { return n-1; }
```

3.2 Si-alors-sinon

3.2.1 Source

```
int distance(int a, int b)
{
    if(a > b)
        return a-b;
    else
        return b-a;
}
```

3.2.2 Traduction

```

1  .distance__Fii:
2      cmpw 1,3,4
3      bc 12,5,L..2
4      subf 3,3,4
5      blr
6  L..2:
7      subf 3,4,3
8      blr

```

3.2.3 Explications

Structure générale : la première instruction (cmpw = Compare Word, ligne 2) compare les contenus des registres R3 et R4, c'est-à-dire les valeurs de a et b. La seconde (bc) est un branchement conditionnel : dans certaines circonstances (liées à la comparaison) l'exécution saute l'adresse L..2, sinon elle se poursuit en séquence l'instruction suivante.

On voit facilement que les lignes 7 et 8 correspondent à la partie "alors", le "sinon" tant que les lignes 4 et 5.

Fonctionnement détaillé de la comparaison : le registre de condition CR contient 8 sous-registres de condition CR0 à CR7, de 4 bits chacun. Le premier argument de cmpw indique que la comparaison des registres R3 et R4 positionnera le sous-registre de condition CR1, qui correspond aux bits 4 à 7 de CR (CR0 contient les bits 0 à 3, etc). Le premier bit d'un sous-registre de condition indique "inférieur", le second "supérieur", le troisième "égal".

Si $a < b$, les bits 4, 5 et 6 de CR vaudront donc 100, si $a = b$ on aura 001 et si $a > b$ 010.

Le branchement conditionnel comporte trois éléments : un critère (ici 12), un numéro de bit (ici 5) et une destination (L..2). Le critère est en réalité un masque binaire qui peut prendre des valeurs entre 0 et 15. Nous ne retiendrons ici que deux valeurs : 12 qui signifie "si le bit d'égale est 1" et 4 qui précise "si le bit d'inférieur est 0"¹.

La combinaison des deux instructions peut donc se lire "si R3 plus grand que R4, aller L..2".

1. Les autres valeurs permettent de tenir compte également du contenu d'un registre spécial (compteur)

3.2.4 Exercices

Écrire les fonctions "maximum de deux nombres" et "valeur absolue".

3.3 Boucles

3.3.1 Le code

```

int triangle(int n)
{
    int r;
    int k;
    r = 0;
    for (k=1; k<=n; k++)
        r += k;
    return r;
}

```

```

1  .triangle__Fi:
2      mr 9,3
3      li 0,1
4      cmpw 1,0,9
5      li 3,0
6      bclr 12,5
7  L..5:
8      add 3,3,0
9      addic 0,0,1
10     cmpw 1,0,9
11     bc 4,5,L..5
12     blr

```

3.3.2 Analyse

- On repère facilement le *corps de la boucle*, délimité par l'étiquette de la ligne 5 et le saut de la ligne 12.
 - Dans ce corps de boucle on doit trouver le cumul dans r, l'incrément de k, et la comparaison de k avec n. On en déduit facilement que r est dans R3, k dans R0, et n dans R9.
 - Remarquez le tour de passe-passe de la ligne 2, qui transfère n dans R9, ce qui permet d'employer le registre R3 pour r, qui sera le résultat.
 - La boucle for n'est pas traduite sous la forme "naturelle" d'une boucle tant-que k = 1
- ```

boucle:
 comparer k et n
 si > aller ...
 ...
 k++

```

```

 aller boucle
mais en “d pliant” le premier cas
 k = 1
 comparer k et n
 si > aller
boucle:

 k++
 comparer k et n
 si <= aller boucle

```

Ceci conomise l'ex cution d'une instruc-  
tion chaque tour de boucle (ici on aurait  
5 instructions au lieu de 4, soit une perte  
de 25 %).

- L'instruction `bclr` est un retour condi-  
tionnel.

```

.somtab__FPii:
 mr 10,3
 li 3,0
 cmpw 1,3,4
 mr 11,3
 bclr 4,4
 mr 9,3

L..5:
 lwzx 0,9,10
 addi 11,11,1
 cmpw 1,11,4
 addi 9,9,4
 add 3,3,0
 bc 12,4,L..5
 blr

```

## 4 Tableaux

```

int element(int t[], int i)
{
 return (t[i]);
}

```

```

.element__FPii:
 slwi 4,4,2
 lwzx 3,4,3
 blr

```

```

int somtab(int t[], int n)
{
 /* somme des n premiers
 l ments du tableau t */
 int k, s;
 s = 0;
 for (k=0; k<n; k++)
 s += t[k];
 return s;
}

```

## 5 Exercices

### 6 Passage de param tre par valeur (C++)

```

void incrementer(int &n)
{
 n++;
}

```

```

.incrementer__FRi:
 lwz 0,0(3)
 addic 0,0,1
 stw 0,0(3)
 blr

```

```

void echanger(int &a, int &b)
// pont aux nes
{
 int c;
 c=a;
 a=b;
 b=c;
}

```

```

.echanger__FRiT0:
 lwz 0,0(4)
 lwz 9,0(3)
 stw 0,0(3)
 stw 9,0(4)
 blr

```

## 7 Conventions de passage de param tres

Les param tres sont pass s dans les registres R3, R4, R5 etc. Si il y a un r sultat il est transmis dans R3.

- le contenu courant de R1 est sauv au sommet de ce nouveau bloc (attention la pile cro t vers le bas!)
- R1 est mis jour pour pointer sur ce nouveau bloc.

## 8 Utilisation de la pile

Pour voir comment se passe l'appel :

```
extern int foo(int a, int b);

int bar()
{
 return(bar(123,456));
}
```

```
.bar__Fv:
 mflr 0
 stw 0,8(1)
 stwu 1,-56(1)
 li 3,123
 li 4,456
 bl .foo__Fii
 cror 31,31,31
 addi 1,1,56
 lwz 0,8(1)
 mtlr 0
 blr
```

Le coeur de la fonction **bar** consiste appeler la fonction **foo**, avec les param tres 123 et 456. C'est ce qui est fait par les 3 instructions

```
li 3,123
li 4,456
bl .foo__Fii
```

L'instruction **cror 31,31,31** est une "non-op ration" qui ne fait rien. Sa pr sence est cependant obligatoire (contrainte de l' diteur de liens).

En entrant dans **bar**, le registre de liens contient l'adresse laquelle il faudra revenir. Le contenu de ce registre sera modifi par l'appel de **foo**, il faut donc en sauver le contenu avant d'effectuer cet appel, et le restaurer ensuite.

Pour la sauvegarde, manoeuvre en 2 temps : on transf re le registre de liens dans le registre R0, que l'on sauve ensuite dans la pile. (La restauration se fera de fa on sym trique).

Sur la pile dont le sommet est point par R1, la fonction **bar** se r serve ensuite un bloc de 56 octets :

### 8.1 Un autre exemple

```
int pascal(int n, int p)
{
 if (n==0) return 1;
 if (p==0) return 1;
 return (pascal(n-1,p)
 + pascal(n-1,p-1));
}
```

```
.pascal__Fii:
 mflr 0
 stw 28,-16(1)
 stw 29,-12(1)
 stw 30,-8(1)
 stw 31,-4(1)
 stw 0,8(1)
 stwu 1,-72(1)
 mr. 3,3
 mr 31,4
 bc 12,2,L..3
 cmpwi 1,31,0
 bc 12,6,L..3
 addi 29,3,-1
 mr 3,29
 mr 4,31
 bl .pascal__Fii
 mr 28,3
 mr 3,29
 addi 4,31,-1
 bl .pascal__Fii
 add 3,28,3
 b L..5

L..3:
 li 3,1

L..5:
 addi 1,1,72
 lwz 0,8(1)
 mtlr 0
 lwz 28,-16(1)
 lwz 29,-12(1)
 lwz 30,-8(1)
 lwz 31,-4(1)
 blr
```

## A Annexe : Jeu d'instructions

### A.1 Notations

|                  |                                                                   |
|------------------|-------------------------------------------------------------------|
| $D$              | le d placement                                                    |
| $CR$             | le Registre de Condition                                          |
| $Rx$             | registre entre r0 et r31                                          |
|                  | (RT pour "Target", RS pour "Source", RA et RB pour des op randes) |
| $(Rx)$           | le contenu du registre Rx                                         |
| $(RA 0)$         | si RA=0, alors 0, sinon le contenu de RA                          |
| $V, SI$          | quantit imm diate sign e sur 16 bits                              |
| $UI$             | quantit imm diate non sign e sur 16 bits                          |
| $rep(N, B)$      | champ form de $N$ r p tions du bit $B$                            |
| $mem(A, N)$      | $N$ octets en m moire partir de l'adresse $A$                     |
| $A  B$           | Le champ obtenu par concat nation de $A$ et $B$                   |
| $exts(V)$        | la valeur $V$ avec extension de signe sur 32 bits                 |
| $A\{i \dots j\}$ | les bits d'indice $i \dots j$ de $A$                              |
| $A \leftarrow B$ | affectation                                                       |

### A.2 Mouvements de donn es

#### A.2.1 Par octet

##### Chargement octet et z ro

lbz RT,D(RA)  $RT \leftarrow rep(24, 0) || mem((RA|0) + D, 1)$

##### Chargement octet et z ro avec mise- -jour

lbzu RT,D(RA)  $RT \leftarrow rep(24, 0) || mem((RA) + D, 1);$   
 $RA \leftarrow (RA) + D$

##### Chargement index octet et z ro

lbzx RT,RA,RB  $RT \leftarrow rep(24, 0) || mem((RA|0) + (RB), 1)$

##### Chargement index octet et z ro avec mise- -jour

lbzux RT,RA,RB  $RT \leftarrow rep(24, 0) || mem((RA) + (RB), 1)$   
 $RA \leftarrow (RA) + (RB)$

##### Rangement octet

stb RS,D(RA)  $mem((RA|0) + D, 1) \leftarrow (RS)24..31$

##### Rangement octet avec mise- -jour

stbu RS,D(RA)  $mem((RA) + D, 1) \leftarrow (RS)24..31$   
 $RA \leftarrow (RA) + D$

##### Rangement index octet

stbx RS,RA,RB  $mem((RA|0) + (RB), 1) \leftarrow (RS)24..31$

##### Rangement index octet avec mise- -jour

stbux RS,RA,RB  $mem((RA) + (RB), 1) \leftarrow (RS)24..31$   
 $RA \leftarrow (RA) + (RB)$

### A.2.2 Par demi-mot

#### Chargement demi-mot et z ro

lhz RT,D(RA)  $RT \leftarrow rep(16, 0) || mem((RA|0) + D, 2)$

#### Chargement demi-mot et z ro avec mise- -jour

lhzu RT,D(RA)  $RT \leftarrow rep(16, 0) || mem((RA) + D, 2)$   
 $RA \leftarrow (RA) + D$

#### Chargement index demi-mot et z ro

lhzx RT,RA,RB  $RT \leftarrow rep(16, 0) || mem((RA|0) + (RB), 2)$

#### Chargement index demi-mot et z ro avec mise- -jour

lhzux RT,RA,RB  $RT \leftarrow rep(16, 0) || mem((RA) + (RB), 2)$   
 $RA \leftarrow (RA) + (RB)$

#### Chargement alg brique demi-mot

lha RT,D(RA)  $RT \leftarrow exts(mem((RA|0) + D, 2))$

#### Chargement alg brique demi-mot avec mise- -jour

lhau RT,D(RA)  $RT \leftarrow exts(mem((RA) + D, 2))$   
 $RA \leftarrow (RA) + D$

#### Chargement alg brique index demi-mot

lhax RT,RA,RB  $RT \leftarrow exts(mem((RA|0) + (RB), 2))$

#### Chargement alg brique index demi-mot avec mise- -jour

lhaux RT,RA,RB  $RT \leftarrow exts(mem((RA) + (RB), 2))$   
 $RA \leftarrow (RA) + (RB)$

#### Rangement demi-mot

sth RS,D(RA)  $mem((RA|0) + D, 2) \leftarrow (RS)16..31$

#### Rangement demi-mot avec mise- -jour

sthu RS,D(RA)  $mem((RA) + D, 2) \leftarrow (RS)16..31$   
 $RA \leftarrow (RA) + D$

#### Rangement index demi-mot

sthx RS,RA,RB  $mem((RA|0) + (RB), 2) \leftarrow (RS)16..31$

#### Rangement index demi-mot avec mise- -jour

sthux RS,RA,RB  $mem((RA) + (RB), 2) \leftarrow (RS)16..31$   
 $RA \leftarrow (RA) + (RB)$



### A.2.3 Par mot

#### Chargement mot

lwz RT,D(RA)  $RT \leftarrow mem((RA|0) + D, 4)$

#### Chargement mot avec mise-~~-~~jour

lwzu RT,D(RA)  $RT \leftarrow mem((RA) + D, 4)$   
 $RA \leftarrow (RA) + D$

#### Chargement index mot

lwzx RT,RA,RB  $RT \leftarrow mem((RA|0) + (RB), 4)$

#### Chargement index mot avec mise-~~-~~jour

lwzux RT,RA,RB  $RT \leftarrow mem((RA) + (RB), 4)$   
 $RA \leftarrow (RA) + (RB)$

#### Rangement mot

stw RS,D(RA)  $mem((RA|0) + D, 4) \leftarrow (RS)$

#### Rangement mot avec mise-~~-~~jour

stwu RS,D(RA)  $mem((RA) + D, 4) \leftarrow (RS)$   
 $RA \leftarrow (RA) + D$

#### Rangement index mot

stwx RS,RA,RB  $mem((RA|0) + (RB), 4) \leftarrow (RS)$

#### Rangement index mot avec mise-~~-~~jour

stwux RS,RA,RB  $mem((RA) + (RB), 4) \leftarrow (RS)$   
 $RA \leftarrow (RA) + (RB)$

### A.2.4 Chargement de constantes

#### Chargement imm diat

li RT,V  $RT \leftarrow exts(V)$

#### Chargement imm diat en partie haute

lis RT,V  $RT(0..15) \leftarrow V$

En fait, ces deux instructions sont des mn moniques tendus qui repr sentent respectivement addi RT,0,V et addis RT,0,V

### A.2.5 Mouvement de registre registre

#### Chargement registre g n ral

mr RT,RS  $RT \leftarrow (RS)$   
 C'est un mn monique tendu pour or RT,RS,RS

#### Chargement partir du registre de lien (move from link register)

mflr RT  $RT \leftarrow (LR)$

#### Rangement dans registre de lien (move to link register)

mtlr RS  $LR \leftarrow (RS)$

### A.3 Branchements et conditions

BI (Bit In the CR) = num ro du bit de condition tester (entre 0 et 31).  
BO (Branch Option) = sp cification du test effectuer. Principales valeurs :  
— 4 : branchement si faux  
— 12 : branchement si vrai  
— 20 : branchement inconditionnel

#### Branchement

b adr

**Branchement avec lien (adresse suivante dans LR, Link Register)**

bl adr

#### Branchement conditionnel

bc B0,BI,adr

#### Branchement conditionnel avec lien

bcl B0,BI,adr

#### Branchement conditionnel au registre de lien

bclr B0,BI

#### Branchement conditionnel au registre de lien avec lien

bclrl B0,BI

#### Branchement au registre de lien

blr C'est une abr viation pour bclr 20,0

#### Combinaisons de bits du registre de condition

|                 |                                                                              |
|-----------------|------------------------------------------------------------------------------|
| crand BT,BA,BB  | $CR_{BT} \leftarrow CR_{BA} \text{and} CR_{BB}$                              |
| cror BT,BA,BB   | $CR_{BT} \leftarrow CR_{BA} \text{or} CR_{BB}$                               |
| crxor BT,BA,BB  | $CR_{BT} \leftarrow CR_{BA} \text{xor} CR_{BB}$                              |
| crnand BT,BA,BB | $CR_{BT} \leftarrow CR_{BA} \text{and} CR_{BB}$                              |
| crnor BT,BA,BB  | $CR_{BT} \leftarrow CR_{BA} \text{or} CR_{BB}$                               |
| creqv BT,BA,BB  | $CR_{BT} \leftarrow CR_{BA} \text{eqv} CR_{BB} (\text{eqv} = \text{notxor})$ |
| crandc BT,BA,BB | $CR_{BT} \leftarrow CR_{BA} \text{andnot} (CR_{BB})$                         |
| crorc BT,BA,BB  | $CR_{BT} \leftarrow CR_{BA} \text{ornot} (CR_{BB})$                          |

### A.4 Op rations arithm tiques et logiques

La notation [.] signifie que le mn monique peut tre suivi d'un point. Dans ce cas, l'op ration met jour le sous-registre de condition CR0.

#### Op rations arithm tiques

|                   |                                                    |
|-------------------|----------------------------------------------------|
| addi RT,RA,SI     | $RT \leftarrow (RA 0) + \text{exts}(SI)$           |
| addis RT,RA,SI    | $RT \leftarrow (RA 0) + (SI) \ll \text{rep}(16,0)$ |
| add[.] RT,RA,RB   | $RT \leftarrow (RA) + (RB)$                        |
| subf[.] RT,RA,RB  | $RT \leftarrow (RB) - (RA)$                        |
| neg[.] RT,RA      | $RT \leftarrow -(RA)$                              |
| mulli RT,RA,SI    | $RT \leftarrow (RA) * \text{exts}(SI)$             |
| mullw[.] RT,RA,RB | $RT \leftarrow (RA) * (RB)$                        |
| divw[.] RT,RA,RB  | $RT \leftarrow (RA) / (RB)$                        |

**Note :** la s quence suivante permet de calculer le reste d'une division enti re :

```
divw RT,RA,RB
mullw RT,RT,RB
subf RT,RT,RA
```

Les comparaisons positionnent les indicateurs d'un sous-registre de condition.

### Comparaisons

```
cmpwi CR,RA,SI comparaison de (RA) et exts(SI), r sultat dans CR
cmpw CR,RA,RB comparaison de (RA) et (RB)
```

### Op rations logiques

```
andi RA,RS,UI RA ← (RS)andrep(16,0)||UI
ori RA,RS,UI RA ← (RS)orrep(16,0)||UI
xori RA,RS,UI RA ← (RS)xorrep(16,0)||UI
andis RA,RS,UI RA ← (RS)andUI||rep(16,0)
oris RA,RS,UI RA ← (RS)orUI||rep(16,0)
xoris RA,RS,UI RA ← (RS)xorUI||rep(16,0)
and[.] RA,RS,RB RA ← (RS)and(RB)
or[.] RA,RS,RB RA ← (RS)or(RB)
xor[.] RA,RS,RB RA ← (RS)xor(RB)
nand[.] RA,RS,RB RA ← (RS)nand(RB)

nor[.] RA,RS,RB RA ← (RS)nor(RB)
eqv[.] RA,RS,RB RA ← (RS)eqv(RB)(eqv = notxor)
andc[.] RA,RS,RB RA ← (RS)andnot(RB)

orc[.] RA,RS,RB RA ← (RS)ornot(RB)
```

### Extension de signe

```
extsb[.] RA,RS RA ← exts(RS24..31)
extsh[.] RA,RS RA ← exts(RS16..31)
```

### D calages

```
slw[.] RA,RS,RB RA ← (RS) << (RB)
srw[.] RA,RS,RN RA ← (RS) >> (RB)
sraw[.] RA,RS,RB RA ← exts(RS0..31 - (RB))
 (d calage alg brique)
srawi[.] RA ← exts(RS0..31 - SI)
RA,RS,SI
```