Initiation à la programmation en langage d'assemblage

BILLAUD Michel, LÉPINE Jean-Michel, WOIRGARD Éric

LY Olivier

olivier.ly@labri.fr, eric.woirgard@ims-bordeaux.fr

7 septembre 2011

Plan du cours

Objectifs du cours

Le PowerPC G5

Exemples simples

Tableaux

Exercices

Passage de paramètres par adresse

Conventions de passage de paramètres

Utilisation de la pile

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 (déroulage 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.

Le PowerPC G5 (www.apple.com)

- Processeur RISC, 58 M transistors, 66 mm^2, fréq. horloge 1,8 GHz
- 32 registres banalisés de 64 bits (+ 32 registres flottants de 64 bits et registres spéciaux). Utilisé en 32 bits pour nos applications.
- 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 r8,r4,r3 additionne les contenus de R4 et R3 et met le résultat dans R8. Signification : R8 <— (R4) + (R3)

L'iMac G5 d'Apple

Que diriez-vous de faire tenir toute votre vie : vos musiques, photos, vidéos et e-mails, dans un ordinateur aussi élégant et performant qu'iPod ? Cela est désormais possible, avec le nouvel écran panoramique 17 ou 20 pouces de l'iMac G5. L'intégralité de l'ordinateur, dont la carte mère basée sur le processeur G5, le lecteur optique à chargement frontal, le disque dur, les haut-parleurs et le bloc d'alimentation, est nichée derrière l'écran. Adoptez un mode de vie résolument moderne à partir de 1 299 EUR TTC seulement.

Tout est dans l'écran

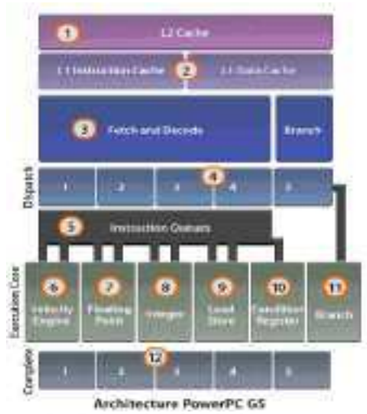
Les designers d'Apple ont éliminé le superflu, miniaturisé l'indispensable, boosté les performances et dissimulé le tout dans une pure perfection. iMac G5 tient suspendu grâce à un élégant pied en aluminium anodisé et son écran panoramique vous permet de retoucher des photos et de surfer sur

Noyau exécution processeur G5 (1/3)

Le PowerPC G5 renferme une puissance de traitement sans précédent sur un ordinateur de bureau. Ses circuits massivement parallèles sont capables de gérer plusieurs tâches différentes simultanément. Répondant au nom de noyau d'exécution, il s'agit de l'endroit où votre Mac réfléchit en permanence.

Sa conception est basée sur le processeur POWER4 64 bits d'IBM - couronné en 2001 par des analystes du Microprocessor Report, comme le Meilleur processeur pour station de travail/serveur, en reconnaissance de ses qualités exceptionnelles en matière d'innovation, de conception et d'implémentation de la technologie des semi-conducteurs. Avec deux unités en virgule flottante à double précision, une logique de prédiction élaborée et un bus frontal à large bande passante, le POWER4 est au coeur des célèbres serveurs IBM eBusiness.

A ce noyau d'exécution superscalaire à configuration de pipelines de pointe, Apple et IBM ont ajouté le Velocity Engine pour permettre d'exécuter toute application Mac OS X sans aucune difficulté. En outre, le PowerPC *G5* est doté d'innovations de



3 **Deux unités de calcul en nombres entiers**

Les unités en nombre entier réalisent des calculs simples en nombre entier - pour ajouter, soustraire et comparer - souvent utilisés dans des applications d'imagerie,

Noyau exécution processeur G5 (2/3)

traitement qui optimisent le flux des données et les instructions - le PowerPC G5 peut ainsi traiter 200 instructions simultanément.

(1) Cache N2
Les 512 Ko de cache N2 fournissent au noyau d'exécution un accès ultra-rapide de 64 Mo/s aux données et aux instructions.

(2) Cache N1
Les instructions sont pré-acheminées du cache N2 vers un immense cache N1 à accès direct de 64 Ko à 64 Go/s. En outre, 32 Ko de cache N1 peuvent pré-acheminer jusqu'à huit flux de données actives simultanément.

(3) Acheminement et décodage
A partir du cache N1, jusqu'à huit instructions par cycle d'horloge sont acheminées, décodées et partagées en opérations plus petites et plus simples à organiser. Cette préparation efficace optimise la vitesse de traitement alors que les instructions sont réparties dans le noyau d'exécution et que les données sont chargées dans les nombreux registres des unités fonctionnelles.

(4) Répartition
Avant que les instructions soient réparties dans les unités fonctionnelles, elles sont organisées par groupes de cinq au maximum. Dans le noyau à lui seul, le PowerPC G5 peut suivre jusqu'à 20 groupes en même temps ou bien 100 instructions individuelles. Ce suivi efficace permet au PowerPC G5 de gérer un nombre étonnant d'instructions "à la volée" : 20 instructions dans chacun des cinq groupes, ainsi qu'une centaine d'instructions aux différents stades d'acheminement, de décodage et de mise en file d'attente.

(5) File d'attente
Une fois le groupe d'instructions réparti dans le noyau d'exécution, il est séparé en instructions individuelles, qui passent à l'unité fonctionnelle correspondante. Chaque unité fonctionnelle possède sa propre file d'attente, où les instructions sont organisées pour être

vidéo et audio. Le PowerPC G5 est doté de deux unités en nombre entier capables d'exécuter un large éventail d'instructions simples et complexes impliquant à la fois des calculs 32 bits et 64 bits. Elles tirent parti, en outre, des registres et des chemins de données 64 bits du proceseur pour réaliser des calculs 64 bits en nombre entier en une seule passe.

(9) Chargement/Stockage
Alors que les instructions entrent dans la file d'attente, les unités de chargement/stockage chargent les données correspondantes du cache N1 vers les registres de données des unités qui traiteront les données. Une fois les données manipulées par les instructions, ces unités les stockent à nouveau sur le cache N1, N2 ou dans la mémoire principale. Chaque unité fonctionnelle est généreusement équipée de 32 registres de 128 bits sur le Velocity Engine et de 64 bits sur les unités en virgule flottante et sur les unités en nombre entier. Avec deux unités de chargement/stockage, le PowerPC G5 a la possibilité de remplir ces registres de données en permanence pour une efficacité de traitement optimale.

(10) Registre des conditions
Ce registre spécial de 32 bits récapitule l'état des unités en virgule flotante et en nombre entier. Le registre des conditions indique aussi les résultats des comparaisons effectuées et fournit un moyen de les tester comme conditions de branchement. En transmettant les informations entre l'unité de branchement et les autres unités fonctionnelles, le registre des conditions améliore le flux de données dans l'intégralité du noyau d'exécution.

(11) Logique de prédiction
Le PowerPC G5 énonce en général les réponses avant que les questions ne soient posées, utilisant la prédiction et des opérations de spéculation pour accroître son efficacité. De même qu'une personne qui finirait les phrases de son interlocuteur, la prédiction anticipe quelle instruction doit suivre, et l'opération de spéculation entraîne l'exécution de cette instruction. Si la

Noyau exécution processeur G5 (3/3)

traitées dans un ordre bien défini.

(6) Velocity Engine optimisé

Le PowerPC G 5 utilise un Velocity Engine à deux pipelines avec deux files d'attente indépendantes, des registres 128 bits dédiés et des chemins de données pour un flux d'instructions et de données efficace. Cette unité de traitement vectoriel accélère la manipulation des données en appliquant une instruction unique à plusieurs données en même temps, procédé connu sous le nom de traitement SIMD. Le Velocity Engine du PowerPC G5 utilise le même ensemble de 162 instructions que le PowerPC G4 et peut ainsi exécuter - et accélérer- les applications Mac OS X existantes déjà optimisées pour le Velocity Engine.

(7) Deux unités en virgule flottante à double précision

Deux unités en virgule flottante à double précision fournissent la précision nécessaire pour les calculs scientifiques très complexes. Si les processeurs 32 bits sont capables d'exécuter des calculs 64 bits à double précision en plusieurs cycles d'horloge, une unité à double précision sur un processeur 64 bits peut réaliser le même calcul en un seul cycle d'horloge. Les deux unités en virgule flottante à double précision permettent au G5 d'exécuter au moins deux calculs mathématiques 64 bits par cycle d'horloge. Cette opération accélère de façon spectaculaire les calculs complexes essentiels dans les applications de comparaison des génomes et dans de nombreux filtres utilisés pour manipuler des images 3D ou de la vidéo.

prédiction est correcte, le processeur fonctionne plus efficacement - puisque l'opération spéculative a exécuté une instruction avant qu'elle ne soit requise. Si la prédiction est erronée, le processeur doit effacer l'instruction et les données associées inutiles, créant ainsi un espace vide appelé "bulle de pipeline". Les bulles de pipeline réduisent les performances car le processeur marque un temps d'arrêt en attendant l'instruction suivante, comme quelqu'un qui perdrait du temps à écouter de fausses affirmations. Le G5 peut prédire l'étape suivante avec une précision pouvant atteindre 95 %, permettant au processeur d'utiliser efficacement chaque cycle de traitement.

(12) Phase finale

Une fois les opérations de données terminées, le PowerPC G5 recombine les opérations sur les données par groupes de cinq et les unités de chargement/stockage stockent les données dans le cache ou dans la mémoire principale pour les traitements ultérieurs.

Accueil > Matériel > PowerPC G5 > Noyau d'exécution

Etude d'exemples simples

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

Un Calcul Simple : le source C (foo.c)

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

Etude d'exemples simples

Traduction en langage d'assemblage

On demande la traduction pour un PowerPC604 32 bits de ce programme foo.c en langage d'assemblage, grâce à l'option -S de gcc, avec les optimisations maximum.

```
$ gcc-3.4  -mcpu=604  -S  -0 foo.c
```

Ce qui donne le fichier assembleur foo.s. Cette commande est lancée sur la machine de modèle iMac, dotée d'un processeur G5, de 512 Mo de RAM et de nom info-euphor. On accède à euphor par une connexion sécurisée (ssh info-euphor).

Etude d'exemples simples

Le code en langage d'assemblage (foo.s) :

```
.section __TEXT,...

.section __TEXT,...
.section __TEXT,...
        .align 2
        .align 2
        .gobl  _foo
. section __TEXT,...
        .align 2
__foo:
        subf r4,r4,r3
        addi r3,r4,42
        blr
```

Etude d'exemples simples

Analyse du code

La partie intéressante se limite en fait aux trois instructions qui sont après l'étiquette d'entrée de la fonction :

foo :

```
subf r4,r4,r3
addi r3,r4,42
blr
```

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

Etude d'exemples simples

Commençons par la fin ...

L'instruction blr *(Branch to link register)* fait revenir à la fonction appelante. Celle-ci, lors de l'appel de foo, a mis *l'adresse de retour* dans le *registre de liens*, blr copie cette adresse dans le registre pointeur de programme (compteur ordinal).

Etude d'exemples simples

L'instruction `addi r3,r4,42` 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.

Etude d'exemples simples

subf est une soustraction *(subtract from)* entre registres "R4 ← (R3) - (R4)" (attention à l'ordre). À l'entrée de la fonction foo les paramètres a et b sont donc reçus respectivement dans R3 et R4.

Etude d'exemples simples

Exercices (1)

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;  }
```

Etude d'exemples simples

Si-alors-sinon

Source

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

Etude d'exemples simples

Si-alors-sinon

Traduction en assembleur

```
1   .distance:
2           subf r0,r4,r3
3           cmpw crl,r3,r4
4           bgt- crl,Ll
5           subf r0,r3,r4
6   Ll:
7           mr r3,r0
8           blr
```

Etude d'exemples simples

Si-alors-sinon : explications

- la première instruction **(subf** = *SUBtract From*, ligne 2) est destinée à placer R3-R4 *(a-b)* dans R0
- 1' instruction suivante (cmpw = *CoMPare Word*, ligne 3) compare les contenus des registres R3 et R4, c'est-à-dire les valeurs de a et b.
- l'instruction de la ligne 4 **(bgt** = *Branch if Greater Than)* est un branchement conditionnel : dans certaines circonstances (liées à la comparaison : ici, si a>b) l'exécution saute à l'adresse Ll, sinon elle se poursuit en séquence à l'instruction suivante, **subf,** qui place b-a dans R0, puisque b est supérieur à a.
- R0 est copié dans R3 (mr) avant le retour (blr, ligne 8).

Etude d'exemples simples Si-alors-sinon : explications (suite)

On voit facilement que les lignes 6 à 8 correspondent à la partie "alors", le "sinon" étant codé par les lignes 5 à 8.

Fonctionnement détaillé de la comparaison (cmpw) : 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 de 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 "<", le second ">", le troisième "=". Les bits 4, 5 et 6 de CR vaudront donc 100 si a < b, 001 si $a = b$ et 010 si $a > b$.

Etude d'exemples simples Si-alors-sinon : explications (suite et fin)

Le branchement conditionnel (bgt-) comporte deux opérandes : un sous-registre de condition (CRi, i=0,7) et une adresse de destination (Ll). La forme générale de cette instruction est donnée dans la fiche *jeu d'instructions Assembleur PowerPC* :

bxx CR,adr Avec *xx* défini ci-après

La combinaison des deux instructions cmp et bgt- peut donc se lire : si R3 plus grand que R4, aller à Ll . Le signe moins dans bgt- indique qu'on suppose que, la plupart du temps, l'exécution s'effectue en séquence du branchement conditionnel, c'est-à-dire que R3<=R4.

Signification du code *xx* du branchement conditionnel

lt	*less than* : <
eq	*equal to* : =
gt	*greater than* : >
le	*less than or equal to (not greater than)*
ge	*greater than or equal to (not less than)*
ne	*not equal to* : ! =
nl	*not less than* : >=
ng	*not greater* : <=
z	*zero*
nz	*not zero*

Etude d'exemples simples

Si-alors-sinon

Exercices (1)

Utilisation du debugger PowerPC

Voir énoncé exercice.

Exercices (2)

Ecrire et tester les fonctions : "maximum de deux nombres" "valeur absolue" "modulo"

Etude d'exemples simples

Boucles

Le code C

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

Etude d'exemples simples

Boucles : le code assembleur

```
 1   _triangle:
 2           mr r0,r3
 3           li r3,0
 4           li r2,l
 5           cmpw cr7,r2,r0
 6           bgtlr- cr7
 7   L6:
 8           add r3,r3,r2
 9           addi r2,r2,l
10           cmpw cr7,r2,r0
11           bgtlr- cr7
tL2          b L6
```

Etude d'exemples simples

Boucles : analyse

- On repère facilement le *corps de la boucle*, délimité par l'étiquette de la ligne 7 et le saut de la ligne 12.
- Dans ce corps de boucle on doit trouver :
  - le cumul dans r
  - l'incrémentation de k
  - la comparaison de k avec n.

  On en déduit facilement que r est dans R3 (valeur retournée), k dans R2, et n dans R0.

Etude d'exemples simples

Boucles : analyse (suite)

- Remarquez le tour de passe-passe de la ligne 2, qui transfère n dans R0, 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 à ...
        …
        k++
        aller à boucle
```

I Etude d'exemples simples : analyse de 1 boucle (fin)

En fait, on "isole" le premier cas :

```
        k =  1
        comparer k et n
        si > aller à ....
boucle :
        ...
        k++
        comparer k et n
        si  <= aller à boucle
```

On économise ici une instruction sur 5 ...

- Les instructions **bgtlr** des lignes 6 et 11 sont des retours conditionnels aux *bonnes* prédictions de branchement (-).

Exercices sur les boucles (3)

1-Programmer et tester des suites générées à partir de la fonction de Syracuse, syr(n) qui renvoie :

- n/2 si n est pair
- sinon, 3*n+l

Vérifiez, notamment à l'aide d'un lanceur, si cette suite est bornée ou infinie ...en lançant syr(syr(syr(...syr(n)..)).

2-Exercice N°l du devoir du 13 juin 1998 *(fonction mystérieuse).*

Exercice supplémentaire : tester la différence de durée d'exécution entre une petite boucle et une longue boucle (indication : utilisez la commande time).

Tableaux

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

```
  élément :
        slwi  r4,r4,2
        lwzx  r3,r4,r3
        blr
```

Tableaux

```
int somtab(int t [], int n)
{
  /* somme des n premiers

     éléments du tableau t */
  int k, s;
  s = 0;
  for (k=0; k<n;  k++)
    s + = t [k] ;
  return s;
  }
```

Tableaux

```
1   _somtab:
2           mr r9,r3
3           li r3,0
4           li r2,0
5           cmpw cr7,r3,r4
6           bgelr- cr7
7   L6:
8           slwi r0,r2,2
9           lwzx r0,r9,r0
10          add r3,r3,r0
11          addi r2,r2,l
12          cmpw cr7,r2,r4
13          bgelr- cr7
14          b L6
```

Passage de paramètre par adresse

```
void incrementer(int *n)
{
    (*n)++;
}
```

```
_incrementer:
        lwz r2,0(r3)
        addi r2,r2,l
        stw r2,0(r3)
        blr
```

Passage de paramètre par adresse

```
void échanger (int *a, int *b)
// pont aux ânes
  {
  int c;
  c=*a;
  *a=*b;
  *b=c;
  }
```

Passage de paramètre par adresse : code généré

```
1   _echanger :
2           lwz r2,0(r3)
3           lwz r0,0(r4)
4           stw r0,0(r3)
5           stw r2,0(r4)
6           blr
```

Conventions de passage de paramètres (rappel)

Les paramètres sont passés dans les registres R3, R4, R5 etc… Si il y a un résultat, il est transmis dans R3.

Et si il y a plus de 32 paramètres ?

I Exercices sur les tableaux et le passage par adresse (4)

1. Reprendre les deux exemples précédents (**incrémenter** et **échanger)** et les exécuter à partir :
   - d'un lanceur écrit en C
   - des sources assembleur produits par gcc

   Pourquoi n'est-il pas tellement intéressant d'utiliser xor pour l'échange ?
2. Afficher les trente premières décimales de Pi (à partir d'un texte).
3. Ecrire une fonction booléenne **cherche (int t [] , int entier)** qui renvoie **vrai** si entier est dans t, **faux** sinon.

Utilisation de la pile

Visualisation de l'appel à une fonction

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

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

```
_bar:
mflr r0
stw r0,8(r1)
stwu r1,-80(r1)
li r3,123
li r4,456
bl L_foo$stub
lwz r0,88(r1)
addi r1,r1,80
mtlr r0
blr
```

Le cœur de la fonction bar consiste à appeler la fonction foo avec les paramètres 123 et 456 (instructions grisées)

```
mflr r0
stw r0,8(r1)
```

En entrant dans **bar**, le registre de lien contient l'adresse de retour. L'appel à **bar** va modifier LR.
Le sauvegarder en le transférant dans r0 puis dans la pile.
La restauration est symétrique `lwz r0,88(r1)` puis `mtlr r0`.
Sur la pile dont le sommet est pointé par R1, **bar** se réserve un bloc de 80 octets :

- le contenu courant de R1 est sauvé au sommet de ce nouveau bloc
- R1 est mis à jour pour pointer sur ce nouveau bloc (u :update).

Assembler Language Référence

A ce sujet, voir

`http ://www.nersc.gov/vendor_docs/ibm/asm/`,

Assembler Language Référence