Architecture et Programmation des mécanismes de base d'un système informatique (M2101)

BILLAUD Michel, WOIRGARD Eric

Février 2014

Plan du cours

- Un microprocesseur fictif
- Initiation à la programmation
  - Objectifs
  - Le powerPC G5
  - Exemples simples
  - Tableaux
  - Conventions de passage de paramètres
  - Utilisation de la pile
- Mécanismes de gestion des interruptions

Première partie

Un microprocesseur fictif

Contenu

- Exemple pédagogique
- Jeu d’instructions
- Les classes d’instructions
- Programmes
- Utilisation de mnémoniques
- Réservation de données
- Utilisation d’étiquettes
- Conventions d’écriture des sources

Un processeur fictif

Eléments

- machine à mots de 16 bits, adresses sur

  12 bits 1 accumulateur 16 bits

- compteur ordinal 12 bits
- jeu de 13 instructions sur 16 bits
  - arithmétiques : addition et soustraction 16 bits, complément à 2.
  - chargement et rangement directs et indirects,
  - saut conditionnel et inconditionnel,
  - appel de sous-programme,

  ...

Format des instructions

1 instruction = 16 bits. Format unique :

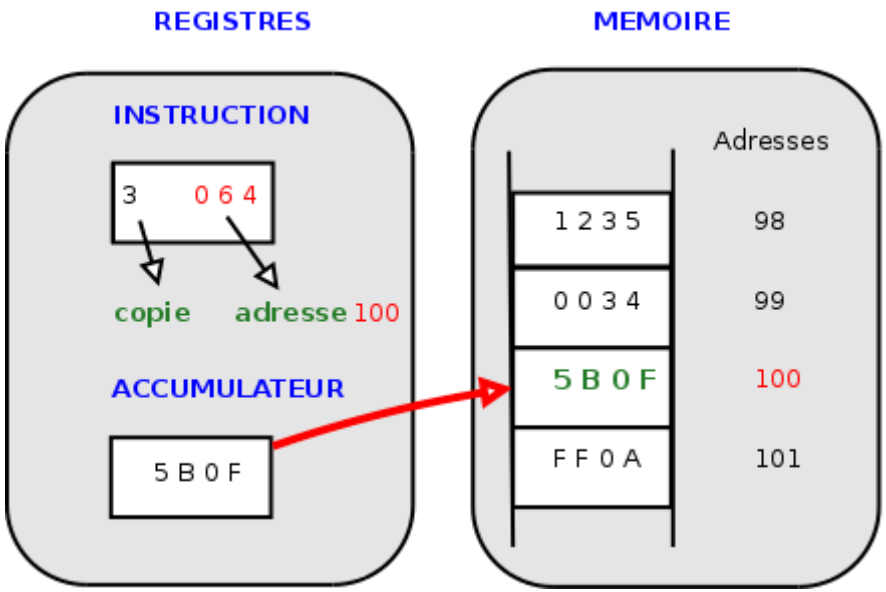
- code opération sur 4 bits (poids forts)
- opérande sur 12 bits

code 4 bits	opérande 12 bits
- - - -	- - - - - - - - - - - -

exemple

Le mot 0011 0000 0110 0100 (0x3064) peut représenter une instruction de code 0011 = 0x3 d'opérande 0000 0110 0100 = 0x064 (100 en décimal)

qui signifie "*ranger le contenu de l'accumulateur dans le mot mémoire d'adresse 100*"



Instruction ou donnée ?

- Le mot 0x3064 représente :
  - une instruction (code 3, opérande 100)
  - le nombre +12388 en binaire complément à 2
- La signification d'un mot en mémoire dépend de ce qu'on en fait.

Le jeu d'instructions

	Mnémonique	Description	Action	Cp =
0	loadi *imm12*	chargement immediat	Acc = ext(imm12)	Cp + 1
1	load *adr12*	chargement direct	Acc = M[adr12]	Cp + 1
2	loadx *adr12*	chargement indirect	Acc = M[M[adr12]]	Cp + 1
3	store *adr12*	rangement direct	M[adr12] = Acc	Cp + 1
4	storex *adr12*	rangement indirect	M[M[adr12]] = Acc	Cp + 1
5	add *adr12*	addition	Acc += M[adr12]	Cp + 1
6	sub *adr12*	soustraction	Acc -= M[adr12]	Cp + 1
7	jmp *adr12*	saut inconditionnel		adr12
8	jneg *adr12*	saut si négatif		si Acc < 0 alors adr12 sinon Cp+1
9	jzero *adr12*	saut si zero		si Acc==0 alors adr12 sinon Cp+1
A	jmpx *adr12*	saut indirect		M[adr12]
B	call *adr12*	appel	M[adr12] = Cp+1	adr12
C	halt 0	arrêt		

Les classes d'instructions

4 classes :

Transferts

pour charger une valeur dans l'accumulateur
ou placer le contenu de l'accumulateur en mémoire (load, store).

Arithmétique

addition et soustraction (add, sub)

Branchements

pour continuer à une adresse donnée (jump, call)

Divers

halt

Programmes

- Charger un programme, c'est remplir la mémoire avec un contenu : instructions et données.

- Exemple de programme)
- 0009 5005 6006 3007 C000 0005 0003 0000

- l'exécution commence (par convention) au premier mot :
  - le premier mot contient 0009 , qui correspond à "loadi 9" (charger la valeur immédiate 9 dans l'accumulateur)
  - le second mot contient 5005 , soit "add 5" (ajouter le mot d'adresse 5 à l'accumulateur)
  - ...

Utilisation de mnémoniques

Exemple de programme

0009 5005 6006 3007 C000 0005 0003 0000

Traduisons les 5 premiers mots en utilisant les codes mnémoniques des opérations

adresse	contenu	mnémonique	opérande
0	0009	loadi	9
1	5005	add	5
2	6006	sub	6
3	3007	store	7
4	C000	halt	0

En clair, ce programme charge la valeur 9 dans l'accumulateur, lui ajoute le contenu du mot d'adresse 5, retranche celui de l'adresse 6 et range le résultat à l'adresse 7. Et il s'arrête.

Réservation de mots

Exemple de programme

0009 5005 6006 3007 C000 0005 0003 0000

aux adresses 5, 6, et 7 on trouve les valeurs 5, 3 et 0,

adresse	contenu	directive	opérande
5	0005	word	5
6	0003	word	3
7	0000	word	0

La *directive* word indique la réservation d'un mot mémoire, avec sa valeur initiale.

Etiquettes symboliques

Il est commode de désigner les adresses par des noms symboliques, les étiquettes :

```
loadi   9
add     5
sub     6
store   7
halt    0
word    5
word    3
word    0
```

```
        loadi   9
        add     premier
        sub     second
        store   resultat
        halt    0
premier word    5
second  word    3
resultat word   0
```

Assemblage

Le programmeur écrit ses programmes en langage d'assemblage.
Le code source comporte:

- des instructions
- des directives de réservation
- des commentaires

qui font apparaître:

- des codes mnémoniques
- des étiquettes

La traduction de ce code source est faite par un assembleur.

Conventions d'écriture

Sur chaque ligne l'étiquette est facultative.

En colonne 1 si elle est présente.

Si elle est absente, la ligne commence par au moins un espace.

```
debut  loadi    100             # étiquette et instruction
         sub        truc        # instruction sans étiquette
```

si l'étiquette est seule, elle se rapporte au prochain mot

```
fin                             # étiquette seule
halt 0
```

Deuxième partie

Programmation

Instructions de base

Pour commencer :

chargement	immédiat	loadi valeur
chargement	direct	load adresse
rangement	direct	store adresse
addition	directe	add adresse
soustraction	directe	sub adresse
arrêt		halt 0

Attention ne pas confondre les opérandes immédiats et directs

loadi 100 charge *la constante 100* dans l'accumulateur

load 100 copie *le mot d'adresse 100* dans l'accumulateur

Programmation en assembleur

- **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

- Processeur RISC, 58 M transistors, 66 mm2, fréq. horloge 1,8 GHz
- 32 registres banalisés de 64 bits (+ 32 registres flottants de 64 bits et registres spéciaux). Utilisés 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.

Le PowerPC G5

- **Mono ou biprocesseurs à 2 GHz**. Chaque processeur PowerPC G5 64 bits intègre un Velocity Engine optimisé, deux unités en virgule flottante et une logique de prédiction de branchement.

Pour exécuter davantage de tâches plus rapidement, son architecture superscalaire à configuration de pipelines de pointe peut gérer une multitude d'opérations complexes en parallèle.

Mémoire cache

- la lecture et l'écriture d'informations
- Ce sont des RAM (Random Access Memory) mémoire dont le temps d'accès à l'information est le même quelque soit le mot sollicité ⇒ mémoires vives

Deux familles de RAM: statiques et dynamiques

•Les *RAM statiques*
⇒garantir la mémorisation de l'information aussi longtemps que l'alimentation électrique est maintenue sur la mémoire

•Les *RAM dynamiques* sont composées d'un ensemble de petits **condensateurs**, chacun pouvant recevoir ou restituer une charge électrique emmagasinée.
Inconvénient: la charge électrique mémorisée diminue avec le temps
⇒ les rafraîchir une fois toutes les quelques millisecondes → disposer d'un dispositif interne auto rafraîchissement: *les mémoires quasi-statiques*.

Mémoire cache

❖**Les unités centrales deviennent beaucoup plus rapides que les mémoires principales. Ainsi, lorsque l'unité centrale sollicite la mémoire, elle passe une bonne partie de son temps à attendre que la mémoire réagisse.**

❖**Le microprocesseur doit donc "attendre" la mémoire vive à chaque accès, on dit que l'on insère des "Wait State" dans le cycle d'horloge d'un micro.**

❖**le problème** **technologique : NON**
économique : Oui

*Il est possible de construire des mémoires aussi rapides que les unités centrales mais leur coût, pour des capacités de plusieurs mégaoctets serait prohibitif***.**

❖**Les choix :** *à l'exception des superordinateurs (performance>coût)*
→ **disposer d'une faible quantité de mémoire rapide associée à une quantité importante de mémoire relativement plus lente. Cette mémoire plus rapide est appelée mémoire cache, cache ou antémémoire.**

Mémoire cache

❖mémoire cache → mémoire vive statique
s'insère entre le processeur et la RAM dynamique.

❖Un contrôleur de mémoire cache est chargé de recopier les instructions et les données les plus fréquemment utilisées par le processeur dans le cache.

❖Le principe du cache repose sur deux constatations:
- en cours d'exécution d'un programme, lorsque le microprocesseur va chercher une instruction en mémoire il y a statistiquement de fortes chances pour que celle-ci se trouve à proximité de l'instruction précédente.
- de plus, les programmes contiennent un grand nombre de structures répétitives de sorte qu'ils utilisent souvent les mêmes adresses.

❖Gestion de la mémoire cache
le contrôleur du cache intercepte les adresses émises par le microprocesseur et dans un premier temps recopie le contenu d'un bloc entier de mémoire dans le cache de sorte qu'il y ait une forte probabilité que la mémoire cache contienne les prochaines instructions.
Ce principe permet au microprocesseur, via le contrôleur de cache, d'avoir 90% de chance d'obtenir l'information dans la mémoire cache donc sans "Wait State".

Mémoire cache

Principe de fonctionnement

❖Le dispositif est constitué de deux éléments principaux:

-la mémoire cache, constituée de RAM
-le contrôleur de mémoire cache comprenant

- **la gestion du cache**
- **un index d'adresses stockant les adresses des blocs contenus dans le cache**
- **un comparateur qui met en correspondance les adresses émises par le microprocesseur et celles contenues dans l'index quand il y a correspondance**

❖l'information est prise dans le cache sinon elle est transférée de la RAM et le contrôleur recopie tout un bloc dans le cache.

Mémoire cache

Différentes étapes du fonctionnement d'un cache:

1. Le microprocesseur demande une information (instruction ou donnée) I1 située à l'adresse A1 de la mémoire vive
2. Le contrôleur de mémoire cache intercepte la demande et examine sa table d'index pour vérifier si A1 y est présente, et donc si une copie de l'information se trouve dans le cache.
3. Si c'est le cas, l'information est délivrée au microprocesseur depuis la mémoire cache sans temps d'attente
4. Dans le cas contraire, le contrôleur de cache accède à l'information de A1 de la RAM, la délivre à l'unité centrale avec plusieurs temps d'attente et simultanément la recopie en mémoire cache en même temps qu'un bloc d'informations contiguës, parmi lesquelles I2, I3, I4 etc... et actualise sa table d'index.
5. Le microprocesseur réclame l'information suivante, il y a statistiquement 90% de chance pour que ce soit I2, donc présente dans le cache et délivrée dans l'UC sans temps d'attente.

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.

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.

Acheminement et décodage

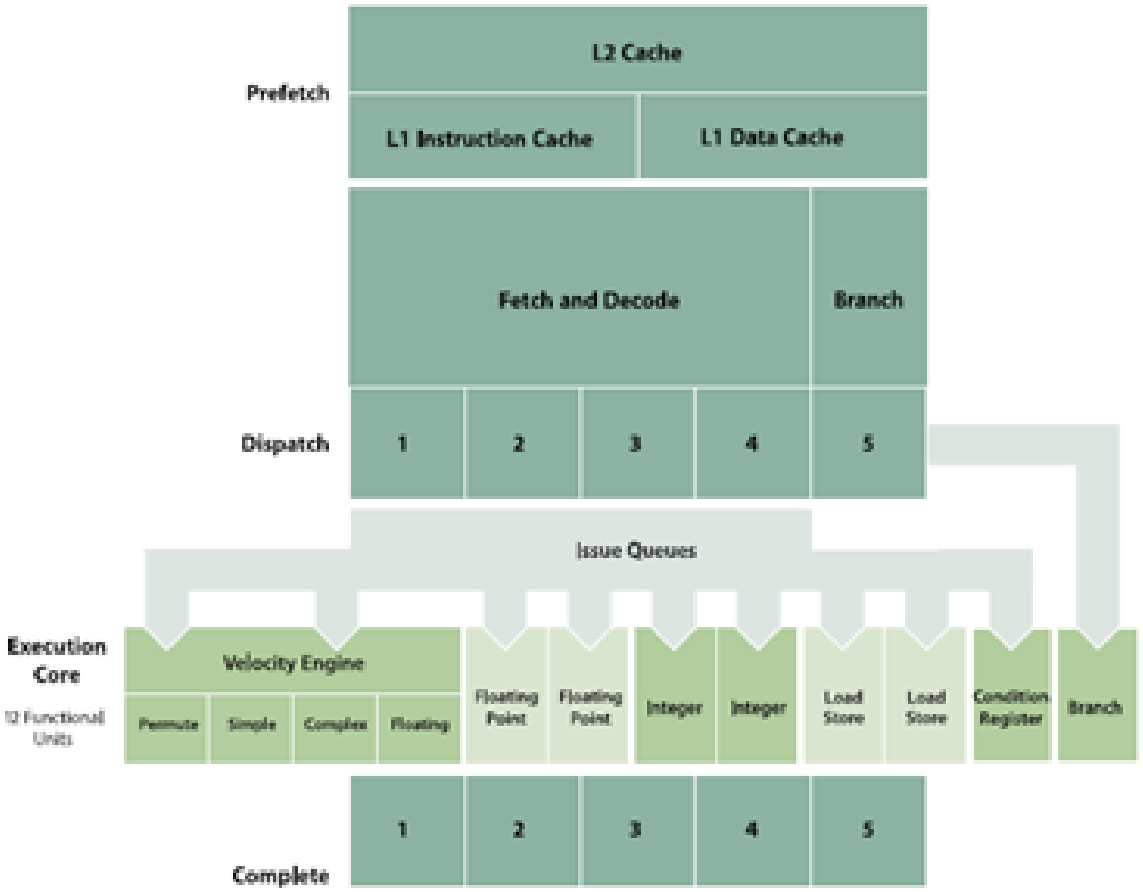
- A partir du cache N1, jusqu'à huit instructions par cycle d'horloge sont acheminées, décodées et partagées en opérations plus petites et plus simples à organiser. Cette préparation efficace optimise la vitesse de traitement alors que les instructions sont réparties dans le noyau d'exécution et que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

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.

Le PowerPC G5

Architecture PowerPC G5



Le noyau d'exécution contient 12 unités fonctionnelles distinctes :

- Le Velocity Engine utilise deux longues files d'attente et des registres 128 bits dédiés pour le traitement vectoriel.
- Deux unités en virgule flottante à double précision 64 bits fournissent la vitesse et la précision nécessaires pour les calculs très complexes.
- Deux unités en nombre entier 64 bits pour un vaste éventail de tâches de calcul.
- Deux unités de chargement/stockage gèrent les données lors de leur traitement, remplissant constamment les vastes registres du processeur pour des opérations plus rapides.
- Le registre des conditions stocke les résultats des prédictions de branchement pour améliorer la précision des prédictions futures.
- L'unité de prédiction de branchement utilise une logique à trois éléments innovante afin d'optimiser l'efficacité des processeurs.

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 ou info-stykades. On accède à euphor ou à stykades par une connexion sécurisée (ssh info-euphor ou info-stykades).

Etude d'exemples simples

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 3 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

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

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 4,4,3
        addi 3,4,42
        blr
```

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

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
_____ f oo :
        subf 4,4,3
        addi 3,4,42
        blr
```

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

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 0,4,3
3.      cmpw 1,3,4
4.      bgt- 1,L1
5.      subf 0,3,4
6.  L1:
7.      mr 3,0
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

- l'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 L1, 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 (L1).

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 L1 .

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

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 0,3
3           li 3,0
4           li 2,l
5           cmpw 7,2,0
6           bgtlr- 7
7 L6:
8           add 3,3,2
9           addi 2,2,l
10          cmpw 7,2,0
11          bgtlr- 7
12          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
```

Etude d'exemples simples

Analyse de 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
```

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

Tableaux

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

élément :
        slwi 4,4,2
        lwzx 3,4,3
        blr
```

Tableaux

```
/* somme des n premiers éléments du tableau t */

int somtab(int t[], int n)
{
        int k, s;
        s = 0;
        for (k=0; k<n; k++)
        s + = t [k] ;
        return s;
}
```

Tableaux

```
1 _somtab:
2         mr 9,3
3         l i  3,0
4         l i  2,0
5         cmpw 7,3,4
6         bgelr- 7
7 L6:
8         slwi 0,2,2
9         lwzx 0,9,0
10        add 3,3,0
11        addi  2,2,l
12        cmpw 7,2,4
13        bgelr- 7
14        b L6
```

Passage de paramètre par adresse

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


_incrementer:
lwz 2,0(3)
addi 2,2,l
stw 2,0(3)
blr
```

Passage de paramètre par adresse

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

Passage de paramètre par adresse : code généré

```
1 echanger :
2       lwz 2,0(3)
3       lwz 0,0(4)
4       stw 0,0(3)
5       stw 2,0(4)
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 ?

Utilisation de la pile

- Visualisation de l'appel à une fonction

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

Utilisation de la pile

```
_bar:
        mflr 0
        stw 0,8(1)
        stwu 1,-80(1)
        li 3,123
        li 4,456
        bl L_foo$stub
        lwz 0,88(1)
        addi 1,1,80
        mtlr 0
        blr
```

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

Utilisation de la pile

mflr 0

stw 0,8(1)

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 0,88(1)* puis *mtlr 0*.

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).

Troisième partie

Mécanismes de gestion des interruptions

Notion d’interruption

- En informatique, une **interruption** est un arrêt temporaire de l'exécution normale d'un programme par le microprocesseur afin d'exécuter un autre programme (appelé service d'interruption).
- La notion **d’interruption** se base sur un mécanisme par lequel certaines composantes physiques (horloge, E/S, mémoire, processeur) peuvent interrompre le traitement normal du processeur.

Exemples d’interruptions

- Les interruptions ont été définies pour faire face à certaines situations dans lesquelles un programme est en cours de traitement, et une situation nouvelle, extérieure au programme, nécessite un traitement urgent.
- Exemple : vous lisez un livre et le téléphone sonne. Si vous ne posez pas votre livre assez rapidement, le téléphone s'arrêtera de sonner. La bonne façon de faire consiste à poser le livre après y avoir mis un marque-page, à décrocher le téléphone, puis une fois la conversation terminée, poser le téléphone, reprendre le livre et continuer la lecture à l'endroit où vous aviez mis la marque.

http://www.dil.univ-mrs.fr/~gispert/enseignement/coursSysteme/Interruptions.php

Définitions

- Interruptions asynchrones pour désigner celles provoquées par un **événement externe** : avancement d'une horloge, signalisation de l'achèvement d'un transfert de données, etc.
- Interruptions synchrones pour **désigner les exceptions**, c'est-à-dire des arrêts provoqués par une condition exceptionnelle dans le programme (instruction erronée, accès à une zone mémoire inexistante, calcul arithmétique incorrect, appel volontaire au système d'exploitation...).

http://fr.wikipedia.org/wiki/Interruption_(informatique)

Fonctionnement

- Lors d'une interruption, le microprocesseur sauve tout ou une partie de son état interne, généralement dans la pile système, et exécute ensuite une routine d'interruption, généralement en suivant les directives d'une table indiquant pour chaque type d'interruption, le sous programme à exécuter.
- Une fois le traitement de l'interruption terminé, la routine se finit normalement par une instruction de retour d'interruption, qui restaure l'état sauvé et fait repartir le processeur de l'endroit où il avait été interrompu. Dans certains cas, la routine d'interruption modifie les adresses de retour, notamment pour effectuer des commutations de tâches.

http://fr.wikipedia.org/wiki/Interruption_(informatique)

Masque d'interruptions

- Lors du fonctionnement de certaines parties du système d'exploitation, il peut être nécessaire de ne pas permettre les interruptions, soit parce que celles-ci perturberaient un compte serré du temps, soit parce que des structures de données sont en cours de modification.

$\Rightarrow$ bloquer (**masquer**) les interruptions. Dans la plupart des systèmes, les interruptions bloquées sont accumulées, elles sont exécutées dès qu'elles sont démasquées.

- Sur certains systèmes, il existe des interruptions non masquables, généralement dédiée au signalement d'une erreur catastrophique pour le système

$\Rightarrow$les interruptions peuvent par ailleurs être hiérarchisées suivant des priorités. Une interruption de priorité supérieure est prise en compte lors du traitement d'une autre interruption, et une interruption de priorité inférieure est mise en attente.

http://fr.wikipedia.org/wiki/Interruption_(informatique)

Usages

- On utilise les interruptions principalement dans deux buts :
  - afin de permettre des communications non bloquantes avec des périphériques externes;
  - afin de commuter entre les tâches dans un ordonnanceur.
- Un autre usage, celui-là non prévu initialement, est l'introduction de malversations : lors de la restauration du contexte, si le contenu de la zone de sauvegarde a été altéré depuis l'appel (c'est le cas si l'interruption provoque une altération du contenu de la zone de sauvegarde ou de la pile), le contexte restauré sera totalement différent du contexte d'appel.

http://fr.wikipedia.org/wiki/Interruption_(informatique)