# ARCHITECTURE DES ORDINATEURS
# Examen Décembre 2009 (CORRIGÉ)
# 3H – Tous documents autorisés

## PROGRAMMATION ASSEMBLEUR (NIOS II)

**Q 1) Que fait la fonction F1 suivante?**

```
F1:    ADD R1, R0, R2
       BGE R2,R0, Suite
       SUB R1,R0,R2
Suite : JMP R31
```

*Calcule la valeur absolue du contenu du registre R2 et renvoie le résultat dans R1*

Soit la fonction F2 suivante

```
F2 :   SRAI R3,R2,31
       XOR R4,R3,R2
       SUB R1, R4, R3
       JMP R31
```

**Q 2) En supposant que R2 contienne la variable x**

1. Après exécution de l'instruction SRAI, que contient R3 lorsque x ≥ 0 ? Que contient R3 lorsque x <0 ?
2. Après exécution de l'instruction XOR, que contient R4 lorsque x ≥ 0 ? Que contient R4 lorsque x <0 ?
3. Après l'exécution de l'instruction SUB, que contient R1 lorsque x ≥ 0 ? Que contient R1 lorsque x <0 ?

| | x ≥ 0 | x <0 |
|---|---|---|
| SRAI | R3 = $00000000_H$ | R3 = $FFFFFFFF_H$ = -1 |
| XOR | R4 = x | R4 = $\overline{x}$ (complément de x) |
| SUB | R1 = x - 0 = x | R1 = $\overline{x}$ + 1 = -x |

**Q 3) Est-ce que les fonctions F1 et F2 renvoient le même résultat ? Si oui, quel peut être l'intérêt d'utiliser la fonction F2 au lieu de la fonction F1 ?**

Les deux fonctions F1 et F2 calculent la valeur absolue de R2 et renvoient le résultat dans R1. La fonction F2 n'utilise pas de branchement, donc n'a pas de pénalité de branchement mal prédit.

## PREDICTIONS DE BRANCHEMENT

Un programme a cinq branchements conditionnels notes B1 à B5. Le comportement des branchements est le suivant pour une exécution du programme (P pour branchement pris et N pour branchement non pris).

Branch 1: P-P-P-P-P
Branch 2: N-N-N
Branch 3: P-N-P-N-P-N-P-N
Branch 4: P-P-P-N-N-N
Branch 5: P-P-P-N-P-P-P-N-P
On suppose que le comportement de chaque branchement reste le même pour chaque exécution du programme. Les prédicteurs 1 bit et 2 bits sont toujours initialisés dans le même état avant chaque exécution

**Q 4) Donner le nombre de bonnes prédictions pour chaque branchement avec les prédicteurs suivants :**

**a)** Toujours pris
**b)** Toujours non pris
**c)** Prédicteur 1 bit, initialise à prédit pris.
**d)** Prédicteur 2 bits, initialisé à prédit faiblement pris.

| **Branch 1** | **P** | **P** | **P** | **P** | **P** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| TP | P | P | P | P | P | | | | | 5 |
| TNP | N | N | N | N | N | | | | | 0 |
| 1 bit | P | P | P | P | P | | | | | 5 |
| 2 bits | p | P | P | P | P | | | | | 5 |
| **Branch 2** | **N** | **N** | **N** | | | | | | | |
| TP | P | P | P | | | | | | | 0 |
| TNP | N | N | N | | | | | | | 3 |
| 1 bit | P | N | N | | | | | | | 2 |
| 2 bits | p | n | N | | | | | | | 2 |
| **Branch 3** | **P** | **N** | **P** | **N** | **P** | **N** | **P** | **N** | | |
| TP | P | P | P | P | P | P | P | P | | 4 |
| TNP | N | N | N | N | N | N | N | N | | 4 |
| 1 bit | P | P | N | P | N | P | N | P | | 1 |
| 2 bits | p | P | p | P | p | P | p | P | | 4 |
| **Branch 4** | **P** | **P** | **P** | **N** | **N** | **N** | | | | |
| TP | P | P | P | P | P | P | | | | 3 |
| TNP | N | N | N | N | N | N | | | | 3 |
| 1 bit | P | P | P | P | N | N | | | | 5 |
| 2 bits | p | P | P | P | p | n | | | | 4 |
| **Branch 5** | **P** | **P** | **P** | **N** | **P** | **P** | **P** | **N** | **P** | |
| TP | P | P | P | P | P | P | P | P | P | 7 |
| TNP | N | N | N | N | N | N | N | N | N | 2 |
| 1 bit | P | P | P | P | N | P | P | P | N | 5 |
| 2 bits | p | P | P | P | p | P | P | P | p | 7 |

## OPTIMISATION DE BOUCLES

On ajoute au jeu d'instructions NIOS II des instructions flottantes simple précision (32 bits) (Figure 2) et 32 registres flottants F0 à F31 (F0 est un registre normal).
Les additions, soustractions et multiplications flottantes sont pipelinées. Une nouvelle instruction peut démarrer à chaque cycle. Les latences sont de 2 cycles pour LF et de 4 cycles pour les instructions flottantes.
Les branchements ne sont pas retardés. On suppose qu'il n'y a pas de pénalité de branchement.

| | | | |
|---|---|---|---|
| LF | 2 | LF ft, déplac(rs) | ft ← MEM [rs +SIMM] |
| SF | 1 | SF ft, déplac.(rs) | ft → MEM [rs + SIMM] |
| FADD | 3 | FADD fd, fs,ft | fd ← fs + ft (addition flottante simple précision) |
| FMUL | 3 | FMUL fd, fs,ft | fd ← fs * ft (multiplication flottante simple précision) |
| FSUB | 3 | FSUB fd, fs,ft | fd ← fs - ft (soustraction flottante simple précision) |

**Figure 1 : Instructions flottantes ajoutées (Ce ne sont pas les instructions NIOS)**

Soit le programme assembleur P1, qui travaille sur des tableaux de flottants simple précision (float) X[N] et Y[N] rangés successivement en mémoire, avec N = 128. L'adresse de X[0] est initialement contenue dans le registre R1. L'adresse de S est 100H.

```
        FSUB F0,F0,F0
        ADDI R2, R0,200_10
Boucle :LF F1, 0(R1)
        LF F2, 512(R1)
        FADD F1, F1,F2
        FADD F0,F0,F1
        ADDI R1,R1,4
        BLT R1,R2, Boucle
        SF F0, 100_H(R0)
```

**Q 5) Donner le code C correspondant au programme P1**

```
#define N 128
float X[N], Y[N],S;
for (i=0 ; i<N ; i++)
      S+= X[i]+ Y[i]);
```

**Q 6) En montrant l'exécution cycle par cycle du programme assembleur P1,**

1. **donner le nombre de cycles par itération de la boucle du programme assembleur P1 (après réordonnancement éventuel des instructions)**
2. **donner le temps d'exécution total du programme. (On utilise une prédiction de branchement statique : les branchements avant sont prédits « non pris » et les branchements arrière sont prédits « pris ». Un branchement mal prédit a une pénalité de 3 cycles)**

| | |
|---|---|
| | FSUB F0,F0,F0 |
| | ADDI R2,R1,512 |
| 1 | Boucle : LF F1, 0(R1) |
| 2 | LF F2, 512(R1) |
| 3 | ADDI R1,R1,4 |
| 4 | .....FADD F1, F1,F2 |
| 5 | |
| 6 | |
| 7 | FADD F0,F0, F1 |
| 8 | BLT R1,R2, Boucle |
| +1 | Pénalité bcht |
| +2 | Pénalité bcht |
| +3 | Pénalité bcht |
| +4 | SF F0, $100_H$(R0) |

8 cycles par itération de la boucle

2 cycles avant la boucle
4 cycles après la boucle : 3 pénalité plus exécution de SF

**Temps d'exécution total : 128*8 + 6 = 1318 cycles.**

**Q 7) Refaire la question 8 avec un déroulage de boucle d'ordre 2 (2 itérations de la boucle initiale dans la boucle déroulée.**

1. **Donner le code assembleur de la boucle déroulée et son exécution cycle par cycle**
2. **Donner le nombre de cycles par itération de la boucle initiale**
3. **Donner le temps d'exécution total du programme**

| | |
|---|---|
| | FSUB F0,F0,F0 |
| | FSUB F5,F5,F5 |
| | ADDI R2,R1,400 |
| 1 | Boucle : LF F1, 0(R1) |
| 2 | LF F3, 4(R1) |
| 3 | LF F2, 512(R1) |
| 4 | LF F4, 516(R1) |
| 5 | .....FADD F1, F1,F2 |
| 6 | .....FADD F3, F3,F4 |
| 7 | ADDI R1,R1,8 |
| 8 | FADD F0,F0, F1 |
| 9 | FADD F5,F5, F3 |
| 10 | BLT R1,R2, Boucle |
| +1 | Pénalité bcht |
| +2 | Pénalité bcht |

| | |
|---|---|
| +3 | Pénalité bcht |
| +4 | FADD F0,F5, F0 |
| +5 | SF F0, $100_H$(R0) |

**5 cycles par itération de la boucleTemps d'exécution total : 128*5 + 3 + 5 = 648 cycles.**

## CACHES

Un processeur a un cache L1 de 16 Ko, avec des blocs (lignes) de cache de 64 octets. Il utilise l'écriture simultanée et l'écriture non allouée (pas de défauts de cache en écriture).
Il y a deux variantes :

- Correspondance directe
- Associativité 4 voies (ensembles de 4 blocs)

Soit un programme C qui utilise la fonction vsum()

```
#define X 2048
double A[X], B[X];

double vsum (double *v1, double *v2, int n)
{
double s=0.0;
int i ,
for (i=0; i<n ; i++)
s+= v1[i] + v2[i];
Return (s);
```

**Q 8) Donner le nombre de bits des champs :**

- **déplacement dans le bloc, numéro de bloc et étiquette en correspondance directe;**
- **déplacement dans le bloc et numéro d'index et étiquette en associativité 4 voies.**

Correspondance directe
6 bits de déplacement, 8 bits d'index et 18 bits d'étiquette
Associativité 4 voies
6 bits de déplacement, 6 bits d'index et 20 bits d'étiquette.

**Q 9) Les tableaux A et B sont alloués consécutivement en mémoire. A est placé à partir de l'adresse $00000040_H$. Dans quels blocs du cache vont A[0] et B[0] en correspondance directe ? Dans quels ensembles du cache vont A[0] et B[0] en associativité 4 voies ?**

A[0] est à l'adresse 00000040
B[0] est à l'adresse 00000040H + 2048 x $8_{10}$ = 00000040H+ 4000H = 0000 4040H.
En correspondance directe, A[0] va dans le bloc 1 et B[0] va dans le bloc 1.
En associativité 4 voies, A[0] va dans le l'ensemble 1 et B[0] va dans l'ensemble 1.

**Q 10) La variable s est en registre. Quel est le taux d'échecs (nombre défauts / nombre d'accès)**

1. **pour un appel vsum (A, B, N) si le cache utilise la correspondance directe ?**

Un bloc contient 8 doubles. A et B vont dans le même bloc. Il y a donc 1 défauts pour A et 1 défaut sur B à chaque itération, soit un taux d'échec de 100%.

2. **pour un appel vsum(A, A, N) avec la correspondance directe ?**

Il y a maintenant 1/8 défauts pour l'accès à pour le premier accès à A et 0/8 défauts pour le deuxième accès à A.. Taux d'échec = 1/16 = 6,25 %.

3. **pour un appel vsum (A, B, N) avec l'associativité 4 voies ?**

Avec l'associativité 4 voies, A et B vont dans des blocs différents de l'ensemble 1. Il y a 1/8 défauts pour l'accès à A et 1/8 défauts pour l'accès à B, soit un taux d'échec de 1/8 = 12,5%.

**Q 11) Que devient le taux d'échecs avec l'associativité 4 voies pour la fonction vmsum (A,A,N) suivante :**

```
double vmsum (double *v1, double *v2, int n){
double s=0.0;int i;
for (i=0; i<n/8 ; i++)
      s+= v1[i] + v2[i*8];
return (s);}
```

On charge 2048/8 = 256 doubles pour V1 et autant pour V2.
Un bloc de cache contient 8 doubles. Le cache contient 64 ensembles de 4 blocs.
Les 256 accès à V1 iront dans 256/8 = 32 ensembles successifs et les accès à V2 iront dans les 64 ensembles du cache à partir du premier ensemble utilisé où ils utiliseront 4 blocs par ensemble.
Tous les accès à V2 provoquent un défaut de cache soit 256 défauts. (pour i=0, en fait c'est le même défaut que pour l'accès à V1).
Les accès à V1 ne provoquent pas de défauts de cache (sauf pour i=0, qui est compté avec V2) car les blocs correspondant ont déjà été chargés et ne sont pas remplacés.
Il y a donc un total de 256 défauts pour 256 * 2 = 512 accès, soit un taux d'échec de 50%