TD 2

TD Caches

Les processeurs dédiés au traitement du signal (DSP, *Digital Signal Processor*) sont très largement utilisés dans le domaine des communications. Beaucoup de téléphones portables possèdent l'un de ces processeurs pour réaliser des traitements avancés sur des signaux analogiques digitalisés. Le processeur ADSP-2106x Sharc est un de ces processeurs, dont la particularité est de disposer d'une architecture Harvard modifiée. L'accès aux instructions et aux données stockées en mémoire peut se faire sur des bus séparés. Ce processeur est muni de deux blocs mémoire de SRAM de 2 Mbits chacun assurant des temps d'accès records à leur contenu.

2.1 Hiérarchie mémoire

L'architecture possède deux structures indépendantes permettant de réaliser des accès mémoire simultanés (bus, mémoire et registres d'échange). Concernant la hiérachie mémoire et ses bus, ce processeur possède une architecture Harvard interne liant le processeur à 2 paires de bus destinés à transférer les données et les instructions séparément. Chaque paire de bus est interfaçée à un composant mémoire différent : un bus pour identifier l'élément ciblé et un bus d'échange. L'un des composant mémoire est dit "data" et l'autre "program". On parlera par la suite des bus dm et pm en référence aux bus accédant aux mémoires data ou program. Ce processeur utilise des adresses de mots mémoire. Attention, l'adressage par mot mémoire signifie que chaque adresse correspond à un mot mémoire et non à un octet. Les instructions sont codées en taille fixe en mots de 48 bits. Les données en revanche sont adressées en mots de 32 bits. Cette situation est appelée "Normal word addressing". Les adresses d'instruction sont sur 24 bits, les adresses de données sur 32. En pratique, chaque mémoire (program ou data) a une capacité de stockage de 2 Mbits ($2^2 1$ bits).

Exercice [No. 4] (Adressages et mots)

1. En supposant une configuration de "normal word addressing", calculez la taille maximale en octets que l'ADSP saurait utiliser pour des mots de tailles 48 bits pour les instructions et 32 bits pour les données.

2. Inversement, pour une taille de 2 Mbits, calculez le nombre de mots mémoires effectivement adressables dans un composant mémoire pour des instructions ou des données en mode normal.

Remarque : ce genre d'informations permet de comprendre l'organisation de l'espace d'adressage en différentes "zones".

2.2 Jeu d'instruction

Ce processeur possède un jeu d'instruction dans lequel seul load et store peuvent faire des accès explicites à la mémoire. La représentation textuelle des instructions (langage assembleur) ne suit pas la notation MIPS (nom op1, op2, op3) que vous avez déjà manipulée[1]. Nous allons brièvement décrire le format des instructions.

[1] Le but est de vous faire découvrir d'autres syntaxes de langages assembleurs

Vous pouvez trouver le détail du jeu d'instruction page 461 du manuel de référence (ce n'est pa obligatoire, http://http://www.perso.telecom-paristech.fr/ trobert/se201/ADSPMan.pdf). Vous pouvez aussi y trouver le détail sur le calcul d'adresse.

Notations :

- instructions arithmétiques/logiques binaires (2 opérandes): Dest=operand1 op operande2. Dest, operand1 et operand2 sont des registres ou valeurs immédiates (sauf pour dest). Ces registres sont soit pour des entiers (prefixe r), soit pour des flottants (préfixe f).
- load / store double : il est possible de réaliser des accès doubles à la mémoire load/load, load /store, store/store... La syntaxe pour ces accès est «accès1, accès2;». Un accès load est décrit par `<reg>=<b>m(i<u>,m<v>)` où reg est un registre flottant, et u et v désignent des numéros de registres servant à construire les adresses mémoires. (il y a 16 registres i et m pour définir un adressage indexé). Cette technique permet de définir de manière très riche l'adressage. On supposera que les registres i0-15 et m0-15 ne changent pas au cours de l'exécution.

Du fait que la mémoire est de la SRAM, on supposera que les accès à la mémoire sont résolus en 1 cycle. Ainsi, ce processeur possède un pipeline à 3 étages, chaque étage s'exécutant en 1 cycle en l'absence de pénalité :

Fetch : l'instruction est lue dans le cache instruction ou en mémoire de programme (pm).

Decode **:** l'instruction est décodée

Execute **:** l'instruction est exécutée, et en cas de load/store les opérandes sont transférées vers ou depuis la mémoire.

Exercice [No. 5] (Accès double et pénalités) es bus dm et pm peuvent être accédés par les étages F et E du processeur. Lors d'un double accès mémoire E accède simultanément à pm et dm.

1. Serait il envisageable d'utiliser de l'adressage immédiat avec adresse complète pour un load simultané à pm et dm (il faut être capable de justifier).
2. Détaillez le conflit structurel qui se produit de lors de l'exécution de :

```
f2=pm(i8,m8), r2=dm(i2,m2);
r3=r3+7;
```

   On supposera que r3 n'a pas été modifié récemment lors de l'exécution de ce code.
3. Un cache d'instruction est utilisé pour éviter ce conflit structurel. Peut on savoir à l'exécution quelles instructions doivent être recherchées/placées dans le cache ?
4. Est ce implémentable (à quel cycle cette information est elle disponible/utilisée à votre avis – description RTL non fournie mais similaire à ce que vous avez vu dans le DLX).

Indication : lisez la suite du texte la réponse est presque fournie...

2.3 Fonctionnement du cache d'instruction

La mémoire cache du processeur Sharc est décrite dans la documentation de son fabricant *Analog Devices* de la façon suivante :

> The ADSP-2106x's on-chip instruction cache is a 2-way, set-associative cache with entries for 32 **instructions**. Operation of the cache is transparent to the programmer. The ADSP-2106x caches only instructions that conflict with program memory data accesses (over the PM Data Bus, with the address generated by DAG2 on the PM Address Bus). This feature makes the cache considerably more efficient than a cache that loads every instruction, since typically only a few instructions must access data from a block of program memory.

> Because of the three-stage instruction pipeline, if the instruction at address *n* requires a program memory data access, there is a conflict with the instruction fetch at address *n*+2, assuming sequential execution. It is this fetched instruction (*n*+2) that is stored in the instruction cache, not the instruction requiring the program memory data access.
>
> If the instruction needed is in the cache, a "cache hit" occurs – the cache provides the instruction while the program memory data access is performed. If the instruction needed is not in the cache, a "cache miss" occurs, and the instruction fetch (from memory) takes place in the cycle following the program memory data access, incurring one cycle of overhead. This instruction is loaded into the cache, if the cache is enabled and not frozen, so that it is available the next time the same instruction (requiring program memory data) is executed. In case no entry is available, the least recently used policy is applied.

Concernant les modes Frozen et Disbaled voici ce que dit la documentation :

> Freezing the cache prevents any changes to its contents?a cache miss will not result in a new instruction being stored in the cache.
>
> Disabling the cache stops its operation completely; all instruction fetches conflicting with program memory data accesses are delayed by the access. These functions are selected by the CADIS (cache enable/ disable) and CAFRZ (cache freeze) bits in the MODE2 register.

Exercice [No. 6] (Exécution d'une boucle)

1. Déterminez le mode de fonctionnement de ce cache parmis (associatif complet, par bloc ou placement direct)
2. Le schéma représentant les ressources de stockages associées au cache pour l'ADSP 2106x est donné dans la documentation par la figure 2.1.

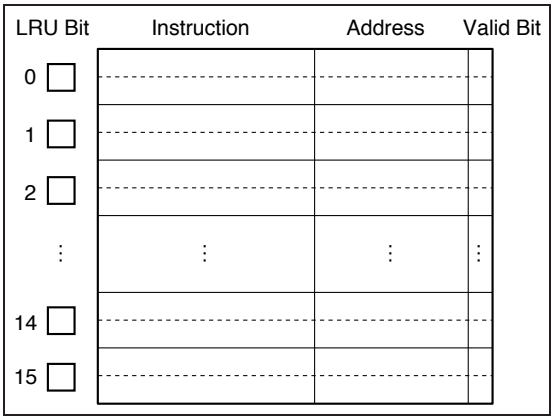

Figure 2.1: Architecture du cache de l'ADSP 2106x

   - Expliquez en quoi ce schéma permet d'identifier le degré d'associativité de ce cache (indication : où sont les ensembles associatifs de blocs, combien d'ensembles sont formés ?).
   - En vous rappelant de la définition de l'index, combien de bit d'index sont nécessaire pour ce cache.
   - La vérification du contenu repose sur la notion de tag ou étiquette. Quelle taille d'étiquette est nécessaire pour déterminer si un bloc donné contient la donnée souhaitée.
   - Quel est le rôle des bits *Valid Bit*? du bit LRU ?
3. Donnez un exemple, incluant un état du cache, et une séquence de 3 instructions permettant de distinguer le cas frozen du cas disbled en termes de performances.
4. On considère le programme consistant en une boucle dans laquelle on fait appel à un sous-programme `sousprg`. Rappelez vous que les opérandes des accès en mémoire sont identifiées par (`pm()` ou `dm()`) et des registres d'adressage (iX et mY). On considère le programme suivant :

```
0x0100          lcntr=1024, do boucle until lce ; RÃ©pÃ©ter 1024 fois le code
; jusqu'Ã  boucle (inclus)
0x0101          r0=dm(i0,m0), pm(i8,m8)=f3 ; mem. donnee --> r0
0x0101                                     ; et f3 --> mem.progr.
0x0102          r1=r0-r15;
0x0103          if eq call (sousprg)       ; appel du sous-prog.
0x0104          f2=float r1                ;
0x0105          f3=f2*f2                   ;
0x0106 boucle:  f3=f3+f4                   ; ==== fin boucle
0x0107          pm(i8,m8)=f3               ;
...
...
0x0200 sousprg: r1=r13;
0x0201          r14=pm(i9,m9)              ; mem. progr. --> r14
...
0x0211          pm(i9,m9)=r12              ; r12 --> mem.progr.
...
0x021F          rts
```

– Expliquer pourquoi ce programme est particulièrement mal adapté à cette structure de cache.

– Proposer une approche reposant sur le déplacement des instructions en mémoire permettant d'augmenter le nombre de cache hit.

==========