Architecture des ordinateurs

TP3 : processeur

C. Alias

Le but de ce TP est de comprendre comment assembler les circuits de base étudiés jusqu'à présent (ALU, banc de registres) pour construire un processeur simple. Il s'agit d'une version simplifiée du processeur MIPS décrit dans le livre de David Patterson et John Hennessy.

Exercice 0. *Jeu d'instructions*

Notre processeur peut adresser une **mémoire de données de 8 Ko** (adresses sur 13 bits), et comporte **8 registres 8 bits de r0 à r7** et les instructions suivantes:

- **add** $\mathbf{r_{dest}}, \mathbf{r_1}, \mathbf{r_2}$: additionne le contenu des registres r_1 et r_2, et place le résultat dans le registre r_{dest}.
- **sub** $\mathbf{r_{dest}}, \mathbf{r_1}, \mathbf{r_2}$: calcule $r_1 - r_2$ et place le résultat dans le registre r_{dest}.
- **ld** $\mathbf{r_{dest}}, [\mathbf{r_{base}} + \mathbf{imm7}]$: charge dans le registre r_{dest} la donnée en mémoire à l'adresse r_{base} + imm7, où imm7 est un entier 7 bits. On parle aussi de *valeur immédiate*, puisque l'entier est immédiatement disponible dans l'instruction.
- **st** $\mathbf{r_1}, [\mathbf{r_{base}} + \mathbf{imm7}]$: stocke la valeur du registre r_1 dans la mémoire à l'adresse r_{base} + imm7.
- **beq** $\mathbf{r_1}, \mathbf{r_2}, \mathbf{imm7}$: si $r_1 = r_2$, saute à l'instruction située à l'adresse imm7 + 1. (sinon, on passe à l'instruction suivante située à l'adresse courante plus 1). Cette instruction permet d'implémenter la boucle `for`, la boucle `while` et le if.
- **ldi** $\mathbf{r_{dest}}, \mathbf{imm8}$: écrit l'entier 8 bits imm8 dans le registre r_{dest}.
- **ja** $\mathbf{r_1}, \mathbf{r_2}$: saute à l'adresse 13 bits définie par r_1 pour les 8 bits de poids faible et par r_1 pour les 5 bits restants (de poids fort).
- **j imm13**: saute à l'adresse 13 bits imm13, où imm13 est un entier 13 bits. Cette instruction, avec `ja`, permet d'implémenter les appels de fonctions.

Chaque instruction est codée sur 16 bits comme précisé en annexe.

Questions.

- Ecrire un programme qui initialise le registre r0 avec 1 et qui l'incrémente jusqu'à ce qu'il soit égal à 10.
- En vous aidant de l'annexe, donnez le codage de votre programme sous forme binaire, puis sous forme hexadécimale.
- Complétez le tableau suivant avec le codage binaire, puis hexadécimal du programme donné dans la colonne de gauche.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ld r0,[r1+1]	0	1	0													
st r0,[r1+1]	0	1	1													

Exercice 1. *ALU, banc de registres et RAM*

Copiez le fichier `/home/tpetu/INF2003L/diglog/tp/digmips_mask.tgz` sur votre compte.
Décompactez le avec `tar xvfz digmips_mask.tgz`, allez dans le répertoire `digmips_mask`, puis chargez le processeur avec `diglog *.lgf`.

Le processeur comporte plusieurs fichiers:

- **base.lgf** (onglet 2) contient tous les composants de base (multiplexeurs, décodeurs, etc)
- **alu.lgf** (onglet 1) contient l'unité arithmétique et logique 8 bits.
- **register.lgf** (onglet 5) contient le banc de 8 registres 8 bits.
- **datapath.lgf** (onglet 4) contient le *chemin de données* du processeur.
- **control.lgf** (onglet 3) contient le *circuit de contrôle* du processeur.

Le but de cet exercice est de vous familiariser avec les composants des trois premiers fichiers. Le chemin de données et le circuit de contrôle seront étudiés dans les exercices qui suivent.

A) ALU

Les entiers 8 bits à additionner, soustraire ou comparer entrent par les fils de gauche, le bit de poids faible est toujours en haut. Le résultat est disponible sur les 8 premiers fils de droite en partant du haut. Les fils suivants indiquent successivement un dépassement (overflow), le résultat d'un test d'égalité (1 si égal, 0 sinon), le résultat d'un test de supériorité (1 si l'entier du haut est plus grand que l'entier du bas). Cette dernière fonction n'est pas utilisée.

- Dans un onglet libre (6 par exemple), **instanciez et testez l'ALU**. On testera l'addition, la soustraction et le test d'égalité.

B) Banc de registres

Il s'agit d'un banc de 8 registres 8 bits, dont la spécification est analogue à celle du banc de 4 registres 4 bits étudié dans le TP 2.

- **Instanciez et testez le banc de registres**. On placera la valeur 1 dans le registre 2, puis on lira les registres 1 et 2.

C) RAM

Le programme et les données doivent être stockés dans une mémoire. On utilise la mémoire SRAM8K disponible dans la bibliothèque de composants de DIGLOG.

- Allez dans la bibliothèque (cliquez sur CAT, puis sur LIBR), trouvez là, et instanciez là.

Comme son nom l'indique, c'est une mémoire RAM de 8 Ko (8×2^{10} octets), les adresses sont donc sur 13 bits. Le **port d'adresse** est sur la gauche, le poids faible en haut. Sur le haut, il y a trois bits de contrôle: horloge, R et OE. **Les bits de contrôle Horloge et OE doivent toujours être à 0**.

- **Lecture.** Lorsque le bit R vaut 1, la mémoire est en mode lecture. Le port de lecture (en bas) contient la valeur de l'octet à l'adresse spécifiée sur le port d'adresse. Le bit poids faible est à gauche. **Expérimentez.**
- **Ecriture.** Lorsque le bit R vaut 0, la mémoire est en mode écriture. La valeur disponible sur le port de données est écrite à l'adresse spécifiée sur le port d'adresse, jusqu'à ce que le bit R revienne à 0. **Expérimentez.**
- **Initialisation.** On peut préremplir la mémoire avec des données placées dans un fichier, ce qui est bien pratique pour charger des programmes. Le fichier contient les données à charger en RAM sous forme hexadécimale, c'est un simple fichier texte. **Ouvrez le fichier simple_lo.ram** avec un éditeur de texte.

Pour charger un tel fichier en RAM, il suffit de se placer en mode **CNFG** (en bas à droite), puis de **cliquer** sur la RAM. Dans la fenêtre *newcrt*, **descendre** le curseur à *file name to load* en appuyant sur la flèche du bas, puis **entrer** le nom du fichier. Enfin, **cliquer** sur la fenêtre *newcrt* pour revenir dans la fenêtre *mylib*.

Exercice 2. *Chemin de données*

Le chemin de données (datapath.lgf, onglet 4) est le circuit qui permet au processeur de réaliser son calcul. Le chemin de données comporte :

- Une **mémoire d'instructions (16 bits)** dans deux SRAM 8 bits. Pour chaque instruction, la *SRAM de droite* fournit les bits de 0 à 7 (poids faible) et la *SRAM de gauche* fournit les bits de 8 à 15 (poids fort).
- Une **mémoire de données (8 bits)**, dans une SRAM 8 bits.
- Un **registre PC (program counter)**, qui contient l'adresse de l'instruction courante sur 13 bits. En réalité, PC est reparti sur 2 registres 8 bits: PC_lo pour les bits de 0 à 7 et PC_hi pour les bits de 8 à 12. Les bits 13, 14 et 15 sont inutilisés.
- Une partie dédiée au **calcul de l'adresse de l'instruction suivante**. La plupart du temps, on incrémente PC.
- Un **banc de 8 registres 8 bits**.
- Une **ALU 8 bits**.
- Des **signaux de contrôle** (ecrire_reg, reg_dst, etc) qui permettent d'exécuter l'instruction courante en spécifiant quel registre lire/écrire, quelle opération arithmétique effectuer, quelle donnée lire/écrire en mémoire de données et plus généralement comment aiguiller les données. Par exemple, les signaux de contrôle pour les instructions ld et st sont donnés dans le tableau de l'exercice 3.

A) Chargement du programme

Comme la mémoire d'instructions est répartie sur deux SRAM 8 bits, il faut construire deux fichiers pour chaque programme:

- **Un fichier avec les 8 bits de poids fort** de chaque instruction à charger dans la RAM de gauche.
- **Un fichier avec les 8 bits de poids faible** de chaque instruction à charger dans la RAM de droite.

Les fichiers **simple_hi.ram** et **simple_lo.ram** contiennent l'encodage du programme étudié dans l'exercice 0:

```
ld r0,[r1+1]
st r0,[r1+1]
```

- Quel fichier contient les 8 bits de *poids fort* de chaque instruction ?
- Quel fichier contient les 8 bits de *poids faible* de chaque instruction ?
- Chargez les en RAM.
- Ajoutez des composants 7SEG pour afficher la valeur courante de PC. Donnez des quelques tops d'horloge.
- Pour remettre PC à 0, mettez `reset` à 1, et faites passer l'horloge à 1, puis à 0. Enfin, remettez `reset` à 0.

B) Exécution de ld r0,[r1+1]

On suppose désormais que PC=0, l'instruction courante est donc ld r0,[r1+1]. Ne touchez plus à l'horloge.

- Repérez les différentes parties de l'instruction courante à la sortie de la mémoire d'instructions (opcode, opérandes r0, r1 et 1).
- Pour exécuter ld r0,[r1+1], il faut calculer r1+1, lire la case r1+1 dans la mémoire de données, l'écrire dans r0. Nous allons détailler ces trois étapes.
  - **Calcul de l'adresse r1+1.**
    - Comment le contenu de r1 est recupéré et placé à l'entrée de l'ALU ?
    - Comment 1 est recupéré et placé à l'entrée de l'ALU ?
    - Quel est le rôle du signal de contrôle alu_src ?
    - Quel est le rôle du signal de contrôle alu_op ?
    - Où se trouve la valeur de r1+1 ?
  - **Lecture dans la mémoire de données.**
    - Repérer comment la valeur de r1+1 est amenée sur le port d'adresse de la mémoire de données. La donnée est maintenant disponible sur le port de données.
    - Quel est le rôle du signal mem_vers_reg ?
    - Que se passe-t-il lorsque mem_vers_reg=imm_vers_reg=0 ? A quelles instructions cela peut-il servir ?
    - Que se passe-t-il lorsque imm_vers_reg vaut 1 ? A quelle instruction cela peut-il servir ?
  - **Ecriture dans r0.**

    Après un temps de stabilisation, la donnée à écrire est disponible sur les bits w0 ... w7 du banc de registres. Le numéro du registre à écrire (r0) est également disponible.
    - Réperez les.
    - Quel signal manque-t-il pour écrire la donnée dans r0?
    - Faites passer l'horloge à 1. Ne touchez plus à l'horloge. Que constatez vous ? Pourquoi attendre que l'horloge soit à 1 pour envoyer le signal d'écriture ?

Le tableau de l'exercice 3 récapitule les signaux de contrôle nécessaires à l'exécution de l'instruction ld.

C) Exécution de st r0,[r1+1]

Faites passer l'horloge à 0 avec un clic. On exécute maintenant l'instruction st r0,[r1+1]. Ne touchez plus à l'horloge.

- Repérez l'opcode de l'instruction st et les opérandes r0, r1 et 1.
- Pour exécuter st r0,[r1+1], il faut calculer r1+1, récupérer la valeur de r0, l'écrire dans la mémoire de données à l'adresse r1+1. Nous allons détailler ces trois étapes.
  - **Calcul de r1+1.** Vérifiez que tout ce passe comme pour l'instruction précédente.
  - **Récupérer la valeur de r0.**
    - Quel est le rôle du signal reg_dst ?
    - Réperez comment la valeur de r0 est amenée à l'entrée de la mémoire.
  - **L'écrire dans la mémoire de données.**
    - Quels signaux manquent pour écrire r0 à l'adresse r1+1 ?
    - Faites passer l'horloge à 1. Commentez.

Le tableau de l'exercice 3 récapitule les signaux de contrôle nécessaires à l'exécution de l'instruction st.

Exercice 3. *Contrôle*

Pour chaque instruction, il faut spécifier quel registre lire/écrire, quelle opération arithmétique effectuer, quelle donnée lire/écrire en mémoire de données, comment aiguiller les données. Pour cela, on utilise des **signaux de contrôle**, comme ecrire_reg qui décide l'écriture dans le banc de registres, ou alu_op au dessus de l'ALU, qui vaut 0 pour une addition et 1 pour une soustraction. Les signaux de contrôle d'une instruction ne dépendent que de son *opcode*, et peuvent être générés avec un circuit combinatoire (voir control.lgf, onglet 3).

Le tableau suivant récapitule les signaux de contrôle nécessaires à l'exécution des instructions ld et st vues dans l'exercice 2.

	ecrire_reg	reg_dst	alu_src	alu_op	ecrire_mem	beq	mem_vers_reg	imm_vers_reg	ja
`add 000`									
`sub 001`									
`ld 010`	1	0	1	0	0	0	1	0	0
`st 011`	0	1	1	0	1	0	0	0	0
`beq 100`									
`ldi 101`									
`ja 110`									

Placez vous dans control.lgf (onglet 3). *opcode0*, *opcode1* et *opcode2* sont les bits 13, 14 et 15 de l'instruction courante (l'opcode).

- **Vérifiez** que les signaux de contrôle émis pour ld et st sont bien conformes à ce qui est spécifié dans le tableau. Il faut maintenant compléter control.lgf pour gérer les autres instructions.
- **Complétez le tableau ci-dessus** pour chacune des autres instructions.
- **Complétez control.lgf** (onglet 3) pour les prendre en charge.
- **Testez sur le programme test.asm** (test_lo.ram, test_hi.ram) dans le répertoire courant.

Exercice 4. *Le saut immédiat J*

Il manque à notre processeur une implémentation du saut immédiat j.

- **Complétez le chemin de données** pour gérer le saut immédiat. *Indication: Il suffit d'ajouter un multiplexeur à la partie 'calcul de l'instruction suivante'.*
- **Complétez le contrôle** pour actionner le signal de contrôle du multiplexeur lorsque l'instruction est j.
- **Testez sur le programme inc.asm** (inc_lo.ram, inc_hi.ram) dans le répertoire courant.

Format des instructions

Les instructions se codent sur 16 bits de la façon suivante. Le caractère '-' indique un bit inutilisé.

`add` r_{dest}, r_1, r_2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0										-	-	-	-
opcode			r_{dest}			r_1			r_2						

`sub` r_{dest}, r_1, r_2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1										-	-	-	-
opcode			r_{dest}			r_1			r_2						

`ld` $r_{dest}, [r_{base} + imm7]$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0													
opcode			r_{dest}			r_{base}			imm7						

`st` $r_1, [r_{base} + imm7]$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1													
opcode			r_{dest}			r_{base}			imm7						

`beq` $r_1, r_2, imm7$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0													
opcode			r_1			r_2			imm7						

`ldi` $r_{dest}, imm8$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1				-	-								
opcode			r_{dest}					imm8							

`ja` r_1, r_2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0							-	-	-	-	-	-	-
opcode			r_1			r_2									

`j` imm13

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1													
opcode			imm13												