Compilation

TP 0.0 : L'architecture cible: Digmips

C. Alias & G. Iooss

Le but de ces TPs est de construire un compilateur C pour Digmips, un petit processeur MIPS implémenté dans le logiciel de simulation logique Diglog. Le but de ce TP est de prendre en main Diglog et Digmips, puis de construire quelques programmes en assembleur.

Exercice 1. *Prise en main de* Diglog

Ajouter le répertoire `/home/calias/M1-Compilation/diglog/bin` à votre variable d'environnement `PATH`. Ouvrir un terminal et entrer `diglog`.

- Diglog est composé de deux fenêtres appelées *mylib* et *mycrt*. *mylib* permet de construire le circuit et *mycrt* est une fenêtre annexe utilisée pour le dialogue (chargement/sauvegarde, etc). Diglog vous permet de dessiner des circuits de manière intuitive à l'aide d'une interface graphique.

- Au bas de la fenêtre *mylib*, il y a une zone contenant quelques portes logiques et des points. Cette barre vous permet de stocker les portes logiques que vous utilisez le plus souvent. Pour placer une porte logique dans cette barre, cliquez simplement sur une porte et amenez-la à l'endroit desiré. Si vous placez une porte sur une porte déjà existante, elle la remplace.

- A gauche et à droite se trouvent des menus, mais la plupart des commandes peuvent être lancées à partir du clavier, ce qui est souvent plus rapide.

- Pour **placer une porte logique**, cliquez sur la porte dans la barre du bas, et faites glisser à l'endroit désiré.

- Pour **tracer un fil**, utilisez la souris comme dans un logiciel de dessin. En cliquant sur le bouton gauche vous commencez un fil, vous marquez un angle ou la fin du fil en cliquant à nouveau et quand vous avez terminé, vous cliquez sur le bouton droit.

- Pour **imposer une valeur sur un fil**, on peut utiliser un *générateur*. C'est le composant en bas à gauche de la barre. En cliquant au centre du générateur, on fait passer sa valeur de 0 à 1.

- Pour **visualiser la valeur sur un fil**, on peut utiliser une *sonde*. C'est le composant carré à droite du générateur. On peut également passer en *glow mode* en appuyant sur la touche g. On peut sortir le glow-mode en appuyant à nouveau sur la touche g.

- Pour **utiliser d'autres portes logiques**, on peut utiliser le *catalogue*. Pour çà, cliquer sur CAT en bas à gauche et cliquer sur le composant voulu. Si çà ne suffit pas, on peut chercher dans la bibliothèque de composants en cliquant sur LIBR. Une liste de composants s'affiche alors dans la fenêtre *newcrt*. On peut passer à la catégorie suivant en appuyant sur la touche + et revenir à la catégorie précédente en appuyant sur la touche -. Pour choisir un composant, il suffit de cliquer dessus. Pour quitter, appuyer sur la touche espace.

- Diglog se compose d'**onglets**, accessibles en appuyant sur les touches 1, 2, 3, etc.

- Pour **sauvegarder l'onglet courant dans un fichier**, faites Maj+S. Entrez le nom du fichier dans la fenêtre *newcrt*.

- Pour **charger un fichier dans l'onglet courant**, faites Maj+L. Entrez le nom du fichier dans la fenêtre *newcrt*.

- Pour **quitter** Diglog, faites Maj+Q. Confirmez ensuite dans la fenêtre *newcrt*.

Manip.

- **Construire un ET 4 bits**. Ajouter des émetteurs. Passer en *glow-mode* et expérimenter.
- **Insérer une horloge**. Régler la fréquence à 1 top par seconde. Pour çà, **passer en mode CNFG** (en bas à droite) et cliquer sur l'horloge. Les paramètres de l'horloge s'affichent dans la fenêtre *newcrt*. Régler en utilisant les flèches. Appuyer sur espace pour valider. Retourner dans la fenêtre principale.
- **Sauvegarder** dans le fichier `tp1.lgf`.

Exercice 2. *Prise en main de* DIGMIPS

Copiez le contenu du répertoire `/home/calias/M1-Compilation/digmips/` sur votre compte. Placez vous dans votre répertoire `digmips/`, 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 6) 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.
- **io.lgf** (onglet 5) contient un écran et un petit clavier pour les entrées/sorties.

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**. Les instructions sont codées sur 16 bits (voir annexe) et seront détaillées plus tard. Le programme est stocké dans la mémoire d'instructions, qui est répartie sur deux SRAM 8 bits:

- La mémoire de droite contient les **8 bits de poids faible** de chaque instruction
- La mémoire de gauche contient les **8 bits de poids fort** de chaque instruction.

DIGLOG permet de **remplir une mémoire** avec des données placées dans **un fichier**. Ce fichier contient les données à charger en RAM sous forme hexadécimale. Il faut donc construire **deux fichiers**: un fichier pour la RAM de gauche avec les 8 bits de poids fort de chaque instruction, et un fichier pour la RAM de droite avec les 8 bits de poids faible de chaque instruction. Les fichiers **hello_hi.ram** et **hello_lo.ram** contiennent l'encodage d'un programme qui affiche en boucle la chaine "Hello" à l'écran.

Pour charger un fichier dans une SRAM, 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, **appuyer sur espace** pour valider et revenir dans la fenêtre *mylib*.

Manip.

- **Charger le programme "Hello"**. **Mettre l'émetteur reset à 1** (en dessous de l'horloge) pour remettre le compteur d'instruction à zéro. **Remettre reset à 0** pour lancer l'exécution du programme. **Observer le résultat** sur l'onglet 5 (entrées/sorties).
- **Mettre reset à 1**. Placer un émetteur à la place de l'horloge. **Remettre reset à 0**, et cliquer sur l'émetteur pour exécuter le programme **pas-à-pas**.

Exercice 2. *Programmer avec des 1 et des 0*

Notre processeur comporte 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.
- **ble $\mathbf{r_1}, \mathbf{r_2}, \mathbf{imm7}$**: si $r_1 \leq r_2$, 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 $\mathbf{r_{dest}}$, 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.

Manip.

- **Sur papier, écrire 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.
- **Construisez les fichiers .ram** correspondants, et **chargez** votre programme dans DIGMIPS. **Exécutez pas à pas**.

Exercice 3. *Programmer en assembleur*

Il existe un outil qui génére automatique les fichiers .ram à partir d'un programme écrit en langage machine. Cet outil est appelé *assembleur*. Voilà un exemple de programme accepté par notre assembleur:

```
      ldi r1,1
      ldi r2,1            // compteur = 1
      ldi r3,10           // max = 10

loop:
      ble r2,r3,loop_stmt
      j end_loop          // compteur > 10? => stop
loop_stmt:

      ldi r4,'*'
      st r4,[r0+255]      // afficher une étoile

      add r2,r2,r1        // compteur = compteur + 1
      j loop              // itération suivante

end_loop:
      j end_loop
```

`loop` et `end_loop` sont des *labels*. Ils nomment les points d'exécution du programme sur lesquels effectuer un saut. L'assembleur se charge de calculer le décalage pour beq et l'adresse absolue pour j.

Manip.

- Ajouter le répertoire `/home/calias/M1-Compilation/asm/bin/` à votre variable `$PATH`.
- Entrez le programme ci-dessus dans le fichier `affiche10.asm`.
- Assemblez-le avec `asm affiche10.asm`. L'assembleur produit:
  - Le code hexadecimal correspondant sur la sortie standard.
  - Les deux fichiers **affiche10_hi.ram** et **affiche10_lo.ram** nécessaires au chargement dans DIGMIPS.
- Testez dans DIGMIPS les fichiers produits par l'assembleur.

Exercice 4. *Entrées/Sorties*

DIGMIPS possède un fichier supplémentaire, (io.lgf, onglet 5) qui permet de réaliser des entrées sur le clavier et des sorties sur l'écran. En principe, il faudrait ajouter deux instructions dédiées aux entrées/sorties. Comme on ne peut disposer que de 8 instructions, on choisi de recycler les instructions ld et st de la façon suivante:

- **Pour afficher un caractère sur l'écran.** Placer le code ASCII le caractère dans un registre r, puis exécuter st r,[n_importe_quel_registre + 255]. Le registre de base n'a pas d'importance, il faut juste que la valeur immédiate soit 255. Par exemple:

  ```
  ldi r0,'a'     // place le code ascii de 'a' dans r0
  st r0,[r7+255] // affiche r0.
  ```

- **Pour lire un caractère au clavier.** De manière analogue, il suffit d'exécuter: ld r,[n_importe_quel_registre + 255]. Par exemple:

  ```
  ld r,[r7+255] // Lit l'état du clavier.
  ```

  Les caractères du clavier sont stockés dans une file de taille 4. Il est possible qu'aucun caractère ne soit disponible. Dans ce cas, on lit la valeur 0. Pour saisir un caractère au clavier, il faudra donc boucler jusqu'à ce qu'un caractère soit disponible.

Manip.

- Ecrire un programme `saisie.asm` qui affiche `nb?`, passe à la ligne, saisit un entier entre 0 et 9, passe à la ligne, et l'affiche. On passe à la ligne en affichant le caractère 13.

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						

`ble` $r_1, r_2, imm7$

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												