Architecture des ordinateurs
Corrigé du TP 2 : DLXview

Arnaud Giersch, Benoît Meister et Frédéric Vivien

Ce document comprend 4 parties : 2 parties présentent l'architecture dlx et le simulateur dlxview [1] , les 2 dernières parties correspondent au travail demandé. Tous les fichiers cités dans ce documents sont téléchargeables dans le répertoire `http://icps.u-strasbg.fr/~vivien/Enseignement/Archi-2001-2002/`

Architecture DLX

- 32 registres entiers : $r0, r1, ..., r31$;
- 32 registres flottants (utilisation en simple ou double précision) : $f0, f1, ..., f31$;
- les instructions les plus courantes.

Pour l'utilisation, voir l'exemple `example.s` disponible à l'url : `http://icps.u-strasbg.fr/~vivien/Enseignement/Archi-2001-2002/example.s`.

On appelle *valeur immédiate*, ou en abrégé *immédiat*, une valeur (constante) du programme. Lorsque le nom d'une instruction se termine par « i » l'un de ses opérandes est un *immédiat*.

Les différentes instructions de l'architecture DLX sont présentées figure 1.

Dlxview

Dlxview est un logiciel permettant de visualiser dans un pipeline le cheminement d'instructions dlx. Utilisation :

- Lancement de dlxview : taper la commande dlxview
- Visualisation du pipeline : la séquence suivante permet de configurer le pipeline, elle aboutit à l'ouverture d'un fenêtre contenant un schéma du pipeline : *configure* → *Basic Pipeline* → *Ok* → *Ok*
- Type de pipeline : la fenêtre contenant le pipeline permet l'affichage d'un pipeline détaillé ou non détaillé. Le passage d'un mode à l'autre se fait par pression du boutton *view integer datapath*.
- Chargement d'un programme : dans la fenêtre DLX Visual Simulator : *load* → *sélection des fichiers* → *load* → *done*
- Exécution du programme pas à pas : *next cycle* → *main* (si une étiquette `_main` est présente dans le programme). *next cycle* permet ensuite de passer au cycle suivant alors que *step forward* charge directement la prochaine instruction
- Exécution sans mode pas à pas : dans la fenêtre DLX Visual Simulator : *go*
- Relancer une exécution : dans la fenêtre DLX Visual Simulator : *reset* → *Same Config & [New/Same] Program*
- Accès au contenu des registres : dans la fenêtre de contrôle (xterm) commandes :
  - **get** ri d : pour afficher le contenu du registre entier ri en décimal
  - **fget** fi : pour afficher le contenu du registre flottant fi
  - **put** ri val : pour affécter val au registre entier ri
  - **fput** fi val : pour affécter val au registre flottant fi
- Accès au contenu de la mémoire : commandes **get** et **put**, les instructions d'un programme chargé sont placées à l'adresse 256,
  - **get** 256 10i : affiche 10 instructions à partir de l'adresse 256
  - **get** 0 20 : affiche 20 mots mémoires au format hexadécimal à partir de l'adresse 0

[1] Si vous voulez télécharger dlxview, en voici l'URL : `http://yara.ecn.purdue.edu/~teamaaa/dlxview/`.

Type d'instructions	Signification
Transfert de données registres/mémoire	
lw, sw	chargement/rangement mot vers/depuis des registres entiers
lf, ld, sf, sd	chargement/rangement flottant simple précision (f) ou double précision (d) vers/depuis des registres flottants
Arithmétique et logique	
add, addi, addu, addui	addition, addition avec immédiat (tous les immédiats ont 16 bits) ; signée et non signée
sub, subi, subu, subui	soustraction, soustraction avec immédiat ; signée, non signée
mult, multu, div, divu	multiplication et division, signée et non signée ; les opérandes doivent être des registres flottants ; opérations sur 32 bits
and, andi	et, et avec immédiat
or, ori, xor, xori	ou et xor avec ou sans immédiat
s_, s_i	positionner la condition : "_" peut être **eq** (equal), **ne** (no equal), **lt** ($<$), **gt** ($>$), **le** ($\leq$), **ge** ($\geq$)
Contrôle	
beqz, bnez	branchement si registre égal/non égal à zéro
nop	instruction vide
trap	appel au système d'exploitation à une adresse vectorisée
Opérations flottantes simple précision (SF) et double précision (DF), les opérandes doivent être des registres flottants ; opérations sur 32 bits	
addf, addd, subf, subd	addition, soustraction SF et DF
multf, multd, divf, divd	multiplication et division SF etDF
cvt2f, cvtf2i, cvtd2f, cvtd2i, cvti2f, cvti2d	conversion : cvt$x$2y convertit du type x au type y, **i** correspond à entier, **d** à flottant double et **f** à flottant simple ; les opérandes sont dans des registres flottants

FIG. 1 – Instructions dans l'architecture DLX.

– **get** 100 5d : affiche 5 mots mémoires au format décimal à partir de l'adresse 100

Analyse d'un programme dlx

Récupérer le programme *example.s*. Celui-ci doit être lancé après avoir initialisé le registre r1 à l'adresse 0 : soit en tapant préalablement dans le terminal de contrôle **put r1 0** soit en chargeant un fichier d'initialisation *example.i*. Il faut, de même, initialiser r4 à zéro.

Url du programme : `http://icps.u-strasbg.fr/~vivien/Enseignement/Archi-2001-2002/example.i`

Questions :

– Sans l'exécuter, dites ce que fait ce programme.

– Chargez ensuite le programme dans dlxview. Dans le terminal de contrôle, affichez le contenu de la mémoire pour vérifier que celle-ci contient bien les données ainsi que les instructions.

– Exécutez ce programme avec dlxview. On s'aperçoit qu'il ne produit pas le résultat escompté. Cherchez la cause de cet échec en réalisant une exécution pas à pas et en consultant régulièrement le contenu des registres. Que signifient les « `stall` » qui apparaissent dans le pipeline ? Modifiez ensuite le programme pour rendre son exécution correcte.

Exercice de programmation

L'objectif est de réaliser un programme qui trouve le plus petit et le plus grand entier d'un tableau. Dans le programme, ce tableau sera donné dans la zone `data` qui contiendra le nombre d'élément du tableau puis les éléments.

Les deux résultats doivent être inscrits dans la zone mémoire à la suite des éléments du tableau.

Dans un deuxième temps, vous ordonnerez au mieux vos instructions pour minimiser le nombre de `stall`.