Architecture des microprocesseurs (GIF-3000)
Département de génie électrique et de génie informatique
Automne 2025

UNIVERSITÉ LAVAL

Projet 1

Instructions : – Ce TP est à réaliser individuellement.
– Déposez sur l'ENA une archive **zip** de votre projet.
– Date limite : le lundi **3 novembre**, à 23h59.

Pondération : Ce projet compte pour 10% de la note finale.

1 Objectifs

Ce projet comporte les objectifs suivants :

1. Comprendre le fonctionnement d'un microprocesseur implanté en pipeline ;
2. Réaliser les défis présents lors de la conception d'un microprocesseur ;
3. Approfondir les connaissances de circuits numériques avec microprocesseur.

2 Énoncé

Pour ce premier TP, vous devez concevoir un microprocesseur **pipeliné** de 12 bits en utilisant le logiciel Logisim-Evolution [1].

Votre processeur doit implanter le jeu d'instruction R12 décrit à la figure 1 et posséder quatre (4) registres de 12 bits chacun. Nous nommerons ces registres R0, R1, R2 et R3. Dans votre processeur, la mémoire des instructions sera séparée de la mémoire des données (architecture Harvard). Chacune des deux mémoires possédera une capacité de 4096 mots de 12 bits. Nous allons supposer pour ce premier TP que la mémoire des instructions répond en moins d'un cycle d'horloge, comme une cache L1, et que la mémoire des données répondra aussi en un (1) seul cycle.

Votre microprocesseur doit obligatoirement implanter un pipeline en **cinq phases** comme dans le livre du cours. Pour ce faire, nous vous fournissons une solution partielle qui comporte les éléments suivants :

1. Un module principal (`main`) que vous ne devriez **pas** modifier. Il contient le microprocesseur, un contrôleur de mémoire, ainsi que les mémoires d'instructions et de données. Vous devez travailler **exclusivement** à l'intérieur du sous-module nommé *Microprocesseur*. Notez que le module principal contient aussi deux afficheurs de 4 et 9 chiffres BCD. Le 1er sert à afficher le nombre de

1. https://github.com/logisim-evolution/logisim-evolution

Jeu d'instructions R12 (version 2.0)

Instruction	Encodage sur 12 bits												Description
	11	10	9	8	7	6	5	4	3	2	1	0	
nop	0	0	0	0	sans effet						0		Aucune opération
add	0	0	0	1	rd						1		Regs[rd] ← Regs[rs1] + Regs[rs2]
sub	0	0	1	0							2		Regs[rd] ← Regs[rs1] - Regs[rs2]
mult	0	0	1	1							3		Regs[rd] ← Regs[rs1] * Regs[rs2]
div	0	1	0	0							0		Regs[rd] ← Regs[rs1] / Regs[rs2]
mod	0	1	0	1							1		Regs[rd] ← Regs[rs1] % Regs[rs2]
and	0	1	1	0							2		Regs[rd] ← Regs[rs1] & Regs[rs2]
or	0	1	1	1							3		Regs[rd] ← Regs[rs1] | Regs[rs2]
xor	1	0	0	0							0		Regs[rd] ← Regs[rs1] ^ Regs[rs2]
-	1	0	0	1							1		Réservé pour expansion future
-	1	0	1	0							2		Réservé pour expansion future
not	1	0	1	1							3		Regs[rd] ← Regs[rs1] ^ 1_{12}
addi	1	1	0	0					imm (non signé)				Regs[rd] ← Regs[rs1] + imm
subi	1	1	0	1									Regs[rd] ← Regs[rs1] - imm
multi	1	1	1	0									Regs[rd] ← Regs[rs1] * imm
divi	1	1	1	1									Regs[rd] ← Regs[rs1] / imm
modi	1	1	1	1									Regs[rd] ← Regs[rs1] % imm
shli	1	1	1	1									Regs[rd] ← Regs[rs1] << imm
shri	1	1	1	1									Regs[rd] ← Regs[rs1] >> imm
ld	1	1	1	1									Regs[rd] ← MEM[Regs[rs1] + imm]
sd	1	1	1	1									MEM[Regs[rs1] + imm] ← Regs[rd]
jalr	1	1	1	1									Regs[rd] ← PC + 1, PC ← Regs[rs1] + imm
jal	1	1	1	1			imm (signé)						Regs[rd] ← PC + 1, PC ← PC + imm
bz	1	1	1	1	rs								if (Regs[rs] == 0) PC ← PC + imm
bnz	1	1	1	1									if (Regs[rs] != 0) PC ← PC + imm

FIGURE 1 – Jeu d'instructions du microprocesseur à concevoir.

cycles d'horloge depuis le début de l'exécution d'un programme. Il va nous servir à mesurer la performance de votre processeur. Le 2e afficheur permet d'afficher les valeurs qui sont écrites en mémoire aux adresses `0x000`, `0x001` et `0x002`.

2. Le module **Microprocesseur** contient les composants suivants :
   (a) des registres de phase IF/ID, ID/EX, EX/MEM et MEM/WB ;
   (b) un registre PC ;
   (c) une banque de quatre (4) registres avec deux (2) ports d'accès ;
   (d) un ALU pour les instructions arithmétiques et logiques ;
   (e) une unité fonctionnelle pour les branchements ;

3 Tâches à réaliser

Pour réaliser votre projet, vous devez :

1. Prendre le temps de bien lire le manuel du logiciel **Logisim**, car vous en ferez un usage **intensif**.
2. Maîtriser les différents composants de la **solution** partielle que nous vous fournissons. Assurez-vous de bien comprendre leur fonctionnement et n'hésitez pas à poser des questions lorsque des détails vous échappent.
3. Travailler dans le **cadre** que nous vous fournissons, et non tout refaire. Si vous refaites des choses, faites-les proprement, car cela pourrait rendre la correction **plus** difficile et, par conséquent, affecter votre note à la baisse.
4. Implanter **tous** les détails qui manquent, soit **beaucoup** de branchements entre les composants, la **logique** de contrôle du pipeline et sa mécanique de **blocage**, ainsi que la détection de **tous** les aléas.
5. Corriger **tous** les bogues qui pourraient exister dans ce que nous vous fournissons.
6. Rédiger un rapport qui met en valeur la **qualité** de votre travail et de vos résultats.
7. Votre rapport devrait fournir la performance, évaluée en nombre de coups d'horloge, pour accomplir chacun des programmes-tests (voir section 4).
8. L'entrée ready s'active (se met à ≪1≫ logique) lorsque qu'une donnée de mémoire est lue ou écrite dans la mémoire des données. Pour simplifier au maximum le travail du TP1, cette entrée est laissée à 1 en tout temps. Cependant, durant la conception du pipeline, il faut utiliser cette entrée sans présumer qu'elle sera toujours à 1.
9. L'implémentation du TP2 se base entièrement sur le TP1. Il est donc important de bien faire ce TP pour ne pas partir avec du retard pour le TP2. Le TP2 porte sur les latences de mémoire. L'entrée ready ne sera alors plus une constante.

4 Programmes de test

Nous vous fournissons cinq programmes pour tester l'architecture de votre pipeline. Les programmes seront décrits ci-dessous.

La figure 1 montre le jeu de données utilisé par tous les programmes à l'exception du premier. Les trois premières mémoires (0, 1 et 2) sont utilisées pour afficher des valeurs sur l'afficheur 7-segments. Les deux valeurs suivantes (3 et 4) sont utilisées pour arrêter l'horloge. En écrivant la valeur d'arrêt à l'adresse d'arrêt, le circuit d'horloge est automatiquement interrompu. Les deux valeurs suivantes (5 et 6) représentent respectivement la longueur d'un tableau de nombres et l'adresse de son premier élément. Finalement, le restant des valeurs forment une liste croissante de cent nombres allant de 1 à 100.

Data memory (fichier `data.o`)			
Address	Value	Comments	Binary
0	Undefined	reserved for digits[2 :0]	0x000
1	Undefined	reserved for digits[5 :3]	0x000
2	Undefined	reserved for digits[8 :6]	0x000
3	0x458	stop value	0x458
4	0xfff	stop address	0xfff
5	16	array length	0x010
6	7	array head pointer	0x007
7	1	first data of array	0x001
8	2	second data of array	0x002
...	...	...	...
106	100	last data of array	0x064

TABLE 1 – Programme pour la mémoire de données

La table 2 montre un programme `simple` d'une seule instruction, sans accès à la mémoire de données. Ce programme a pour objectif d'aider l'étudiant à trouver les erreurs dans son programme. Il est possible d'exécuter le programme un coup d'horloge à la fois afin de s'assurer que chaque étape se déroule comme prévu.

Ce programme ne fait qu'écrire la valeur 0xfff dans le registre R0. Ce programme est fonctionnel si après plusieurs coups d'horloge le résultat escompté est atteint.

Instruction memory (fichier `simple.o`)			
Address	Value	Comments	Binary
0	not R0, R0	R0 = fff	0x203

TABLE 2 – Programme trivial. Écriture dans un registre

La table 3 montre le programme `loadstore` capable d'arrêter l'horloge. L'utilisation des deux instructions *nop* produit le même effet que d'avoir inséré des bulles. L'objectif de ce programme est de

permettre de tester le pipeline très tôt dans sa conception sous un nombre limité de contraintes. Le test est réussi si après plusieurs coups d'horloge, l'horloge s'arrête.

Instruction memory (fichier `loadStore.o`)			
Address	Value	Comments	Binary
0	ld R0, R0, #3	R0 = stop value	0xa03
1	ld R1, R1, #4	R1 = stop address	0xa54
2	nop	insert stall	0x000
3	nop	insert stall	0x000
4	sd R0, R1, #0	stop clock	0xb10

TABLE 3 – Programme utilisant un load et un store

La table 4 montre le prochain programme à tester. Le programme `raw` force le pipeline à gérer un aléa. Le pipeline doit alors lui-même insérer les bulles au bon endroit. Ce test est réussi si après plusieurs coups d'horloge le nombre `03f` s'affiche à l'écran, puis l'horloge s'arrête.

Instruction memory (fichier `raw.o`)			
Address	Value	Comments	Binary
0	ld R0, R0, #3	R0 = stop value	0xa03
1	ld R1, R1, #4	R1 = stop address	0xa54
2	shri R2, R1, #6	R2 = 0x03f	0x996
3	sd R2, R3, #0	write digits[2 :0] = 0x3f	0xbb0
4	sd R0, R1, #0	stop clock	0xb10

TABLE 4 – Programme utilisant un load, un store et produisant des RAW

La table 5 montre le dernier programme simple à tester. Ce programme `branch` teste toutes les difficultés possibles à rencontrer dans un pipeline (Aléas, branchement et opérations de lecture et d'écriture). Ce test est réussi si après plusieurs coups d'horloge la valeur `003` s'affiche à l'écran, puis l'horloge s'arrête.

Instruction memory (fichier `branch.o`)			
Address	Value	Comments	Binary
0	addi R3, R3, #3	R3 = 3	0x3f3
1	bz R2, #2	Branch to address 3	0xe82
2	addi R3, R3, #4	R3 = 7	0x3f4
3	sd R3, R2, #0	write digits[2 :0] = 003	0xbe0
4	ld R0, R0, #3	R0 = stop value	0xa03
5	ld R1, R1, #4	R1 = stop address	0xa54
6	sd R0, R1, #0	stop clock	0xb10

TABLE 5 – Programme effectuant un load, un store et un branchement. La couleur relie l'instruction de branchement à son adresse de destination

Finalement, la table 6 montre un programme `performance` un peu plus gros qui a pour objectif de tester la performance de l'architecture de votre solution. Ce premier travail pratique vous donne la chance de vous familiariser avec l'ensemble du jeu de programmes de test. Ce premier TP offre peu de possibilités pour rendre sa solution plus performante. En revanche, le deuxième travail pratique se concentrera sur cet aspect avec l'introduction des chemins de traverse (Forwarding) et l'utilisation d'une cache. L'étudiant est appelé à s'interroger dès maintenant sur les ajouts à apporter au pipeline pour le rendre plus performant.

Ce programme additionne les nombres de 1 à 16 en affichant le total à chaque fois. Prendre note que la sortie est en hexadécimale. Les nombres `1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136` devraient s'afficher.

Instruction memory - `performance.o`			
Address	Value	Comments	Binary
0	ld R2, R2, #5	R2 = array length	0xaa5
1	ld R1, R1, #6	R1 = array start index pointer	0xa56
2	bz R2, #10	jmp to end if array length null	0xe8a
3	subi R2, R2, #1	Decrease number of remaining iterations	0x4a1
4	ld R0, R1, #0	Load next data	0xa10
5	addi R1, R1, #1	Pointer move to next data	0x351
6	sub R3, R3, R3	R3 = 0	0x0fe
7	ld R3, R3, #0	R3 = total	0xaf0
8	add R3, R3, R0	Total += data	0x0f1
9	sub R0, R0, R0	R0 = 0	0x002
10	sd R3, R0, #0	Write total in digits [2 :0]	0xbc0
11	bnz R2, #-8	Loop for remaining elements	0xfb8
12	sub R1, R1, R1	R1 = 0	0x056
13	ld R0, R1, #3	R0 = stop value	0xa13
14	ld R1, R1, #4	R1 = stop address	0xa54
15	sd R0, R1, #0	stop clock	0xb10

TABLE 6 – Programme effectuant beaucoup d'opérations I/O. Les couleurs respectives relient leur instruction de branchement à leur adresse de destination

Expérimentation avec le jeu d'instructions

Dans cette section nous vous demandons de créer un programme qui affiche les nombres de 0 à 59 sur deux interfaces 7-segments. Votre programme doit également gérer l'affichage pour que les nombres affichés soit en base décimale. En effet, afficher la valeur 10 produira `0A` sur l'afficheur. Il faudra afficher la valeur 16 pour voir inscrit `10`. De plus, votre programme doit respecter les spécifications suivantes :

— Le programme doit inscrire les chiffres 0 à 59 en ordre croissant sur les digits 7-segments [1 :0] à l'adresse 0.

— Vous pouvez utiliser la mémoire des données fournie ou encore créer la vôtre.
— Vous devez utiliser les instructions du jeu d'instructions R12
— Après le nombre 59, l'horloge doit s'arrêter

Il est bon de noter qu'une solution existe avec moins de 15 instructions. Bien que vous ne soyez pas limiter dans le nombre d'instructions (à l'exception de la taille de la mémoire de donnée) un design intelligent utilisant un boucle produira un programme plus court et facile à comprendre qu'un programme qui essayerait d'afficher les nombres un après l'autre.

Vous devez également accomplir les tâches suivantes dans votre rapport.
— Présenter votre programme dans un tableau
— Commenter votre programme pour qu'il soit le plus simple possible à comprendre
— Noter sa performance en coup d'horloge
— Joindre le fichier du programme (et sa mémoire si nécessaire) dans l'archive contenant votre rapport afin de pouvoir le faire fonctionner

Testabilité de l'architecture du microprocesseur

Les programmes de test fournis ont pour rôle de vous assister dans le débogage de votre implantation. Si un des tests ne fonctionne pas il y a nécessairement un problème avec l'implantation du microprocesseur. Cependant, la réussite de tous les tests ne garantit pas que le microprocesseur fonctionne sans erreur !

Plusieurs fonctionnalités risquent de ne pas avoir été testées avec les programmes tests et votre programme compteur. De plus, certains problèmes peuvent se révéler uniquement dans des circonstances particulières.

Il est de la responsabilité de l'étudiant de s'assurer que toutes les fonctionnalités du pipeline sont correctement implémentées. D'autres programmes *non-fournis* seront utilisés pour tester de manière plus approfondie votre solution.

L'étudiant est donc encouragé à tester lui-même sa solution en élaborant par lui-même une série de petits programmes servant à tester l'ensemble des instructions ainsi que les cas limites pour s'assurer que sa solution soit fonctionnelle.

5 Assembleur du R12

Un programme Python nommé `r12.py` vous est fourni afin de pouvoir aisément convertir en langage machine vos instructions en assembleur R12. Ce programme supporte le jeu d'instructions de la figure 1.

Exécutez simplement `./cr12.py prog.r12` pour compiler le fichier `prog.r12` contenant des instructions R12. Par défaut, les instructions codées en binaire seront affichées à la console. Pour les sauvegarder dans un fichier, dirigez simplement la console vers le fichier :

```
./cr12.py prog.r12 > prog.bin
```

Le compilateur supporte les formats d'instruction suivants :

```
0. nop
1. op rd, rs1
2. op rd, rs1, rs2
3. op rd, rs1, imm4
4. op rd, imm6
```

où `rd`, `rs1` et `rs2` sont l'un ou l'autre des quatre registres d'**architecture** R0, R1, R2 ou R3, et où `imm4` et `imm6` sont des valeurs entières (constantes) de quatre (4) ou six (6) bits. Notez que chaque instruction doit se trouver sur une ligne distincte, et que les lignes blanches sont permises. Il n'y a pour l'instant **aucun** support pour définir des étiquettes de branchement.

Ce logiciel vous est offert gracieusement, mais sans aucune garantie de bon fonctionnement.

6 Critères d'évaluation et rapport

Le projet sera évalué en utilisant les critères suivants :

1. **Fonctionnalité** (60%) : votre solution produit les bons résultats pour tous les programmes de test.
2. **Performance** (10%) : votre solution minimise les nombres de cycles nécessaires pour l'exécution des programmes de test.
3. **Esthétisme** (10%) : clarté et élégance de votre solution.
4. **Rapport** (20%) : votre rapport présente clairement votre solution et vos résultats.

Pour votre rapport, **évitez** les banalités et le verbiage inutile. Utilisez la structure suivante :

1. **Page titre** avec les noms des membres de l'équipe ainsi que le numéro de l'équipe.
2. **Solution** : autant que possible, illustrez votre solution afin de **faciliter** son évaluation. Qu'avez-vous fait de spécial qui **mérite** notre attention et qui n'est pas a priori évident ?
3. **Résultats et analyse** : présentez les résultats que vous avez obtenus avec nos programmes de test. Le cas échéant, présenter aussi d'autres résultats intéressants que vous auriez pu obtenir avec d'autres programmes. Comment votre microprocesseur se comporte-t-il ? Est-il sans bogue ? Est-il efficace ?
4. **Conclusion** : que doit-on retenir de votre travail ? Comporte-t-il des limitations ? En êtes-vous satisfait ? Si c'était à refaire, que changeriez-vous ?
5. **Format** : le rapport doit être remis en format PDF.

Bonne conception !