

Rapport du projet d'ASR 1

PAULIN Loïs, STAUB RUBEN

5 janvier 2016

1 Partie 1 : Processeur avec pipeline

1.1 Étage IF

Question 1 : Module IF :

Entrées : PC

Sorties : code de l'instruction correspondante + PC.

Question 2 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/628117f505d9bed9d1d7132adda3f4aa45548440/ProcoDeal.circ> ou ici pour le fichier brut.

1.2 Étage ID

Question 1 : Module ID :

Entrées : INS - PC

Sorties : OP - rT - SRC1 - SRC2 - ope0

Le module ID prend en entrée un code d'instruction et PC et retourne les différentes composante du code (op code, numéro du registre de retour, valeurs des registres en paramètres, valeur de la constante).

Question 2 : Register file :

Entrées : SRC1_in - SRC2_ - WE - TGT_sel - TGT

Sorties : SRC1 - SRC2

Register file contient les 8 registres, SRC1 et SRC2 sont le contenu des registres désignés par SRC1_in et SRC2_in, de plus lorsque WE vaut 1 TGT est stocké dans le registre TGT_sel.

Question 3 : CTL 7 :

Entrées : OP_in - rA

Sorties : OP_out - rT

CTL7 met rT à 000 si l'instruction ne modifie pas de registre sinon rT vaut rA. OP_out vaut toujours OP_in.

Question 4 : CTL 6 :

Entrées : OP

Sorties : s2 - op0

CTL6 s2 est à 1 si OP désigne ADD ou NAND, à 0 sinon. op0 est à 1 si OP désigne LUI, à 0 sinon.

Question 5 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/628117f505d9bed9d1d7132adda3f4aa45548440/ProcoDeal.circ> ou ici pour le fichier brut.

1.3 Étage EX

Question 1 : Module EX :

Entrées : OP - rT - PC - OPERAND0 - OPERAND1 - OPERAND 2

Sorties : SRC1 - SRC2

Register file contient les 8 registres, SRC1 et SRC2 sont le contenu des registres désignés par SRC1_in et SRC2_in, de plus lorsque WE vaut 1 TGT est stocké dans le registre TGT_sel.

Question 2 :

Question 3 :

Question 4 :

Question 5 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/628117f505d9bed9d1d7132adda3f4aa45548440/ProcoDeal.circ> ou ici pour le fichier brut.

1.4 Étage MEM

Question 1 :

Question 2 :

Question 3 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/628117f505d9bed9d1d7132adda3f4aa45548440/ProcoDeal.circ> ou ici pour le fichier brut.

1.5 Étage WB

Question 1 :

Question 2 :

Question 3 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/628117f505d9bed9d1d7132adda3f4aa45548440/ProcoDeal.circ> ou ici pour le fichier brut.

1.6 Pipeline

Question 1 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/628117f505d9bed9d1d7132adda3f4aa45548440/ProcoDeal.circ> ou ici pour le fichier brut.

1.7 Assembleur RiSC-16

Question 1 :

Question 2 :

2 Partie 2 : Pipeline avec logique bypass

2.1 Étage WB

Question 1 : Les entrées du module WB restent identiques.

Par contre les sorties rT et RF_WRITE_DATA sont dédoublées après le passage au travers du registre et directement dirigées vers le module EXE. Ainsi, ces données sont traitées comme des entrées conventionnelles de EXE (grâce à la synchronisation des registres après WB et avant EXE).

En effet, nous avons préféré implémenter le bypass principalement au niveau du pipeline, pour plus de simplicité au niveau des modules, et par soucis de modularité (il semble plus naturel que les bypass soient gérés par le pipeline que par les modules en eux-mêmes)

Question 2 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/3016e78db55d66684f28d8adf5c98ce117739173/ProcoDeal.circ> ou ici pour le fichier brut.

2.2 Étage MEM

Question 1 : (Voir l'étage WB) Les entrées du module WB restent identiques.

Par contre les sorties rT et RF_WRITE_DATA sont dédoublées après le passage au travers du registre et directement dirigées vers le module EXE. Ainsi, ces données sont traitées comme des entrées conventionnelles de EXE (grâce à la synchronisation des registres après WB et avant EXE).

En effet, nous avons préféré implémenter le bypass principalement au niveau du pipeline, pour plus de simplicité au niveau des modules, et par soucis de modularité (il semble plus naturel que les bypass soient gérés par le pipeline que par les modules en eux-mêmes)

Question 2 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/3016e78db55d66684f28d8adf5c98ce117739173/ProcoDeal.circ> ou ici pour le fichier brut.

2.3 Étage ID (Bonus)

Le module ID est également modifié puisque les entrées SCR1_in et SCR2_in du RegisterFile (dans l'étage ID) sont dupliquées et dirigées vers les sorties supplémentaires respectives s1 et s2 du module ID.

2.4 Étage EX

Question 1 : Module EXE : Entrées supplémentaires : s1 - s2 -

Question 2 :

Question 3 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/3016e78db55d66684f28d8adf5c98ce117739173/ProcoDeal.circ> ou ici pour le fichier brut.

2.5 Pipeline

Question 1 : Voir version correspondante : <https://github.com/Plopounet13/ProcoDeal/blob/3016e78db55d66684f28d8adf5c98ce117739173/ProcoDeal.circ> ou ici pour le fichier brut.

2.6 Assembleur RiSC-16

Question 1 : Dans le programme donné :

```
lui r2, 10
beq r1, r2, label
addi r2, r2, 1
```

Premièrement, l'instruction lui chargera les 10 bits de poids fort de 0000000000000010 et le reste à 0, donc 0000000000000000 dans r2...

Mais le vrai problème est que lors de l'instruction beq, le changement d'adresse de la prochaine instruction (adresse de l'instruction pointée par label) est calculée à partir de l'étage EXE. Ainsi, les 2 instructions suivantes (ici addi r2, r2, 1) seront exécutées avant que le changement de PC soit effectif.

Par conséquent, une version correcte de ce programme est :

```
addi r2, r0, 10
beq r1, r2, label
.space 2
addi r2, r2, 1
```

ou de manière équivalente :

```
addi r2, r0, 10
beq r1, r2, label
nop
nop
addi r2, r2, 1
```

Question 2 : Voir q7.3.S sur le git associé : <https://github.com/Plopounet13/ProcoDeal/blob/6962e645c41df63370ce54c08b1a13df091c0340/q7.3.S> ou ici pour le fichier brut.

3 Partie 3 : Mapping memory

3.1 Étage MEM

Question 1 :

3.2 Memory mapping

Question 1 :

Question 2 :

3.3 Pipeline

Question 1 :

3.4 Assembleur RiSC-16

Question 1 :

4 Introduction

Si tu es un peu rouillé en Latex, il y a quelques exemples après...

5 Some L^AT_EX Examples

5.1 How to Leave Comments

Comments can be added to the margins of the document using the `todo` command, as shown in the example on the right. You can also add inline comments :

This is an inline comment.

Here's a
comment
in the
margin!

5.2 How to Include Figures

First you have to upload the image file (JPEG, PNG or PDF) from your computer to writeLaTeX using the upload link the project menu. Then use the `includegraphics` command to include it in your document. Use the `figure` environment and the `caption` command to add a number and a caption to your figure. See the code for Figure ?? in this section for an example.

5.3 How to Make Tables

Use the `table` and `tabular` commands for basic tables — see Table 1, for example.

Item	Quantity
Widgets	42
Gadgets	13

TABLE 1 – An example table.

5.4 How to Write Mathematics

L^AT_EX is great at typesetting mathematics. Let X_1, X_2, \dots, X_n be a sequence of independent and identically distributed random variables with $E[X_i] = \mu$ and $\text{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \dots + X_n}{n} = \frac{1}{n} \sum_i^n X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

5.5 How to Make Sections and Subsections

Use section and subsection commands to organize your document. L^AT_EX handles all the formatting and numbering automatically. Use `ref` and `label` commands for cross-references.

5.6 How to Make Lists

You can make lists with automatic numbering ...

1. Like this,
2. and like this.

...or bullet points ...

- Like this,
- and like this.

...or with words and descriptions ...

Word Definition

Concept Explanation

Idea Text

We hope you find writeL^AT_EX useful, and please let us know if you have any feedback using the help menu above.