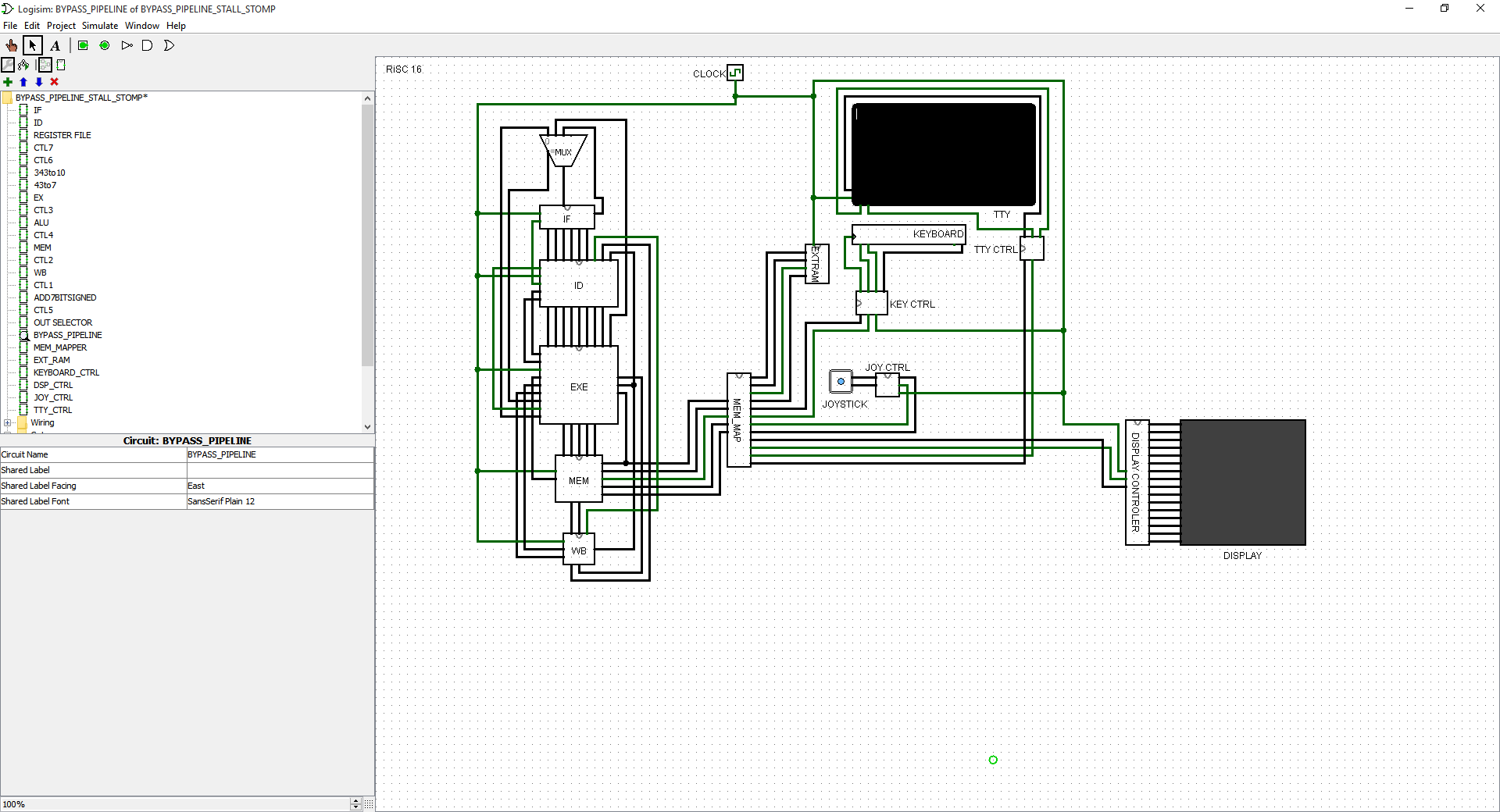
ASR1 – Projet

# RiSC 16-bits



Stérin Tristan

Torres Aurora Dugo Alexy

Note :

#### Tout au long de ce projet, nous avons décidé d’implémenter les registre inter-étages directement dans les étages du pipeline. Il en résulte un câblage du pipeline plus simple et lisible. De plus nous pensions que comme cela, les étages seraient plus modulables et utilisables dans d’autre montages.

# Pipeline simple

### Question 1.

1. IF : Module contenant les instructions du programme assembleur.
   1. Entrées :
      1. Program Counter (PC), contient l’adresse de l’instruction.
   2. Sorties :
      1. Operation Code (OP), le code de l’instruction à exécuter.
      2. rA, le registre A des instructions.
      3. rB, le registre B des instructions.
      4. mid, peut prendre plusieurs valeurs selon l’OP.
      5. rC, le registre C des instructions.
      6. PC + 1, l’adresse de la prochaine instruction dans la mémoire.
      7. CLOCK

### Question 2.

1. ID : Module chargeant les valeurs contenues dans les registres.
   1. Entrées :
      1. OP
      2. rA
      3. rB
      4. mid
      5. rC
      6. PC
      7. WErf, le write enable du register file, obtenu par CTL1 de l’étage WB.
      8. TGTr, le numéro du registre à accéder, obtenu dans l’étage WB.
      9. TGTd, la donnée à écrire dans le registre pointé, obtenu dans l’étage EX.
      10. CLOCK
   2. Sorties :
      1. OP
      2. rT, le registre rA renommé pour les étages suivants.
      3. PC
      4. OPERAND0
      5. OPERAND1
      6. OPERAND2

Pour les définitions formelles de OPERAND, se référer au schéma en annexe.

1. Register File : Module contenant les registres du processeur.
   1. Entrées :
      1. WErf
      2. TGTr
      3. TGTd
   2. Sorties :
      1. SRC1
      2. SRC2

SRC1 et SRC2 sont les données de sorties contenues dans les registres pointés.

1. CTL7 : Module CTL7
   1. Entrées :
      1. OP
      2. rA
   2. Sorties :
      1. rT
2. CTL6 : Module CTL6
   1. Entrées :
      1. OP
   2. Sortie :
      1. SelectMUXop0, permet de gérer le multiplexeur MUXs2
         1. Si sa valeur est 0, MUXs2 sélectionne rA (en cas de ADD ou NAND pour l’OP).
         2. Si sa valeur est 1, MUXs2 sélectionne rC.
      2. SelectMUXs2
         1. Si sa valeur est 0, MUXop0 sélectionne le décalage (en cas de LUI).
         2. Si sa valeur est 1, il sélectionne l’immédiat dans rC.

### Question 3.

1. EX : Module se chargeant d’effectuer le calcul demandé par l’instruction.
   1. Entrées :
      1. OP
      2. rT
      3. PC
      4. OPERAND0
      5. OPERAND1
      6. OPERAND2
      7. CLOCK
   2. Sorties :
      1. OP
      2. rT
      3. PC
      4. STORE DATA, les données à écrire dans la RAM.
      5. ALU OUTPUT, le résultat de l’ALU après l’opération requise.
      6. OPERAND1
      7. ADD (OPERAND0 + PC + 1)
      8. MUXpc, permet de gérer le choix de la prochaine instruction.
         1. Si sa valeur est 0, l’adresse de la prochaine instruction est égale à ADD.
         2. Si sa valeur est 1, la prochaine instruction est donnée pas OPERAND1.
         3. Si sa valeur est 2, la prochaine instruction est la consécutive dans la mémoire des instruction (soit PC + 1 à l’étage IF).
2. CTL3 : Module CTL3
   1. Entrées :
      1. OP
      2. EQ!, le résultat de l’ALU lors du teste d’égalité sur SRC1 et SRC2.
         1. Si sa valeur est 1, SRC1 et SRC2 sont égaux. 0 sinon.
   2. Sorties :
      1. FUNCalu, permet de gérer la fonction que doit prendre l’ALU.
         1. Si sa valeur est 0, l’ALU doit faire une addition signée sur 7 bits sur SRC1 et SRC2.
         2. Si sa valeur est 1, l’ALU doit faire une addition sur SRC1 et SRC2.
         3. Si sa valeur est 2, l’ALU doit effectuer un NAND sur SRC1 et SRC2.
         4. Si sa valeur est 3, l’ALU doit directement transmettre SRC2.
      2. MUXpc, défini plus haut dans les sorties de EX.
3. ALU : Unité arithmétique et logique.
   1. Entrées :
      1. SRC1
      2. SRC2
      3. FUNCalu
   2. Sorties :
      1. EQ!
      2. ALU OUTPUT
4. CTL4 : Module CTL4
   1. Entrées :
      1. OP
   2. Sorties :
      1. MUXimm, permet de gérer MUXimm
         1. SI sa valeur est 0, c’est que l’OP n’est ni JALR, ADD, NAND ou BEQ. MUXimm sélectionne alors OPERAND0
         2. Si sa valeur est 1, c’est que l’OP est JALR. MUXimm sélectionne PC + 1.
         3. Si sa valeur est 2, l’opération est soit ADD, NAND ou BEQ. MUXimm sélectionne alors OPERAND2.

### Question 4.

1. MEM : Module de chargeant le lire et écrire dans la mémoire RAM.
   1. Entrées :
      1. OP
      2. rT
      3. PC
      4. STORE DATA
      5. ALU OUTPUT
      6. CLOCK
   2. Sorties :
      1. rT
      2. RF WRITE DATA, la valeur lue ou écrie dans la mémoire.
2. CTL2 : Module CTL2
   1. Entrées :
      1. OP
   2. Sorties :
      1. WEdmem, permet d’activer l’écrire dans la mémoire.
      2. MUXout, permet de gérer MUXout
         1. Si sa valeur est 0, MUXout sélectionne la valeur lue dans la mémoire.
         2. Si sa valeur est 1, MUXout sélectionne la valeur de l’ALU OUTPUT.

### Question 5.

1. WB : Module permettant d’écrire les résultats des opérations des étages précédents dans les registres.
   1. Entrées :
      1. rT
      2. RF WRITE DATA
      3. CLOCK
   2. Sorties :
      1. RF WRITE DATA
      2. WErf, définit précédemment, le write-enable sur Register File.
2. CTL2 : Module CTL2
   1. Entrées :
      1. rT
   2. Sortie :
      1. WErf

### Question 7.

1. Le problème est la dépendance de la deuxième instruction sur le registre r2. En effet le pipeline n’aura pas fini de charger 0x8000 dans r2 quand lw aura besoin du contenu de r2. R2 ne contiendra pas encore la bonne valeur (WB ne sera pas effectué) et lw chargera une mauvaise adresse (voir plantera si la valeur dans r2 n’est pas bonne).

Correction :

Lui r2, 0x8000

Nop

Nop

Nop

Nop

Lw r1, r2, 1

1. Le fichier « TEST\_PIPELINE\_SIMPLE.asm » couvre toutes les opérations possibles tout en respectant les cycles d’attentes de finalisation des instructions critiques. Il se trouve dans le dossier « PIPELINE\_SIMPLE ».

# Pipeline avec logique Bypass

### Question 1.

1. WB :
   1. Entrées :
      1. rT
      2. RF WRITE DATA
      3. CLOCK
   2. Sorties :
      1. rT1, rT avant le registre vers CTL5.
      2. rT2, rT avant le registre vers TDTd
      3. rT3, rT après le registre vers CTL5.
      4. RF WRITE DATA1, avant le registre.
      5. RF WRITE DATA2, après le registre.
      6. WErf

### Question 2.

1. MEM :
   1. Entrées :
      1. OP
      2. rT
      3. PC
      4. STORE DATA
      5. ALU OUTPUT
      6. CLOCK
   2. Sortie :
      1. rT
      2. RF WRITE DATA
      3. rT1, vers CTL5
      4. ALU OUTPUT

### Question 3.

1. EX
   1. Entrées :
      1. OP
      2. rT
      3. s1, registre rB.
      4. s2, registre rA ou rC, selon le choix à l’étage ID.
      5. PC
      6. OPERAND0
      7. OPERAND1
      8. OPERAND2
      9. rT1, rT depuis le registre WB.
      10. rT2, rT depuis WB.
      11. rT3, rT depuis MEM.
      12. ALU OUTPUT, depuis MEM
      13. RF WRITE DATA1, depuis WB.
      14. RF WRITE DATA2, depuis le registre de WB.
      15. CLOCK
   2. Sorties :
      1. OP
      2. rT
      3. PC
      4. STORE DATA
      5. ALU OUTPUT
      6. MUXalu1, la valeur choisie par le MUXalu1
         1. Si sa valeur de choix est 0, MUXalu1 choisi ALU OUTPUT de l’étage MEM.
         2. Si sa valeur de choix est 1, MUXalu1 choisi RFWRITEDATA1.
         3. Si sa valeur de choix est 2, MUXalu1 choisi RFWRITEDATA2.
         4. SI sa valeur de choix est 3, MUXalu1 choisi OPERAND1.
      7. ADD
      8. MUXpc
2. CTL5 : Module CTL5
   1. Entrées :
      1. rT1
      2. rT2
      3. rT3
   2. Sorties :
      1. MUXalu1
         1. Définit plus haut.
      2. MUXalu2
         1. Si sa valeur de choix est 3, MUXalu1 choisi ALU OUTPUT de l’étage MEM.
         2. Si sa valeur de choix est 2, MUXalu1 choisi RFWRITEDATA1.
         3. Si sa valeur de choix est 1, MUXalu1 choisi RFWRITEDATA2.
         4. Si sa valeur de choix est 0, MUXalu1 choisi OPERAND2.

### Question 5.

* + - 1. Le registre r2 ne sera toujours pas prêt, même à l’aide des bypass. De plus, le pipeline, exécutant quand même l’instruction après le BEQ alors qu’il ne devrait pas. Il faudrait nettoyer le pipeline.

Correction :

Lui r2, 10

Nop

Beq r1, r2, label

Nop

Nop

Addi r2, r2, 1

1. Voir le fichier « TEST\_BYPASS\_PIPELINE.asm » dans le dossier « PIPELINE\_BYPASS ».

# Memory mapping

### Question 1.

* + - 1. MEM
         1. Entrées :

OP

rT

PC

STORE DATA

ALU OUTPUT

CLOCK

* + - * 1. Sorties :

ALU OUTPUT

STORE DATA OUT

CTL2 OUT

RFWRITE DATA OUT

rT OUT

rT1

### Question 2.

1. Memory mapping
   1. Choix des périphériques : RAM, TTY, KEYBOARD, VIDEO CARD, JOYSTICK
   2. Segmentation :
      1. 0x0000 : Joystick
      2. 0x0001 : Keyboard
      3. 0x0002 : Video Card
      4. 0x0003 : TTY

De 0x0000 à 0x0001 les périphériques sont en read only. De 0x0002 à 0x0003, en write only. Cette segmentation permet d’accéder simplement aux périphériques et ne fait perdre que 4 espaces mémoires.

De 0x0004 à 0xFFFF, la RAM est implémentée en lecture et écriture.

1. MEMORY MAPPER
   1. Entrées :
      1. ALU OUTPUT, l’adresse du périphérique / du segment RAM à atteindre.
      2. STORE DATA, la valeur à écrire dans le segment mémoire / périphérique.
      3. CTL2, le write enable de la RAM.
      4. OP
      5. KEYBOARD IN, la valeur lue par le périphérique KEYBOARD.
      6. JOYSTICK IN, la valeur lue par le JOYSTICK.
      7. RFWRITEDATA, la valeur lue dans la RAM.
   2. Sorties :
      1. RF WRITE DATA OUT, la valeur retournée par le périphérique / la RAM.
      2. MEM ADDRESS OUT, l’adresse mémoire à accéder dans la RAM.
      3. MEM DATA OUT, la donnée à enregistrer dans la RAM.
      4. CTL2 OUT
      5. READ KEYBOARD OUT, le read enable du KEYBOARD.
      6. READ JOY OUT, le read enable du JOYSTICK.
      7. WRITE DSP OUT, le write-enable du DISPLAY (video card).
      8. DISPLAY OUT, la valeur à écrire sur l’écran.
      9. WRITE TTY OUT, le write enable du TTY.
      10. TTY OUT, la valeur à écrire sur le TTY.

### Question 4.

Dans le dossier « PIPELINE\_BYPASS\_MEM\_MAPING »

* TEST\_DISPLAY : pour tester l’écran.
* TEST\_KEYBOARD : pour tester le clavier.
* TEST\_TTY : pour tester le TTY.
* TEST\_JOYSTICK : pour tester le Joystick, à tester avec le pipeline avec logique STALL STOMP.
* TEST\_TOTAL : pour tester l’intégralité des périphérique (programme complet, avec indication d’utilisation dans le code).

# Un pipeline avec logique Stall et Stomp

### Question 1.

1. Ajout de STOMP vers les registres inclus dans EX

Ajout de la sortie STOMP OUT vers ID afin de réaliser le STOMP sur les registre de cet étage.

### Question 2.

1. ID avait été modifié dans la question précédente : nous avons ajouté STOMP IN, l’entrée permettant d’effectuer un STOMP. Ceci est dû à notre choix d’implémentation des registres dans les étages directement.

Nous avons maintenant rajouté une sortie STALL OUT vers IF qui se voit ajouter une entrée STALL IN permettant d’effectuer un STALL.

### Question 3.

Dans le dossier « PIPELINE\_BYPASS\_MEM\_MAPING\_STALL\_STOMP »

* TEST\_DISPLAY : pour tester l’écran.
* TEST\_KEYBOARD : pour tester le clavier.
* TEST\_TTY : pour tester le TTY.
* TEST\_JOYSTICK : pour tester le Joystick.
* TEST\_TOTAL : pour tester l’intégralité des périphérique (programme complet, avec indication d’utilisation dans le code).

Les codes précédents ont été récupérés et modifiés (suppression des instructions NOP) afin d’illustrer le bon fonctionnement du pipeline.

# Annexes

