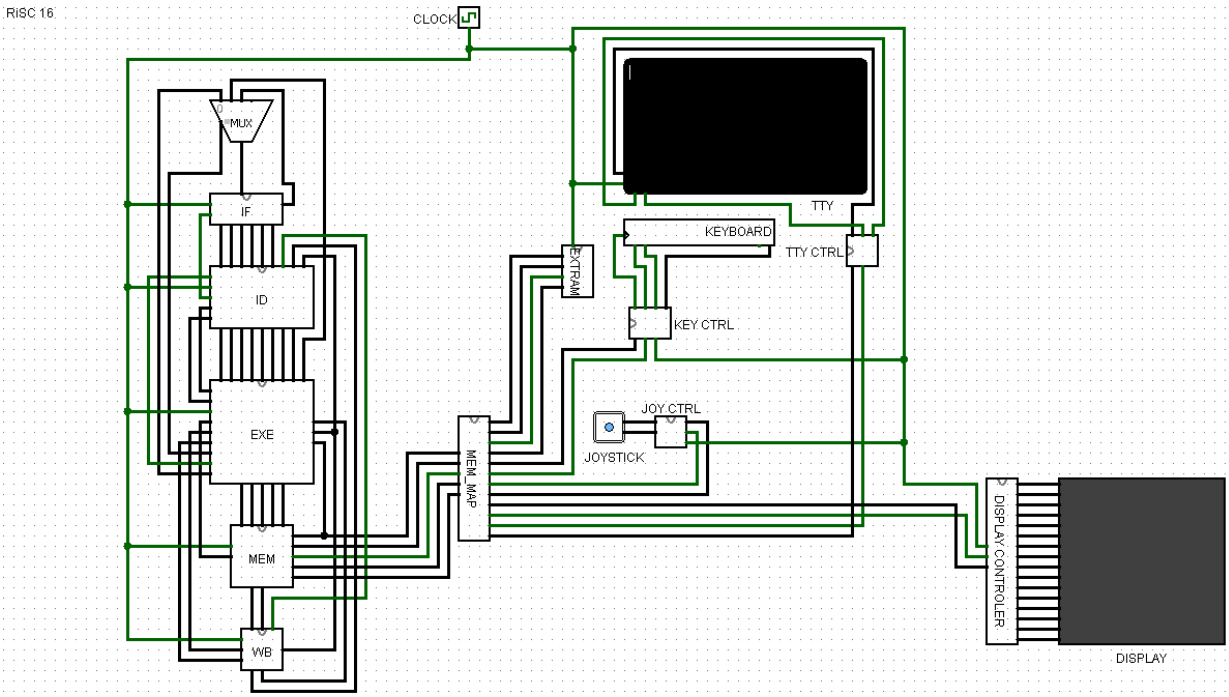
ASR1 – Projet

RiSC 16-bits



STERIN 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.
    a. Entrées :
        i. Program Counter (PC), contient l'adresse de l'instruction.
    b. Sorties :
        i. Operation Code (OP), le code de l'instruction à exécuter.
        ii. rA, le registre A des instructions.
        iii. rB, le registre B des instructions.
        iv. mid, peut prendre plusieurs valeurs selon l'OP.
        v. rC, le registre C des instructions.
        vi. PC + 1, l'adresse de la prochaine instruction dans la mémoire.
        vii. CLOCK

Question 2.

1. ID : Module chargeant les valeurs contenues dans les registres.
    a. Entrées :
        i. OP
        ii. rA
        iii. rB
        iv. mid
        v. rC
        vi. PC
        vii. WErf, le write enable du register file, obtenu par CTL1 de l'étage WB.
        viii. TGTr, le numéro du registre à accéder, obtenu dans l'étage WB.
        ix. TGTd, la donnée à écrire dans le registre pointé, obtenu dans l'étage EX.
        x. CLOCK
    b. Sorties :
        i. OP
        ii. rT, le registre rA renommé pour les étages suivants.
        iii. PC
        iv. OPERAND0
        v. OPERAND1
        vi. OPERAND2

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

2. Register File : Module contenant les registres du processeur.
    a. Entrées :
        i. WErf
        ii. TGTr
        iii. TGTd
    b. Sorties :
        i. SRC1
        ii. SRC2

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

3. CTL7 : Module CTL7
    a. Entrées :
        i. OP
        ii. rA
    b. Sorties :
        i. rT

4. CTL6 : Module CTL6
    a. Entrées :
        i. OP
    b. Sortie :
        i. 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.
        ii. 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.
    a. Entrées :
        i. OP
        ii. rT
        iii. PC
        iv. OPERAND0
        v. OPERAND1
        vi. OPERAND2
        vii. CLOCK
    b. Sorties :
        i. OP

ii. rT
iii. PC
iv. STORE DATA, les données à écrire dans la RAM.
v. ALU OUTPUT, le résultat de l'ALU après l'opération requise.
vi. OPERAND1
vii. ADD (OPERAND0 + PC + 1)
viii. 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
    a. Entrées :
        i. OP
        ii. 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.
    b. Sorties :
        i. 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.
        ii. MUXpc, défini plus haut dans les sorties de EX.

3. ALU : Unité arithmétique et logique.
    a. Entrées :
        i. SRC1
        ii. SRC2
        iii. FUNCalu
    b. Sorties :
        i. EQ!
        ii. ALU OUTPUT

4. CTL4 : Module CTL4
    a. Entrées :
        i. OP
    b. Sorties :
        i. 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.
    a. Entrées :
        i. OP
        ii. rT
        iii. PC
        iv. STORE DATA
        v. ALU OUTPUT
        vi. CLOCK
    b. Sorties :
        i. rT
        ii. RF WRITE DATA, la valeur lue ou écrie dans la mémoire.

2. CTL2 : Module CTL2
    a. Entrées :
        i. OP
    b. Sorties :
        i. WEdmem, permet d'activer l'écrire dans la mémoire.
        ii. 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.
    a. Entrées :
        i. rT
        ii. RF WRITE DATA
        iii. CLOCK
    b. Sorties :
        i. RF WRITE DATA
        ii. WErf, définit précédemment, le write-enable sur Register File.

2. CTL2 : Module CTL2
    a. Entrées :

i. rT

b. Sortie :

i. 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
```

2. 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 :
    a. Entrées :
        i. rT
        ii. RF WRITE DATA
        iii. CLOCK
    b. Sorties :
        i. rT1, rT avant le registre vers CTL5.
        ii. rT2, rT avant le registre vers TDTd
        iii. rT3, rT après le registre vers CTL5.
        iv. RF WRITE DATA1, avant le registre.
        v. RF WRITE DATA2, après le registre.
        vi. WErf

Question 2.

1. MEM :
    a. Entrées :
        i. OP
        ii. rT
        iii. PC
        iv. STORE DATA
        v. ALU OUTPUT
        vi. CLOCK
    b. Sortie :
        i. rT
        ii. RF WRITE DATA
        iii. rT1, vers CTL5
        iv. ALU OUTPUT

Question 3.

1. EX
    a. Entrées :
        i. OP
        ii. rT
        iii. s1, registre rB.
        iv. s2, registre rA ou rC, selon le choix à l'étage ID.
        v. PC
        vi. OPERAND0
        vii. OPERAND1

viii. OPERAND2
ix. rT1, rT depuis le registre WB.
x. rT2, rT depuis WB.
xi. rT3, rT depuis MEM.
xii. ALU OUTPUT, depuis MEM
xiii. RF WRITE DATA1, depuis WB.
xiv. RF WRITE DATA2, depuis le registre de WB.
xv. CLOCK

b. Sorties :
i. OP
ii. rT
iii. PC
iv. STORE DATA
v. ALU OUTPUT
vi. 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.
vii. ADD
viii. MUXpc

2. CTL5 : Module CTL5
a. Entrées :
i. rT1
ii. rT2
iii. rT3
b. Sorties :
i. MUXalu1
1. Définit plus haut.
ii. 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
```

2. Voir le fichier « TEST_BYPASS_PIPELINE.asm » dans le dossier « PIPELINE_BYPASS ».

Memory mapping

Question 1.

1. MEM
    a. Entrées :
        i. OP
        ii. rT
        iii. PC
        iv. STORE DATA
        v. ALU OUTPUT
        vi. CLOCK
    b. Sorties :
        i. ALU OUTPUT
        ii. STORE DATA OUT
        iii. CTL2 OUT
        iv. RFWRITE DATA OUT
        v. rT OUT
        vi. rT1

Question 2.

1. Memory mapping
    a. Choix des périphériques : RAM, TTY, KEYBOARD, VIDEO CARD, JOYSTICK
    b. Segmentation :
        i. 0x0000 : Joystick
        ii. 0x0001 : Keyboard
        iii. 0x0002 : Video Card
        iv. 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.

2. MEMORY MAPPER
    a. Entrées :
        i. ALU OUTPUT, l'adresse du périphérique / du segment RAM à atteindre.
        ii. STORE DATA, la valeur à écrire dans le segment mémoire / périphérique.
        iii. CTL2, le write enable de la RAM.
        iv. OP
        v. KEYBOARD IN, la valeur lue par le périphérique KEYBOARD.
        vi. JOYSTICK IN, la valeur lue par le JOYSTICK.
        vii. RFWRITEDATA, la valeur lue dans la RAM.

b. Sorties :
   i. RF WRITE DATA OUT, la valeur retournée par le périphérique / la RAM.
   ii. MEM ADDRESS OUT, l'adresse mémoire à accéder dans la RAM.
   iii. MEM DATA OUT, la donnée à enregistrer dans la RAM.
   iv. CTL2 OUT
   v. READ KEYBOARD OUT, le read enable du KEYBOARD.
   vi. READ JOY OUT, le read enable du JOYSTICK.
   vii. WRITE DSP OUT, le write-enable du DISPLAY (video card).
   viii. DISPLAY OUT, la valeur à écrire sur l'écran.
   ix. WRITE TTY OUT, le write enable du TTY.
   x. 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.

3. 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.

3. 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.

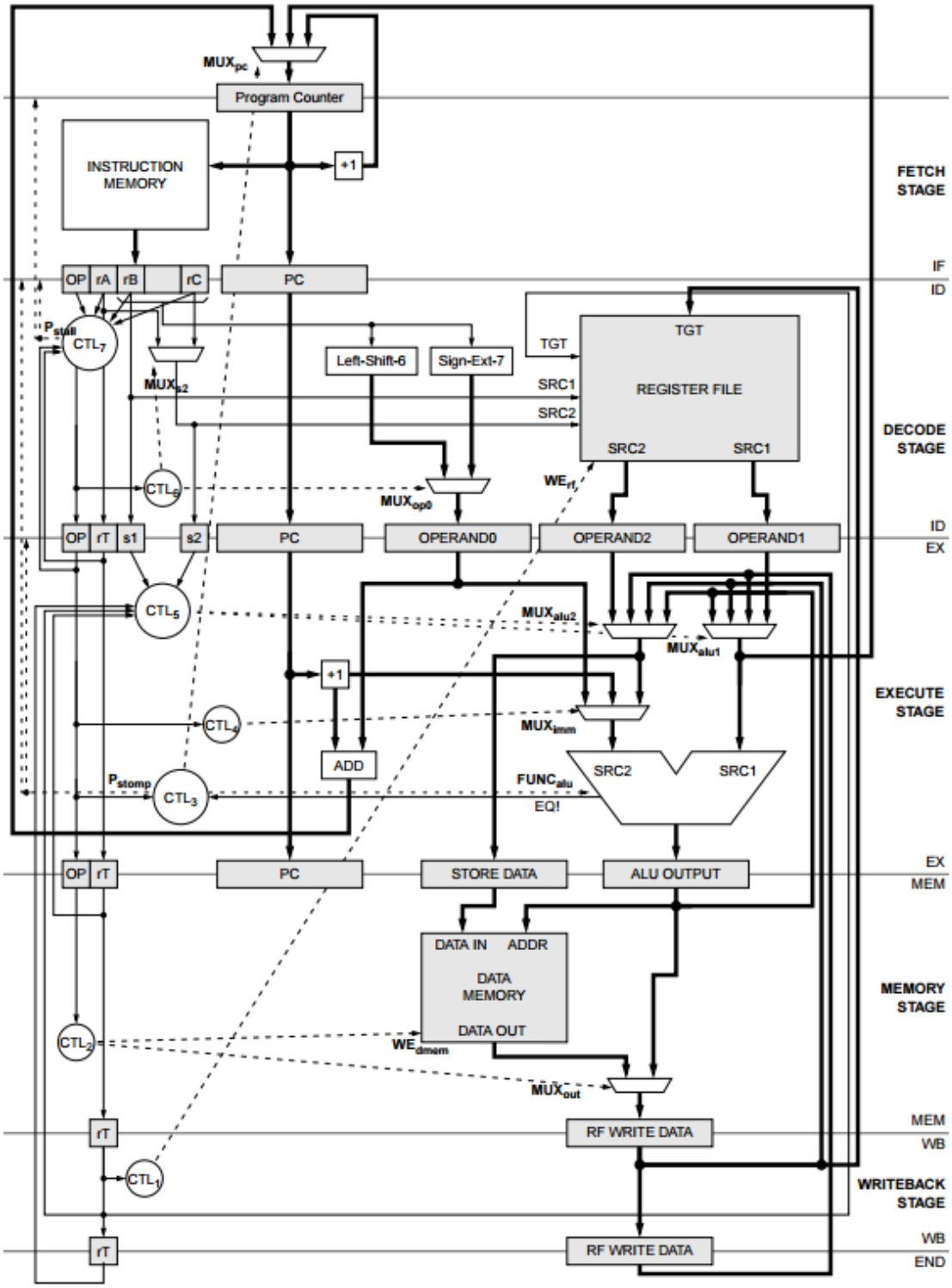
Question 3.

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


MUX_pc
Program Counter
INSTRUCTION MEMORY
+1
FETCH STAGE
OP
rA
rB
rC
PC
IF
ID
P_stall
CTL_7
MUX_s2
Left-Shift-6
Sign-Ext-7
TGT
TGT
REGISTER FILE
SRC1
SRC2
SRC2
SRC1
DECODE STAGE
CTL_6
MUX_op0
WE_rf
OP
rT
s1
s2
PC
OPERAND0
OPERAND2
OPERAND1
ID
EX
CTL_5
MUX_alu2
MUX_alu1
+1
EXECUTE STAGE
CTL_4
MUX_imm
ADD
SRC2
SRC1
P_stomp
CTL_3
FUNC_alu
EQ!
OP
rT
PC
STORE DATA
ALU OUTPUT
EX
MEM
DATA IN
ADDR
DATA MEMORY
DATA OUT
MEMORY STAGE
CTL_2
WE_dmem
MUX_out
rT
RF WRITE DATA
MEM
WB
CTL_1
WRITEBACK STAGE
rT
RF WRITE DATA
WB
END