Université d'Alger 1

Faculté des Sciences
Département Mathématiques et Informatique

Examen L2-ADO
17-Février-2020

Nom : Prénom : Groupe : Matricule :		/ 20

Exercice 1 : (6 points)

Dans une équipe de reverse-engineering, votre patron vous a attribué la tache de désassembler le code d'un concurrent industriel et ce dans le but d'implémenter une solution similaire.

1/ Complétez le décodage des instructions machines indiquées dans les commentaires (5 points)

Adresse de l'instruction	Les étiquettes	Code assembleur	Commentaires
			# Instructions machines à décoder. Ajouter des étiquettes au code si nécessaire !
0x0040 0000		addu $s0 , $zero , $a0	
0x0040 0004		addu $s1 , $zero , $a1	
0x0040 0008		addiu $v0 , $zero , 0	
0x0040 000C		addiu $v1 , $zero , 0	
0x0040 0010	(0.25) L1:	**beq $zero , $s0 , FIN**	# <-- 0x1200000a (0.5 pt)
0x0040 0014		**lw $t0 , 0($s1)**	# <-- 0x8e280000 (0.5 pt)
0x0040 0018		**andi $t8 , $t0 , 1**	# <-- 0x31180001 (0.5 pt)
0x0040 001C		**bne $zero , $t8 , L3**	# <-- 0x17000002 (0.5 pt)
0x0040 0020		addi $v0 , $v0 , 1	
0x0040 0024		**j L0**	# <-- 0x0810000b (0.5 pt)
0x0040 0028	(0.25) L3:	addi $v1 , $v1 , 1	
0x0040 002C	(0.25) L0:	**addi $s1 , $s1 , 4**	# <-- 0x22310004 (0.5 pt)
0x0040 0030		addi $at , $zero , 1	
0x0040 0034		sub $s0 , $s0 , $at	
0x0040 0038		**j L1**	# <-- 0x08100004 (0.5 pt)
0x0040 003C	(0.25) FIN:	**jr $ra**	# <-- 0x03e00008 (0.5 pt)

2/ Que fait donc ce programme ? (1 point)

Votre réponse ici ne sera prise en compte que si votre code assembleur est correct.

une fonction pour compter les nombres **pairs et impairs** dans une **séquence de nombres en mémoire**

Exercice 2 : (7 points)

1/ Considérez la définition suivante en assembleur MIPS (**2 points**) :

```
Array :     .word 10, 11, 12, 13, 14
```

Soit la séquence d'instructions MIPS suivante :

```
la $t0, Array
lw $t0, 4($t0)
```

Quel est le contenu du registre **$t0** (**en hexadécimal**) après avoir exécuté cette séquence ? 0x0000000B

2/ Soit le contenu de la mémoire suivant, où chaque case représente un seul octet en mémoire.

Adresse	+0	+1	+2	+3	+4	+5	+6
0x10010020	0xfa	0x20	0x10	0xC0	0xB0	0x5f	0x94

Donnez les valeurs des registres **$t0** à **$t4** en hexadécimal après l'exécution des instructions ci-dessous. L'ordre little-endian des octets doit être utilisé. Supposez que **$s0 = 0×10010020**. (**5 points**)

`lw $t0, 0($s0)`	**$t0 = 0xC01020fa**	**(1 point)**
`lh $t1, 2($s0)`	**$t1 = 0xffffc010**	**(1 point)**
`lhu $t2, 4($s0)`	**$t2 = 0x00005fb0**	**(1 point)**
`lb $t3, 5($s0)`	**$t3 = 0x0000005f**	**(1 point)**
`lbu $t4, 6($s0)`	**$t4 = 0x00000094**	**(1 point)**

Exercice 3 : (7 points)

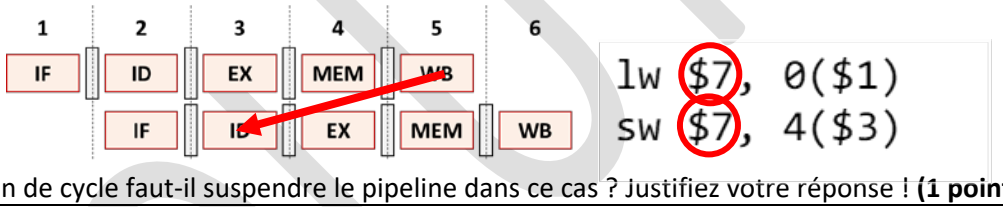
Nous avons vu dans le cours que les instructions de lecture depuis la mémoire génèrent des aléas véritables de donnée (c.-à-d. la donnée nécessaire pour les prochaines instructions n'est pas encore disponible dans le pipeline). Nous avons vu aussi que ce type d'aléa est géré avec la condition suivante

```
if( ID/EX.MemRead == 1 &&
   (ID/EX.RegisterRt == IF/ID.RegisterRs || ID/EX.RegisterRt == IF/ID.RegisterRt ) )
{
   /* alors pause du pipeline */
}
```

Soit les instructions assembleurs suivantes :

```
lw $7, 0($1)
sw $7, 4($3)
```

1/ Dans le diagramme ci-dessous, indiquez par une flèche le flux de donnée exposant l'aléa de donnée lié aux instructions précédentes et mettez un cercle autour de la donnée problématique **(1 point)**.



2/ Combien de cycle faut-il suspendre le pipeline dans ce cas ? Justifiez votre réponse ! **(1 point)**

Un cycle d'horloge suffit. Cela nous permettra de récupérer la donnée destinée au registre $7 depuis le registre MEM.WB et la transférée à l'instruction suivante (sw) – si l'unité de transfert n'était pas utilisée nous serions obligé de suspendre le pipeline pour deux cycles d'horloge pour permettre à l'instruction sw de récupérer sa donnée depuis le banc de registre dans l'étape (ID).

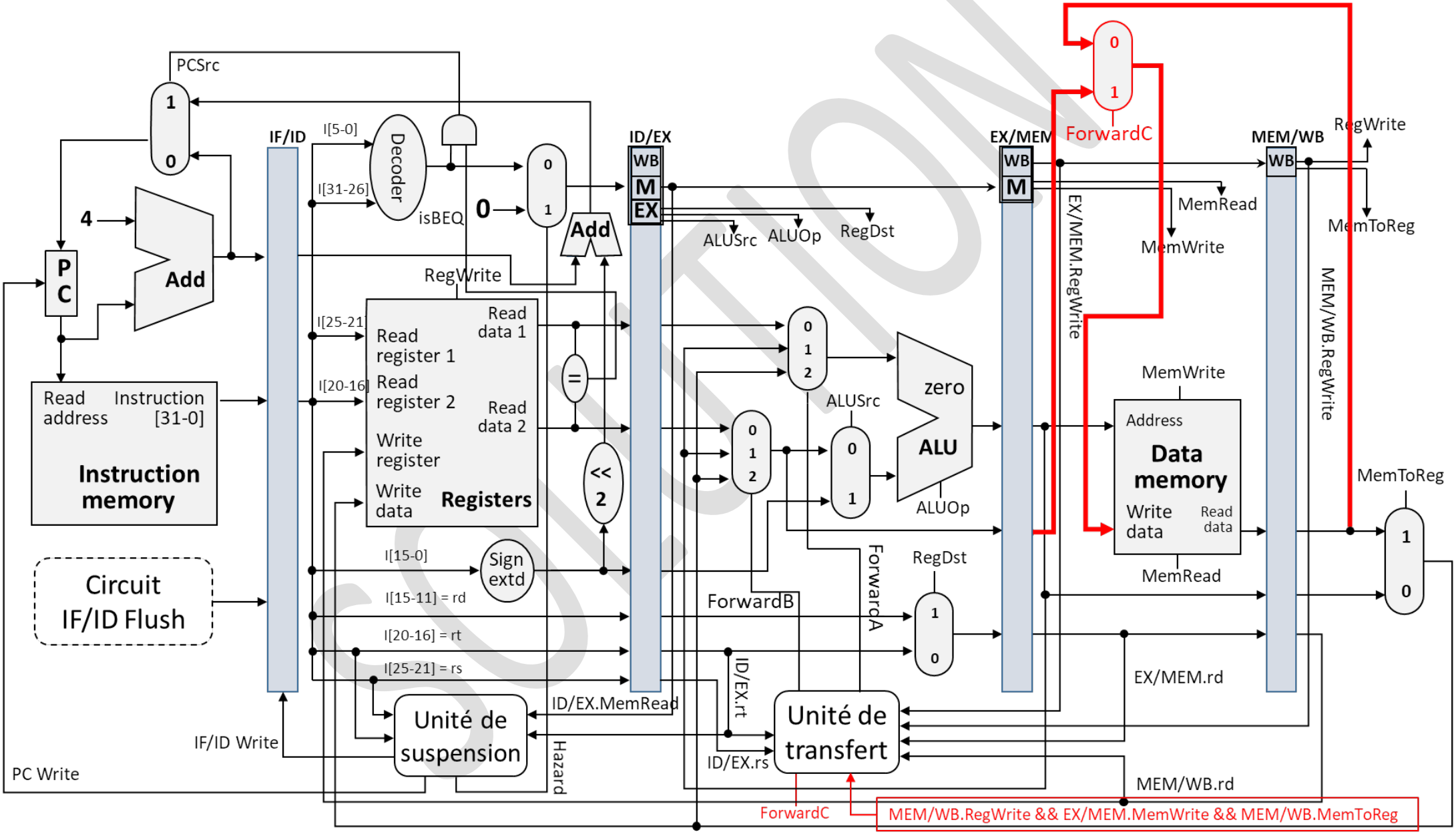
3/ Il est possible d'éviter de suspendre le pipeline pour ce cas précis si notre chemin de donnée est modifié d'une manière adéquate. Proposez vos modifications dans le schéma suivant **(2 points)** et donnez la nouvelle règle qui permet de gérer cet aléa **(2 points)**

```
RegDst = 0 /* D'abord, nous nous assurons que RegDst == 0 pour les instructions 'store'... */

if( MEM/WB.MemToReg == 1 && MEM/WB.RegWrite == 1 && EX/MEM.MemWrite == 1 && MEM/WB.RegisterRd
== EX/MEM.RegisterRd )
       ForwardC = 0;
else   ForwardC = 1;
```

4/Changez la condition citée dans l'énoncé de l'exercice pour prendre en compte votre modification ? **(1 points)**

```
if( ID/EX.MemRead == 1 &&
     ( ( ID/EX.RegisterRt == IF/ID.RegisterRt && IF/ID.MemWrite == 0 )  ||
         ID/EX.RegisterRt == IF/ID.RegisterRs                          ))
 {   /* alors on suspend le pipeline */ }
```


PCSrc
1
0
4
Add
P
C
Read address
Instruction [31-0]
Instruction memory
Circuit IF/ID Flush
IF/ID
I[5-0]
I[31-26]
Decoder
isBEQ
0
0
1
Add
RegWrite
I[25-21]
I[20-16]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
=
<< 2
I[15-0]
Sign extd
I[15-11] = rd
I[20-16] = rt
I[25-21] = rs
ID/EX
WB
M
EX
ALUSrc
ALUOp
RegDst
0
1
2
ALUSrc
zero
ALU
ALUOp
ForwardB
ForwardA
RegDst
1
0
EX/MEM
WB
M
ForwardC
0
1
EX/MEM.RegWrite
MemRead
MemWrite
MEM/WB
WB
RegWrite
MemToReg
MEM/WB.RegWrite
MemWrite
Address
Data memory
Write data
Read data
MemRead
MemToReg
1
0
EX/MEM.rd
MEM/WB.rd
ID/EX.MemRead
ID/EX.rt
ID/EX.rs
Unité de suspension
Unité de transfert
Hazard
IF/ID Write
PC Write
ForwardC
MEM/WB.RegWrite && EX/MEM.MemWrite && MEM/WB.MemToReg