Pràctica 4: Introducció al RISC 5-Stage Processor en Ripes

Introducció / Objectius

En aquesta pràctica visualitzarem els diferents passos que ha de fer un microprocessador per tal d'executar una instrucció. Veurem que una microinstrucció és la part d'una instrucció que s'executa en un cicle de rellotge del microprocessador. Llavors, comprovarem que no totes les instruccions que usem en els nostres programes ocupen les mateixes microinstruccions i, per tant, no tarden els mateixos cicles de rellotge. En aquesta pràctica veurem com passa això en uns programes determinats.

- Observar les diferents microinstruccions d'algunes instruccions.
- Observar com varien les microinstruccions en certes instruccions segons si es completen o no (com en el cas dels *branchs*).
- Observar el seguit de microinstruccions a través d'un programa sencer.
- Entendre el comportament i funcionament de les microinstruccions.
- Entendre i saber analitzar un cronograma.

Exercici guiat

Haurem d'executar els següents codis, cicle a cicle (microinstrucció a microinstrucció) utilitzant com a microprocessador el *5-Stage Processor* al simulador Ripes:

Codi 1:

```
.data
resultat: .word 0

.text
main:
   add a3, zero, zero
   add a7, zero, zero
   addi a2, zero, 4
   add a3, a2, a3
   addi a7, a7, -1
   la a0, resultat
   sw a3, 0(a0)
```

Execució cicle a cicle:

1. Inicialitza a 0 el registre **a3**
2. Inicialitza a 0 el registre **a7.**
3. Guarda el valor 4 en el registre **a2**.
4. Suma el valor del registre **a3** (0) més el valor del registre **a2** (4) i ho guarda en el registre **a3**, el qual passarà a guardar el valor 4.
5. Suma el valor del registre **a7** (0) amb l'immediat -1 i ho guarda en el registre **a7,** que ara tindrà un valor de -1.
6. La posició de memòria ***resultat*** es guarda en el registre **a0**.
7. Fem un direccionalment relatiu en el que sumem la posició de memòria de **a0** més l'offset que és 0 i hi guardem el valor del registre **a3** (4).

Codi 2:

```
.data
resultat: .word 0
.text
main:
add  a3, zero, zero
add  a7, zero, zero
addi a2, zero, 4
add  a3, a2, a3
addi a7, a7, -1
blt  a7, zero, salta
salta:
la   a0, resultat
sw   a3, 0(a0)
```

Execució cicle a cicle:

1. Inicialitza a 0 el registre **a3**
2. Inicialitza a 0 el registre **a7.**
3. Guarda el valor 4 en el registre **a2**.
4. Suma el valor del registre **a3** (0) més el valor del registre **a2** (4) i ho guarda en el registre **a3**, el qual passarà a guardar el valor 4.
5. Suma el valor del registre **a7** (0) amb l'immediat -1 i ho guarda en el registre **a7,** que ara tindrà un valor de -1.
6. Ara tenim un salt condicional (*blt, branch if lower than*) que saltarà a la direcció que indica l'etiqueta ***salta*** quan el valor del registre **a7** sigui menor a zero. Que com veiem ja ho és pel pas nº 5, així que saltem a la direcció ***salta***.
7. La posició de memòria ***resultat*** es guarda en el registre **a0**.
8. Fem un direccionalment relatiu en el que sumem la posició de memòria de **a0** més l'offset que és 0 i hi guardem el valor del registre **a3** (4).

Preguntes

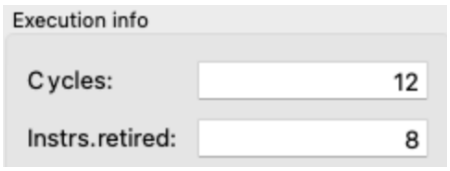
1) **Abans d'executar els codis tracta d'esbrinar la seva funcionalitat. Faran el mateix? Creus que trigaran el mateix nombre de cicles en executar-se?**

   Com hem comentat abans descrivint el codi, la seva funcionalitat és la de realitzar un seguit d'operacions aritmètiques amb i sense immediats i emmagatzemar els resultats als registres corresponents.

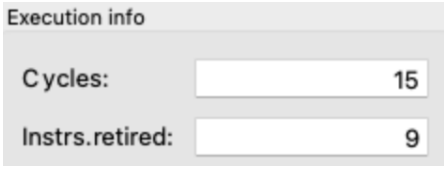
   Tot i presentar-se la petita diferència del salt condicional (***blt a7, zero, salta***) entre el codi 1 i el codi 2 i l'etiqueta ***salta***, els dos codis faran el mateix ja que com hem vist el salt condicional sempre es complirà, igualment tant si s'executés com si no, la següent instrucció en ser executada sempre serà la de la direcció ***salta*** en el cas del codi 2, pel que podem afirmar que la instrucció de salt condicional és totalment innecessària.

   Degut a això, el codi 1 hauria de trigar 3 cicles de *clock* menys en executar-se. Ja que la instrucció haurà de tornar a començar un altre cop el *Fetch* i la descodificació. Perquè hem d'esborrar del pipeline la direcció de salt, ha d'haver-hi una nova. Per tant, perdrem dos cicles de rellotge. 1 cicle per afegir la instrucció del salt condicional i 2 per haver de recalcular el salt.

2) Ves a la finestra del Ripes dedicada al processador (*Processor*). Busca entre les opcions del *Simulator Control*, la *Pipeline table*. Executa els dos codis cicle a cicle fixant-te com les instruccions van passant per les diferents etapes del pipeline. Compta el nombre de cicles que es necessiten per executar cadascun dels codis i compara les pipeline tables.



Cicles del codi 1



Cicles codi 2

	0	1	2	3	4	5	6	7	8	9	10	11	12
add x13 x0 x0	IF	ID	EX	MEM	WB								
add x17 x0 x0		IF	ID	EX	MEM	WB							
addi x12 x0 4			IF	ID	EX	MEM	WB						
add x13 x12 x13				IF	ID	EX	MEM	WB					
addi x17 x17 -1					IF	ID	EX	MEM	WB				
auipc x10 0x10000						IF	ID	EX	MEM	WB			
addi x10 x10 -20							IF	ID	EX	MEM	WB		
sw x13 0(x10)								IF	ID	EX	MEM	WB	

Pipeline table codi 1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
add x13 x0 x0	IF	ID	EX	MEM	WB											
add x17 x0 x0		IF	ID	EX	MEM	WB										
addi x12 x0 4			IF	ID	EX	MEM	WB									
add x13 x12 x13				IF	ID	EX	MEM	WB								
addi x17 x17 -1					IF	ID	EX	MEM	WB							
blt x17 x0 4 <salta>						IF	ID	EX	MEM	WB						
auipc x10 0x10000							IF	ID	IF	ID	EX	MEM	WB			
addi x10 x10 -24								IF		IF	ID	EX	MEM	WB		
sw x13 0(x10)											IF	ID	EX	MEM	WB	

Pipeline table codi 2

3) Per què les etapes de la instrucció *auipc x10 0x 10000* al pipline son *IF ID IF IF EX MEM WB* al codi 2?

Això és degut al *flush* perquè com havia de calcular un altre cop la direcció de salt, i és per això que ha de tornar a fer les següents dues etapes de la instrucció: ***IF*** i ***ID***, és a dir, ha de començar un altre cop el *Fetch* i la descodificació perquè hem d'esborrar del pipeline la direcció de salt, ha d'haver-hi una nova. Per tant,

perdrem dos cicles de rellotge. Per tant 1 cicle per afegir la instrucció del salt condicional i 2 per haver de recalcular el salt. Per aquest motiu, com ha de tornar a fer, la instrucció ja no consisteix en 5 etapes (cicles de rellotge) sinó 7.

4) Com afectaria al nombre total de cicles d'execució el següent canvi en el codi:

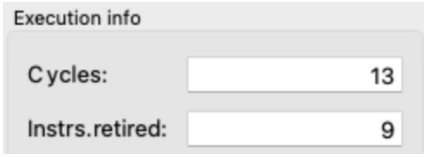
Codi 1

```
.data
resultat: .word 0
.text
main:
add a3, zero, zero
add a7, zero, zero
addi a2, zero, 4
add  a3, a2, a3
addi  a7, a7, -1
la  a0, resultat
sw  a3, 0(a0)
```

Codi 3

```
.data
resultat: .word 0
.text
main:
add  a3, zero, zero
add  a7, zero, zero
addi a2, zero, 4
add  a3, a2, a3
addi a7, a7, -1
bgt  a7, zero, salta
salta:
la    a0, resultat
sw    a3, 0(a0)
```

En aquest cas la condició de salt no es compleix, per tant no s'han esborrat les següents instruccions perquè no s'ha complert la condició. En aquest cas no hi ha penalització, s'executarà el pipelie norma, i llavors en comptes de ser 15 cicles seran 13, perquè els 2 cicles de penalització no hi són.



Realització de la pràctica

Ripes: executa el següent programa

Codi 4

```
.data
valorDada: .word 2
guardaResultat: .word 0
.text
main:
la  a0, valorDada
lw  a7, 0(a0)
addi  a2, zero, 9
add  a3, zero, zero
loop:
add  a3, a2, a3
addi  a7, a7, -1
bgt  a7, zero, loop
la  a0, guardaResultat
sw  a3, 0(a0)
```

Preguntes sobre el simulador RIPES:

1. **Quin és l'estat de cadascuna de les cinc etapes del pipeline al cicle 6? I al 8?**

	0	1	2	3	4	5	6
auipc x10 0x10000	IF	ID	EX	MEM	WB		
addi x10 x10 0		IF	ID	EX	MEM	WB	
lw x17 0(x10)			IF	ID	EX	MEM	WB
addi x12 x0 9				IF	ID	EX	MEM
add x13 x0 x0					IF	ID	EX
add x13 x12 x13						IF	ID
addi x17 x17 -1							IF
blt x0 x17 -8 <loop>							
auipc x10 0x10000							
addi x10 x10 -28							
sw x13 0(x10)							

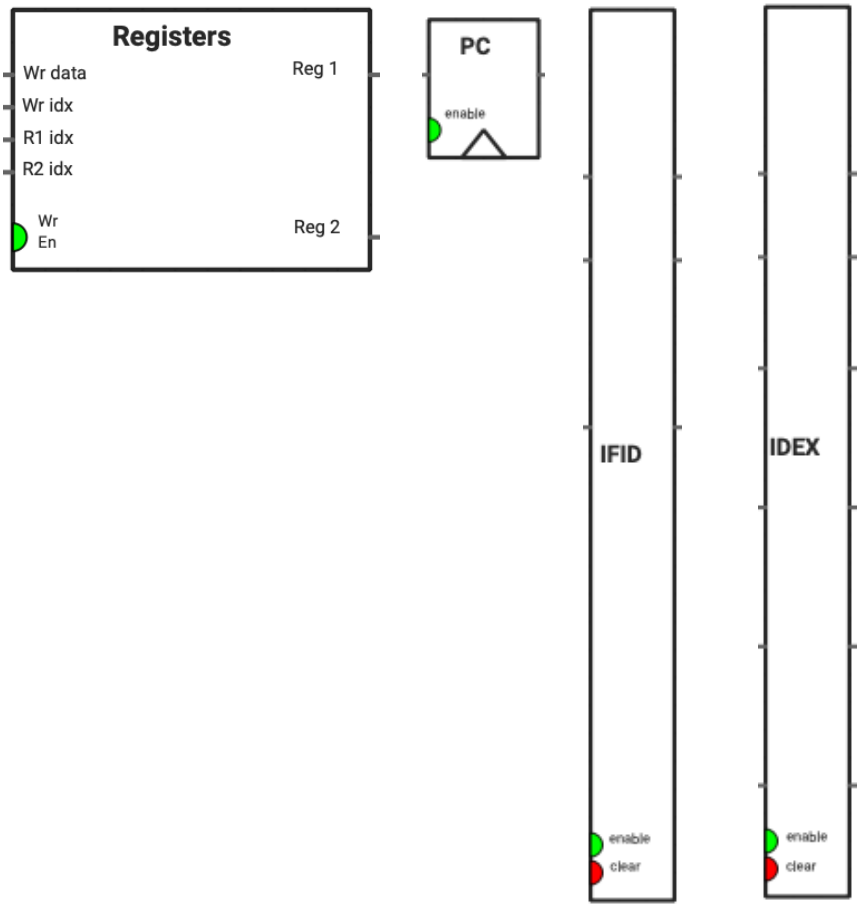
Pipeline al cicle 6

	0	1	2	3	4	5	6	7	8
auipc x10 0x10000	IF	ID	EX	MEM	WB				
addi x10 x10 0		IF	ID	EX	MEM	WB			
lw x17 0(x10)			IF	ID	EX	MEM	WB		
addi x12 x0 9				IF	ID	EX	MEM	WB	
add x13 x0 x0					IF	ID	EX	MEM	WB
add x13 x12 x13						IF	ID	EX	MEM
addi x17 x17 -1							IF	ID	EX
blt x0 x17 -8 <loop>								IF	ID
auipc x10 0x10000									IF
addi x10 x10 -28									
sw x13 0(x10)									

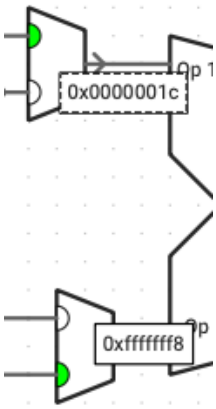
Pipeline al cicle 8

2. **Quins senyals de control s'activen en el cicle 4 a la segona etapa del pipeline? A quines instruccions del codi corresponen?**

   Adjunto imatges dels senyals de control activats en el cicle 4 a la segona etapa del pipeline. El *write* correspon a ***auipc x10 0x 10000*** i el read a ***lw x7 0(x17)***.



3. **Quins són els valors a les sortides dels multiplexors assenyalats a la figura al cicle 9?**

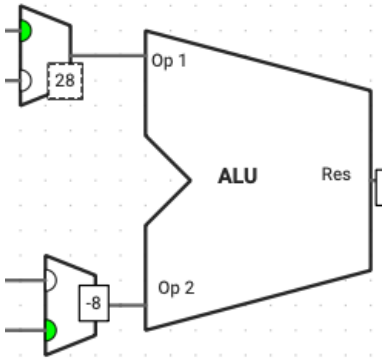


   El primer multiplexor té un valor a la sortida de: **0x0000001c**.
   El segon multiplexor té un valor a la sortida de: **0xffffffff8**.

4. **Què està calculant la ALU al cicle 9?**

   Està realitzant la instrucció de salt condicional ***bgt a7, zero, loop***. Està comprovant si el valor del registre **a7** és més gran que 0, si és així, saltarà a ***loop***.

Més concretament està calculant la nova posició de memòria, està calculant el program counter actual més l'immediat que l'hi ha entrat: **28– 8**.



5. **Llegeix amb cura la part del codi amb que s'implementa el loop. Tenint en compte el que has vist a la qüestió 3), justifica perquè el pipeline al cicle 10 presenta aquest estat:**

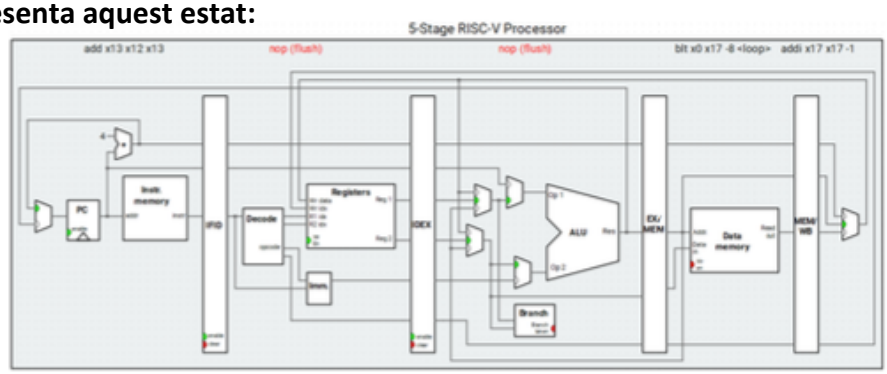


Figura 5. Estat del pipeline al cicle 10.

El pipeline presenta aquest estat perquè la instrucció ***blt z0 z17 -8*** depèn del resultat de ***addi x17 x17 -1***, i per tant ha d'esperar a que aquesta s'avaluï. Ja que en el cas de que la instrucció s'executi, haurà de tornar a començar un altre cop el *Fetch* i la descodificació. Perquè hem d'esborrar del pipeline la direcció de salt, ha d'haver-hi una nova. Per tant, 'perdrem' dos cicles de rellotge. 1 cicle per afegir la instrucció del salt condicional i 2 per haver de recalcular el salt.

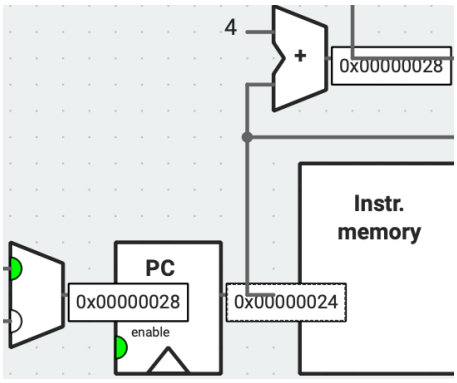
6. **Quan NO es produeix el salt, quantes etapes de la instrucció *auipc x10 0x 10000* s'executen al pipeline?**

Quan NO es produeix el salt, s'executen 5 etapes de la instrucció ***auipc x10 0x 10000*** al pipeline: **IF, ID, EX, MEM, WB**.

auipc x10 0x10000							IF	ID				IF	ID	EX	MEM	WB

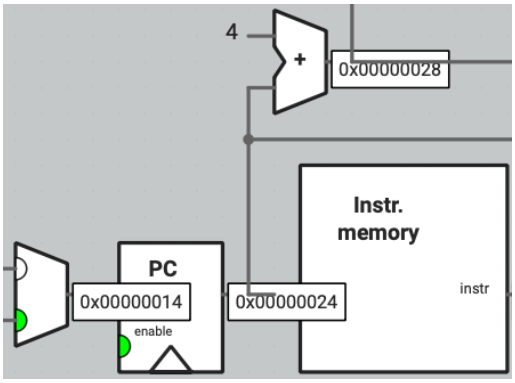
7. **Quan s'està executant la instrucció de salt (etapa EX), però el salt NO es produeix, a quina posició apunta el program counter (PC)?**

Apunta a la posició actual del program counter (**0x00000024**) + 4, és a dir, **0x00000028** (a la següent instrucció):



8. **Quan es produeix el salt, quantes etapes de la instrucció *auipc x10 0x 10000* s'executen al pipeline?**
   Quan es produeix el salt, s'executen 2 etapes de la instrucció ***auipc x10 0x 10000*** al pipeline: **IF, ID**.

9. **Quan s'està executant la instrucció de salt (etapa EX), i el salt es produeix, a quina posició apunta el program counter (PC)?**
   Apunta a la posició on es troba la instrucció ***add x13 x12 x13*** (**0x00000014**), ja que com es produeix el salt, ha de tornar a la posició on comença l'etiqueta del ***loop***.



Conclusions

En aquesta pràctica ens hem familiaritzat amb les diferents microinstruccions de les instruccions que s'utilitzen en els programes. A continuació una llista de la feina realitzada i l'aprenentatge obtingut:

- Hem assolit els coneixements de les microinstruccions de la màquina.
- Hem après el funcionament intern de les microinstruccions.
- Hem après a llegir i entendre un cronograma.
- Hem repassat les instruccions bàsiques de la màquina.
- Hem implementat uns codis senzills ja donats.
- Hem comprovat com canvien les microinstruccions depenent de la naturalesa de la instrucció que tenim.