

Pràctica 2: Exercicis amb el simulador Ripes

Introducció / Objectius

En aquesta pràctica repassarem el conjunt d'instruccions, el funcionament del simulador i l'adreçament a memòria. A continuació l'objectiu principal de la pràctica:

- Familiaritzar-nos amb el funcionament dels registres que hi ha a la CPU. Farem servir el simulador RIPES.

Recordem que com s'ha explicat a teoria, dividim el conjunt de registres fonamentalment en dos tipus:

- **Registres de propòsit general:** es troben al banc de registres. Són 32 registres de propòsit general, el registre x0 sempre val 0, mentre que la resta el seu contingut serà variable. Una bona forma d'inicialitzar un registre (posar a 0 el seu contingut) serà per exemple:
 - ADD a0, zero, zero → Suma el contingut de x0 + el contingut de x0 i es desa en a0.
- **Registre de propòsit específic:** és per exemple el *Program Counter (PC)*, on tenim l'adreça de la següent instrucció a executar. RISC-V té altres registres de propòsit específic (per gestionar el temps, per fer entrada/sortida amb el hardware, etc.) però el simulador Ripes no els implementa.

Exercicis guiats

Exercici 1:

```
1 .data
2 Dades: .word 9, 3, 4, 1 # vector de 4 enters
3 Resultat: .word 0, 0 # vector de 2 enters
4
5 .text
6 main:
7     la a0, Dades # a0 actuarà com a punter
8     lw a1, 0(a0) # 0(a0) = Dades[0]
9     lw a2, 4(a0) # 4(a0) = Dades[1]
10    lw a3, 8(a0) # 8(a0) = Dades[2]
11    lw a4, 12(a0) # 12(a0) = Dades[3]
12
13 loop:
14     add a5, a1, a2
15     add a6, a4, a2
16     addi a3, a3, -1
17     bgez a3, loop
18     sw a5, 16(a0) # 16(a0) = Resultat[0]
19     sw a6, 20(a0) # 20(a0) = Resultat[1]
20
21 end:
22     nop # final del programa
```

Quina operació fa aquest codi?

Si ens fixem en l'estructura del programa, tenim un bucle que es repeteix segons el contingut del registre **a3**. Concretament, quan al restar-li 1 a **a3** ens doni 0, acabarem. El contingut de **a3** és igual a 4, per tant, farem aquest bucle 4 vegades.

El contingut del registre **a5** és el resultat de sumar els valors dels registres **a1** i **a2**. El valor del registre **a5** no canviarà en cap moment de les iteracions, ja que no el modifiquem ni a ell, ni als registres implicats en la suma (**a1**, **a2**).

El contingut del registre **a6** tampoc variarà ja que és el resultat de sumar els valors dels registres **a4** i **a2**, i aquests registres no es veuen modificats a cap iteració que fem del nostre bucle.

Llavors l'operació que fa aquest codi és sumar els valors dels registres **a1** i **a2** i guardar-ho en el registre **a5**; i per altra banda, sumar els valors dels registres **a4** i **a2** i guardar-ho en el registre **a6**.

Hem començat amb el vector de 4 enters **Dades = (9, 3, 4, 1)** i hem obtingut un vector de 2 enters **Resultat = (12, 4)**.

Preguntes a resoldre

1. Un cop acabat el programa quant val el contingut de A5?

El contingut de A5 al acabar el programa dona 12:

x15	a5	12
-----	----	----

2. Un cop acabat el programa quant val el contingut de A6?

El contingut de A6 al acabar el programa dona 4:

x16	a6	4
-----	----	---

3. Un cop acabat el programa quant val el contingut de A3?

El contingut de A3 al acabar el programa dona -1:

x13	a3	-1
-----	----	----

4. Quines instruccions reals s'utilitzen per a les pseudoinstruccions **nop** i **bgez**? (mireu el codi màquina que es genera i consulteu la taula d'instruccions, utilitzant els camps **opcode** i **func3/7**. Si cal)

Codi màquina generat per bgez:

```
0b111111110000001101101101011100011
```

0b111111110000001101101101011100011.

- opcode indicant que es un branch.
- imm[4:1] i imm[11] = -12 (posició PC on es troba la instrucció loop)
- funct3 indicant que es tracta d'un bgez.
- rs1 = 13 = a1
- rs2 = 0 = x0
- imm[10:5] i imm [12]

nop: ***nop*** és una pseudoinstrucció que s'expandeix a `ADDI x0, x0, 0`. El `x0` (o zero) és un registre de només lectura dedicat al valor zero, és a dir, està connectat a zero per a cada bit. Tot el que hi ha escrit en aquest registre simplement es descarta ja que el seu valor no es pot modificar.

Tenint en compte que RISC-V no té indicadors aritmètics (és a dir, indicadors carry, overflow, zero, signes), qualsevol operació aritmètica el registre de destinació de la qual sigui `x0` es farà com a instrucció sense operació independentment dels registres d'origen, ja que el resultat net consistirà d'avançar el comptador del programa a la següent instrucció sense canviar l'estat de cap altre processador rellevant.

5. En quina posició de memòria es troben els valors de Resultat? En quina posició de memòria es guarda el contingut del registre A6?

Els valors de Resultat es troben a les posicions amb adreces `0x10000010` (16) → contingut del registre `a5` i `0x10000014` (20) → contingut del registre `a6`, respectivament:

<code>0x10000014</code>	4	4
<code>0x10000010</code>	12	12

6. Quan executem *bgez*, quin registre es veu afectat?

Al executar la instrucció ***bgez*** (salt condicional) es veu afectat de forma explícita el registre `a3` i implícitament el registre `x0` (que sempre es manté igual a 0). Per tant quan s'executa aquesta instrucció estem comparant el contingut del registre `a3` amb el contingut del registre `x0` (=0), si `a3 >= x0` saltarem condicionalment a ***loop***, sinó seguim amb la instrucció següent.

Exercici 2:

Feu un programa similar al de l'exercici 2. Inicialitzeu primer el contingut de A0. Carregueu al A1 el valor 0000110000001011b. Carregueu al registre A2 el valor 0000000000010001b. Feu l'operació $A0 \leq A0 + A1$ i decrementeu el valor de A2. Feu això en un bucle fins que el valor contingut en A2 sigui 0.

```

1 .data
2 Dades0: .word 0b0000110000001011 #3083
3 Dades1: .word 0b0000000000010001 #17
4
5 .text
6 sub a0, a0, a0
7 la a7, Dades0
8 lw a1, 0(a7) # carreguem el valor de Dades0 al registre a1
9 lw a2, 4(a7) # carreguem el valor de Dades1 al registre a2
10
11 loop:
12 add a0, a0, a1 # A0 <= A0+A1
13 addi a2, a2, -1 # decremtem el valor de A2
14 bgt a2, zero, loop # salta a loop si el registre a2 > zero
15
16 end:
17 nop

```

Preguntes a resoldre:**1. Quina operació matemàtica està realitzant aquest codi?**


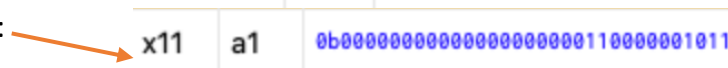
Veiem clarament que es tracta d'una multiplicació ja que estem aplicant directament la definició de multiplicació: sumar n vegades un nombre.

2. Quant val el contingut d'A0, A1 i A2 quan acaba el programa?

A0 (hauríem d'esperar el resultat de la multiplicació de Dades0 (3083) per Dades1(17) que és 52411):

- Decimal: 
- Binari: 

A1 (hauríem d'esperar el mateix valor que a l'inici del programa ja que no l'hem editat en cap moment durant l'execució, és a dir, 3083):

- Decimal: 
- Binari: 

A2 (hauríem d'esperar un valor igual a 0, ja que anem decremantant el seu valor una unitat a cada iteració del bucle **loop**):

- Decimal: 
- Binari: 

3. Quantes iteracions del bucle executa el codi?

El codi executa 17 iteracions del bucle, fins que el contingut del registre **a2** és igual a 0. Això és perquè utilitzem el registre **a2** com a comptador per a aplicar

la definició de multiplicació, en aquest cas sumar 17 cops 3083, per tal d'obtenir el resultat de la multiplicació.

A fer per l'alumne

Exercici 3:

Ens demanen calcular un algorisme que ens faci el següent: Donades dues entrades emmagatzemades en les posicions de memòria A i B fer la comparació. Si $A > B$ calculem la suma. Si $B > A$, fem la diferència (B-A), i si són iguals que multipliqui el seu valor. Carregueu diferents valors en memòria per tal de comprovar el funcionament del programa. Quina instrucció feu servir per fer la multiplicació? Què és més eficient, una instrucció directament que faci aquesta operació o el càlcul iteratiu? Raoneu la resposta.

Codi:

```

1 .data
2 A: .word 4
3 B: .word 4
4
5 .text
6 add a0, zero, zero # inicialitzem el contingut del registre a0
7 la a7, A
8 lw a1, 0(a7) # carreguem el valor de A al registre a1
9 lw a2, 4(a7) # carreguem el valor de B al registre a2
10
11 beq a1, a2, mult # si A = B salta a l'etiqueta mult
12 bgt a1, a2, suma # si A > B salta a l'etiqueta suma
13 blt a1, a2, resta # si B > A salta a l'etiqueta resta
14
15 mult:
16 mul a0, a1, a2 # a0 <= a1·a2
17 j end
18
19 suma:
20 add a0, a1, a2 # a0 <= a1+a2
21 j end
22
23 resta:
24 sub a0, a2, a1 # a0 <= a2-a1
25 j end
26
27 end:
28 nop

```

Cas A > B:

Ho farem amb **A = 10** i **B = 2**, llavors s'hauria de fer la suma (A+B) i hauríem d'obtenir un valor igual a 12 guardat en el registre **a0**.

x10	a0	12
-----	----	----

Cas B > A:

Ho farem amb **A = 2** i **B = 10**, llavors s'hauria de fer la resta (B-A) i hauríem d'obtenir un valor igual a 8 guardat en el registre **a0**.

x10	a0	8
-----	----	---

Cas A = B:

Ho farem amb **A = 4** i **B = 4**, llavors s'hauria de fer la multiplicació (A·B) i hauríem d'obtenir un valor igual a 16 guardat en el registre **a0**.

x10	a0	16
-----	----	----

Quina instrucció feu servir per fer la multiplicació?

Faig ús de la instrucció **mul rd, rs1, rs2**, on **rd** és el registre destí (**a0** en el nostre cas), **rs1** és *source register 1* (**a1** en el nostre cas) i **rs2** és *source register 2* (**a2** en el nostre cas). És a dir, faig ús d'una instrucció incorporada en el simulador Ripes per a dur a terme la multiplicació de les dues variables del programa (A i B).

Què és més eficient, una instrucció directament que faci aquesta operació o el càlcul iteratiu? Raoneu la resposta.

És més eficient utilitzar una instrucció que faci directament aquesta operació. Aquesta afirmació ve donada perquè he testejat el codi de l'exercici 2 on fem la multiplicació implementant la seva definició (càlcul iteratiu) amb valors 13 i 13 (els agafo iguals per poder provar amb els mateixos nombres el codi de l'exercici 3), i això ens hauria de donar 169. L'exercici 2 fa la multiplicació en 45 cicles (**Single Cycle Processor**) i 73 cicles (**5-Stage Processor**), en canvi, l'exercici 3 fa la multiplicació en 9 cicles (**Single Cycle Processor**) i 18 cicles (**5-Stage Processor**), provant així que és més eficient la instrucció **mul rd, rs1, rs2** que fa directament la multiplicació.

- Cicles amb el codi de l'exercici 2:

x10	a0	169
-----	----	-----

Execution info	
Cycles:	45
Instrs.retired:	45
CPI:	1
IPC:	1
Clock rate:	0 Hz

Execution info	
Cycles:	73
Instrs.retired:	45
CPI:	1.62
IPC:	0.616
Clock rate:	0 Hz

- Cicles amb el codi de l'exercici 3:

x10	a0	169
-----	----	-----

Execution info	
Cycles:	9
Instrs.retired:	9
CPI:	1
IPC:	1
Clock rate:	0 Hz

Execution info	
Cycles:	18
Instrs.retired:	9
CPI:	2
IPC:	0.5
Clock rate:	0 Hz

Podem comprovar com la instrucció **mul rd, rs1, rs2** fa la multiplicació de manera molt més eficient.

Preguntes a resoldre:

1. **Expliqueu el funcionament del programa que heu implementat.**

Primerament hem creat dues variables **.word A i B**, amb les quals treballarem i operarem segons la seva relació ($A=B$, $A>B$, $B>A$).

Un cop creades les variables principals i les adreces i registres corresponents, i havent inicialitzat el registre **a0** amb valor **0** (a partir de la instrucció **add a0, zero, zero**) crearem les condicions principals a partir de les quals farem una operació o una altre. La primera condició és **beq a1, a2, mult, beq** vol dir 'Branch if Equal', és a dir, si el valor del registre **a1** (A) és igual al valor del registre **a2** (B) salta condicionalment a l'etiqueta **mult**, on es farà la multiplicació dels valors dels dos registres (**a1, a2**) i es guardarà el resultat al registre **a0** i saltarem incondicionalment amb la instrucció **j end** (a **end** tenim la instrucció **nop**, que vol dir 'No operation').

Si la condició de **a1 = a2** no es compleix, saltem a la següent condició: **bgt a1, a2, suma. bgt** vol dir 'Branch if Greater Than', és a dir, si el valor del registre **a1** (A) és més gran que el valor del registre **a2** (B) salta condicionalment a l'etiqueta **suma**, on es farà la suma dels valors dels dos registres (**a1, a2**) i es guardarà el resultat al registre **a0** i saltarem incondicionalment amb la instrucció **j end**.

Si aquesta última condició no es compleix, saltem a la següent i última condició: **blt a1, a2, resta. blt** vol dir 'Branch if Less Than', és a dir, si el valor del registre **a1** (A) és més petit que el valor del registre **a2** (B) salta condicionalment a l'etiqueta **resta**, on es farà la resta dels valors dels dos registres (**a1, a2**) i es guardarà el resultat al registre **a0** i saltarem incondicionalment amb la instrucció **j end**.

2. **Feu que el resultat es guardi a la posició de memòria 24h bytes després de l'inici de la secció .data.**

Per fer això he creat una nova variable anomenada **Resultat**. Posteriorment he cridat a la instrucció **la a6, Resultat** per utilitzar el registre **a6** com a punter, posteriorment a totes les possibles opcions d'execució del programa (suma, resta o multiplicació) he cridat a la instrucció **sw a0, 28(a6)**, per tal de guardar el resultat a la posició de memòria 24h bytes. Si hagués posat un *offset* de 24, llavors es guardaria a la posició 20h bytes.

A continuació imatge de com queda el programa i un test de tots els possibles cassos:

```

1 .data
2 A: .word 13
3 B: .word 13
4 Resultat: .word 0
5
6 .text
7     add a0, zero, zero
8     la a7, A
9     lw a1, 0(a7)
10    lw a2, 4(a7)
11    la a6, Resultat
12
13    beq a1, a2, mult
14    bgt a1, a2, suma
15    blt a1, a2, resta
16
17 mult:
18     mul a0, a1, a2
19     sw a0, 28(a6)
20     j end
21
22 suma:
23     add a0, a1, a2
24     sw a0, 28(a6)
25     j end
26
27 resta:
28     sub a0, a2, a1
29     sw a0, 28(a6)
30     j end
31
32 end:
33     nop

```

<u>Cas A > B (A=15, B=13, llavors A+B = 28):</u>	0x10000024	28
<u>Cas B > A (B=13, A=1, llavors B-A = 12):</u>	0x10000024	12
<u>Cas A = B (A=B=13, llavors A·B = 169):</u>	0x10000024	169

Conclusions

En aquesta pràctica hem assolit un cert nivell amb algunes de les instruccions de salt, tant condicionals com incondicionals, amb les quals podem generar bucles i així crear algorismes, com per exemple el de la multiplicació. Hem vist també dos formes diferents d'implementar l'algorisme de la multiplicació, de manera iterativa i amb una instrucció que ens proporciona el simulador. A continuació un resum de les tasques realitzades:

- S'han realitzat els exercicis guiats amb l'ajuda del professorat.
- S'ha realitzat el treball a dur a terme per l'alumne.
- S'han repassat les comandes apreses a la pràctica anterior.
- S'ha treballat amb salts condicionals i incondicionals.
- S'ha vist el funcionament de dos bucles senzills.