UNIVERSITAT DE BARCELONA

Pràctica 1: Introducció al simulador Ripes

Alejandro Guzman

18 març 2022

ÍNDEX

1 OBJECTIUS

L'objectiu principal de la primera pràctica de la assignatura és familiaritzar-se amb el simulador **Ripes** i amb els dos processadors amb què treballarem: *Ripes Single Cycle Processor* i *Ripes 5-Stage Processor*. També tindrem una primera toma de contacte amb el format de diverses instruccions i aprendrem quina funció té cadascuna. Per últim, veurem la relació que tenen el llenguatge màquina i l'assemblador.

2 QÜESTIONS

1 Captura del simulador Ripes amb les instruccions modificades

A continuació mostrem primerament les instruccions sense modificar i després les instruccions modificades correctament per tal de que no tinguin errors.

```
# Instruccions sense corregir, podem veure que estàn totes incorrectes excepte una.

LW a1(R0), a1
SW a1,a1(3)
BNE 6(t1)
ADDI t2, #11, 5(t3)
SUB zero ,a2,a3
LOAD 3(a0),a1

# A continuació mostrem totes les instruccions però degudament corregides.

Lw a1, 0(a1)
Sw a1, 3(a1)
BNE t1, t2, etiqueta
ADDI t2, t3, 11
SUB zero ,a2,a3
LW a1, 3(a0)
```

Figura 2.1: Instruccions corregides en el simulador Ripes

A continuació mostrem les instruccions corregides pero ja traduïdes a llenguatge que ho pugui entendre el processador per tal d'executar-les.

```
00000000 <etiqueta>:
    0:      00052583        lw x11 0(x10)
    4:      00b5a1a3        sw x11 3(x11)
    8:      fe731ce3        bne x6 x7 -8 <etiqueta>
    c:      005e0393        addi x7 x28 5
    10:     40d60033        sub x0 x12 x13
    14:     00352583        lw x11 3(x10)
```

Figura 2.2: *Executable mode* de les instruccions corregides

2 Preguntes a), b), c) i d)

2.1 a) En quina posició de memòria escriu aquest programa el resultat?

El resultat d'aquest programa és el contingut del registre a2 i el guardarà en la posició de memòria quatre posicions després de la guardada en el registre a0. I la posició que guarda el registre a0 comença sent xx: .word 3(Amb un valor de 3 en un espai de 32 bits), després el bucle del programa s'executa quatre vegades fins que a la cinquena iteració [a7] == 0 i per tant s'ha executat quatre cops el contingut del bucle on [a0] augmenta en quatre la posició de memòria que guarda, per tant [a2] es guardarà en $4 \cdot 4 + 4 = 20$ posicions de memòria després de la *xx* (la 0x10000000). És a dir, a la 0x10000014.

2.2 b) Què fa el programa?

El programa realitza un bucle on suma les 4 paraules de 32 bits guardades a la memòria de dades. El registre a2 emmagatzema la suma i el a3 es guarda el següent valor a sumar. Finalment es guarda el contingut del registre a2 en la direcció de memòria esmentada en l'apartat a).

2.3 c) Quina sentència de control de flux (en llenguatge d'alt nivell) implementa el programa?

El programa implementa una sentència de control de flux de tipus *while*. Aquesta consisteix en un conjunt d'instruccions que es repeteixen iterativament fins que la condició del *while* no es compleix. En el cas d'aquest programa la condició és que el contingut del registre a7 ha de ser igual a zero, per tant fins a la quarta iteració això no es compleix.
Un cop es compleix al condició de parada del *while*, el programa fa un salt a la instrucció de la etiqueta **final**, donant per a finalitzat el bucle.

2.4 d) Quina és la funció d'a1 al programa? I la d'a0?

La funció d'a1 en el programa serveix com a variable de bucle. És a dir, permet aconseguir quatre iteracions i aturar el bucle en la cinquena. Això és degut a que el contingut del registre a1 comença a 0 i a cada iteració es resta a a7 el registre a1 menys 4 unitats i després es suma una unitat a a1.
D'altra banda, la funció del registre a0 serveix únicament com a variable de posició per guardar l'adreça de les diferents paraules de 32 bits que s'utilitzen per a sumar a a2.

3 Preguntes e), f) i g)

3.1 e) Estabiu la relació entre el codi que hi ha a la finestra 'source code' i el que apareix a la finestra 'Executable code'.

La principal diferència es que en la finestra 'source code' es troba el codi escrit en un llenguatge més senzill per tal de facilitar la programació a l'usuari que està dissenyant el programari. De fet, el codi que apareix en 'Executable mode' és el mateix programa però traduït a instruccions de 32 bits o 8 xifres hexadecimals executables pel processador.

3.2 f) Observeu que inicialment PC=0h (el PC apunta a la següent instrucció executable del programa). Esbrineu com ho fa.

Com cada instrucció és de 32 bits, el PC (**Program Counter**) incrementa el valor de 4 en 4 per apuntar a la següent instrucció (podem veure a l'esquema del processador un sumador amb entrades l'immediat 4 i el PC, mentre que la sortida connecta amb l'entrada del PC).

La primera instrucció a la que apunta no és a la primera programada perquè la primera executada sempre és una operació amb el Program Counter per carregar la primera instrucció des de la memòria de programa.

3.3 g) Executeu el programa, instrucció per instrucció, observant el resultat d'executar cada una de les instruccions

- **auipc x10 0x10000** El PC apunta ara a la primera instrucció del programa.
- **addi x10 x10 0** a0 guarda l'adreça de la memòria de dades *xx*.
- **sub x11 x11 x11** Borrem el contingut del registre a1 (per defecte 0).
- **add x12 x0 x0** Borrem el contingut del registre a2 (per defecte 0).
- **addi x17 x11 -4** Guardem al registre a7 el contingut del registre a1 menys 4.
- **beqx17 x0 24 <final>** No es fa res perquè [a7]no és 0.
- **lw x13 0(x10)** Carreguem al registre a3 el contingut de la posició de la memòria de dades*xx*. [a3] <= 3
- **add x12 x12 x13** El contingut del registre a2 és la suma dels continguts dels registres a2 i a3. [a2] <= [a2] + [a3] = 3.
- **addi x10 x10 4** Augmentem el contingut del registre a0 en 4 unitats (apunta a l'adreça *yy*)
- **addi x11 x11 1** Augmentem el contingut del registre a1 en una unitat (ara és 1)
- **jal x0 0x10 <loop>** Saltem a la posició de memòria *loop*, [PC] <= 0x10
- **addi x17 x11 -4** Guardem al registre a7 el contingut del registre a1 menys 4(ara és -3)
- **beq x17 x0 24 <final>** No es fa res perquè [a7]no és 0
- **lw x13 0(x10)** Carreguem al registre a3 el contingut de la posició de la memòria de dades *yy*. [a3] <= 5
- **add x12 x12 x13** El contingut del registre a2 és la suma dels continguts dels registres a2 i a3. [a2] <= [a2] + [a3] = 3 + 5 = 8
- **addi x10 x10 4** Augmentem el contingut del registre a0 en 4 unitats (apunta a l'adreça *oo*)
- **addi x11 x11 1** Augmentem el contingut del registre a1 en una unitat (ara és 2)
- **jal x0 0x10 <loop>** Saltem a la posició de memòria *loop*, [PC] <= 0x10
- **addi x17 x11 -4** Guardem al registre a7 el contingut del registre a1 menys 4(ara és -2)
- **beq x17 x0 24 <final>** No es fa res perquè [a7]no és 0
- **lw x13 0(x10)** Carreguem al registre a3 el contingut de la posició de la memòria de dades *oo*. [a3] <= 2
- **add x12 x12 x13** El contingut del registre a2 és la suma dels continguts dels registres a2 i a3. [a2] <= [a2] + [a3] = 10 = 0xa
- **addi x10 x10 4** Augmentem el contingut del registre a0 en 4 unitats (apunta a l'adreça *zz*)
- **addi x11 x11 1** Augmentem el contingut del registre a1 en una unitat (ara és 3)
- **jal x0 0x10 <loop>** Saltem a la posició de memòria *loop*, [PC] <= 0x10
- **addi x17 x11 -4** Guardem al registre a7 el contingut del registre a1 menys 4(ara és -1)
- **beq x17 x0 24 <final>** No es fa res perquè [a7]no és 0
- **lw x13 0(x10)** Carreguem al registre a3 el contingut de la posició de la memòria de dades *zz*. [a3] <= 8
- **add x12 x12 x13** El contingut del registre a2 és la suma dels continguts dels registres a2 i a3. [a2] <= [a2] + [a3] = 18 = 0x12extbfaddi x11 x11 1 Augmentem el contingut del registre a1 en una unitat (ara és 1)
- **addi x10 x10 4** Augmentem el contingut del registre a0 en 4 unitats (apunta a l'adreça *zz* +4, 0x10000010)
- **addi x11 x11 1** Augmentem el contingut del registre a1 en una unitat (ara és 4)
- **jal x0 0x10 <loop>** Saltem a la posició de memòria *loop*, [PC] <= 0x10
- **addi x17 x11 -4** Guardem al registre a7 el contingut del registre a1 menys 4(ara és 0)
- **beq x17 x0 24 <final>** Com [a7]és 0, saltem a la posició de memòria *final*, [PC] <= 0x2c

- **sw x12 4(x10)** Guardem el contingut dea2 a +4 posicions de l'adreça guardada a a0, [@0x10000014] <= [a2] = 0x12

4 Preguntes h), i) i j)

4.1 h) Quan triguen a executar el programa? Per què?

En el cas del Single Cycle Risc-V Processor triga un total de 20 cicles en executar el programa, i el 5-Stage Processor triga 33 cicles en completar el mateix programa. Això es degut principalment a que cada instrucció s'executa cinc cops i que potser s'han produït cicles extres en el segon processador degut a la introducció de cicles d'espera per estar treballant amb registres que encara s'han d'actualitzar.
Però això no vol dir que el Single Cycle sigui més ràpid que el 5-Stage Processor, ja que en el Syngle Cycle la duració dels cicles està directament condicionada a la duració de la instrucció més costosa, en canvi en el segon processador, al estar treballant en diferents fases els cicles es poden definir com el màxim del temps més costós en cadascuna de les parts diferenciades del processador (que utilitzen diferents recursos i ens permeten executar parts de 5 instruccions simultàniament). En conclusió, els cicles del 5-Stage són més curts.

4.2 i) Quin és el diagrama d'execució?

El diagrama d'execució del Syngle Cicle Risc-V Processor és el següent:

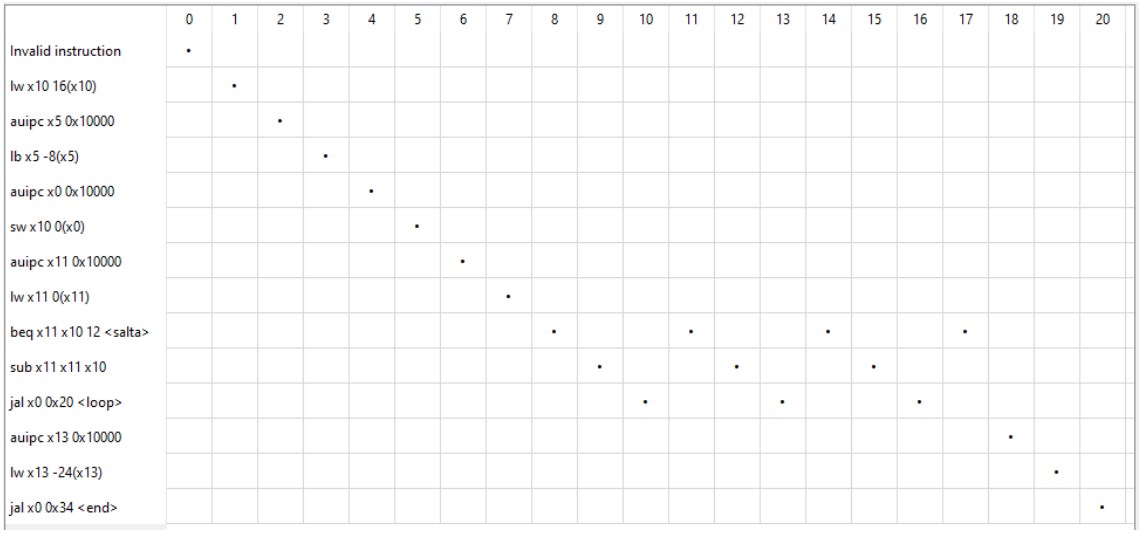

Figura 2.3: Diagrama d'execució del Single Cycle Processor

Com es pot veure, les instruccions s'executen en un sol cicle i en el següent el processador passa a la següent instrucció.
A continuació mostrem el diagrama d'execucií del 5-Stage Risc-V Processor:
Aquí es pot veure que les instruccions triguen diversos cicles en executar-se fins a un màxim de 5, també n'hi han diversos cicles d'espera on no s'utilitza un registre en concret perquè encara no s'ha actualitzat.

4.3 j) Quin és el màxim nombre d'instruccions que s'han executat simultàniament?

En el 5-Stage Risc-V Processor les instruccions s'executen en diversos cicles fins en un màxim de cinc com es pot comprobar en el diagrama d'execució mostrat en l'apartat anterior.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
Invalid instruction	IF	ID	EX	MEM	WB																													
lw x10 16(x10)		IF	ID	EX	MEM	WB																												
auipc x5 0x10000			IF	ID	EX	MEM	WB																											
lb x5 -8(x5)				IF	ID	EX	MEM	WB																										
auipc x0 0x10000					IF	ID	EX	MEM	WB																								
sw x10 0(x0)						IF	ID	EX	MEM	WB																								
auipc x11 0x10000							IF	ID	EX	MEM	WB																						
lw x11 0(x11)								IF	ID	EX	MEM	WB																						
beq x11 x10 12 <salta>									IF	ID	-	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB					
sub x11 x11 x10										IF	-	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID							
jal x0 0x20 <loop>												IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	IF							
auipc x13 0x10000													IF	ID				IF	ID				IF	ID				IF	ID	EX	MEM	WB		
lw x13 -24(x13)														IF					IF					IF					IF	ID	EX	MEM	WB	
jal x0 0x34 <end>																														IF	ID	EX	IF/MEM	ID/WB

Figura 2.4: Diagrama d'execució del 5-Stage Processor

3 Exercicis sessió teoricopràctica

1 Conversió

Transforma en representació de coma flotant el següent valor: 3,14. Un cop trobat el valor, feu el procés invers per recuperar el valor.
Primer de tot, agafem la part entera: 3 = 11b.
Després agafem la part decimal: 0.14 i la convertim a binari.
Ho agrupem tot: 11.00100011101011100001.
Finalment passem a notació científica i agrupem valors.

- Notació científica: $11.00100011101011100001 \cdot 2^1$.
- Valor positiu → Bit de signe = $0 \cdot (-1^0 = 1)6$
- Exponent = 127 + valor exponent → 127 + 1 = 128.
- La mantissa és el valor decimal calculat directament.

Com a resultat final ens queda: 0.10000000110010001111010111100001.
Ara procedim a realitzar el procediment invers.Primerament sabem que el bit més significatiu correspon al bit de signe.
Agrupem valors:

- Exponent: Corresponent a 8 bits = 10000000 = 128 → e = 128 - 127 = 1.
- Mantissa: 11001000111101011100001. $m = 2^{-1}+2^{-2}+2^{-5}+2^{-9}+2^{-10}+2^{-11}+2^{-12}+2^{-14}+2^{-16}+2^{-17}+2^{-18}+2^{-23} = 0.7849999666$

Finalment obtenim el resultat inicial x = 3.139999966

2 Màscares

A continuació es mostra una forma de resoldre l'exercici proposat:

```
xx: word -2147483648
yy: word -123
oo: word 1

.text
lw a0, xx
lw a1, yy
lw a2, oo
and a2, a1, a0
beqz a2, end
cados:
        not a3, a1
        addi a3, a3, 1
```

3 Punters

La instrucció MOV A, M guarda en el registre A el contingut que hi ha a la posició de memòria M. Indiqueu el contingut del registre A en funció dels possibles valors de M.

- $M = 6 \rightarrow [A] = 1234$
- $M = 5 \rightarrow [A] = 65434$
- $M = 4 \rightarrow [A] = 5435$
- $M = 3 \rightarrow [A] = 33223$
- $M = 2 \rightarrow [A] = 67$
- $M = 1 \rightarrow [A] = 23457$
- $M = 0 \rightarrow [A] = 65432$

4 Exercicis finals

Tenim una memòria de 4kBytes. La unitat de memòria és el word(32 bits). Quants bits necessitem per adreçar-la? Analitzeu i determineu la diferència entre adreça i contingut.
4 KBytes són $4 \cdot 2^{10} \cdot 8 = 32000$ bits/32 = 1024 → Necessitem tants bits com per guardar 1024 adreces diferents → Notem que $1000 = 2^1 0$ per tant, amb 10 bits seria suficient.

La instrucció LOAD R1, A carrega el contingut que hi ha a la posició de memòria A en el registre R1. Si la memòria és la de l'exercici 1, quants bits ens calen per identificar el valor de A? Si tenim 8 registres de propòsit general R0, R1, ..., R7. Quants bits calen per identificar el registre R1? Si el conjunt d'instruccions total que té aquest processador és de 100 instruccions, quants bits calen per identificar la instrucció LOAD?
Per identificar A necessitem 10 bits, i n'hi han 100 instruccions LOAD → $2^6 < 100 < 2^7$ → Necessitarà 7 bits com a mínim.
Si hi ha 8 registres neccessitarem 3 bits per guardar les adreces de cada registre ($2^3 = 8$).

4 Conclusions

En aquesta primera pràctica de la assignatura **Introducció als Ordinadors** s'ha començat a programar i testejar amb el simulador **Ripes** i des d'una perspectiva personal ha sigut un molt bon començ de part pràctica de la assignatura, ja que en poques sessions s'ha pogut veure el funcionament amb diversos programes de dos processadors (Single Cycle Risc-V Processor i 5-Stage Risc-V Processor), estudiant les seves característiques i diferències, sobretot en la part de la execució, la velocitat i nombre de cicles.
També s'ha tingut una primera toma de contacte amb el llenguatge ensamblador i el funcionament de diverses instruccions que s'han pogut veure reflexades en les diferents execucions del simulador.
En general ha sigut una pràctica interessant i que prén un bon punt de partida per a les pròximes sessions teoricopràctiques de la assignatura.