UNIVERSITAT DE BARCELONA

Pràctica 3: Operacions i bucles amb RISC-V

Alejandro Guzman

14 Abril 2022

ÍNDEX

1 OBJECTIUS

L'objectiu d'aquesta pràctica és habituar-se a les sentències de control de flux, generació de bucles i salts condicionals i incondicionals programant estructures condicionals i iteratives d'un llenguatge d'alt nivell. També reforçarem allò après en les dues pràctiques anteriors.
Per fer-ho realitzarem dos exercicis guiats, en els quals se'ns introduiran una estructura if i una while i un últim exercici per realitzar independentment en el qual combinarem els coneixements adquirits als primers dos exercicis per implementar un while amb una subsentència condicional dins del bucle.

2 QÜESTIONS

1 Exercici 1

En el primer exercici comencem amb les implementacions de estructures condicionals per tal de realitzar unes accions o unes altres segons la situació.
En aquest exercici se'ns demana implementar un algorisme que calculi la distància entre dos nombres enters a i b (el valor absolut de la seva diferència). Per determinar-la correctament, restarem a – b només si a és major o igual que b; en cas contrari, restarem b – a (ja que si a < b, a – b < 0 i el valor absolut serà – (a – b) = b – a).

El codi escrit en llenguatge d'alt nivell (llenguatge de programació C) proporcionat en el guió de pràctiques que realitza la funció descrita anteriorment és el següent:

```
int a = 5;
int b = -6;
int resultat;

if (a >= b)
 resultat = a - b;
else
 resultat = b - a;
```

Figura 2.1: Implementació de càlcul de distància en valor absolut en llenguatge C

Una possible implementació del mateix algorisme escrit en llenguatge assemblador és el seguent:

```
# Exercici 1

.data
a:   .word 5
b:   .word -6
resultat:    .word 0

.text
    la a0, a
    lw a1, 0(a0)
    lw a2, 4(a0)
    # condició a >= b -> branca certa, a < b -> branca falsa
    blt a1, a2, fals
cert:
    # branca certa
    sub a3, a1, a2
    j end
fals:
    # branca falsa
    sub a3, a2, a1
end:
    sw a3, 8(a0)
```

Figura 2.2: Implementació de càlcul de distància en valor absolut en llenguatge assemblador

El flux normal del programa s'altera si, i només si, a < b, en aquest cas es produeix un salt condicional a la etiqueta fals, on es calcula la diferència b-a, enlloc de a-b, que és la que calcula

l'algorisme en cas de que es segueixi el flux normal de programa i que, per tant, a > b.

A continuació responem les qüestions plantejades en el guió de pràctiques proporcionat.

Quines instruccions de salt condicional hem fet servir? En quin cas salten?
Hem fet servir una única instrucció de salt condicional: **blt a1, a2, fals** (significat en anglès; *branch if less than*). En cas que l'entrada a (guardada a a1) sigui menor que la b (guardada a a2), executa el bloc de codi marcat per fals. L'altra instrucció de salt (*j end* és incondicional.

Què fan les instruccions de salt condicional quan la condició no es compleix?
Quan la instrucció de salt condicional no es satisfà, la instrucció no s'executa i automàticament passa a la següent instrucció escrita en el codi de llenguatge assemblador, en canvi si s'executès produiria, evidentment, un salt condicional i la següent instrucció que s'executaria seria un altre (depenent de com estigui definit el salt).

Per què hem utilitzat les instruccions de salt incondicional?
Hem utilitzar les instruccions de salt incondicional per a mantenir la estructura condicional, ja que si no estigués la única instrucció incondicional (j end) el programa continuaria el fluxe principal i executaria just després de les instruccions de *cert* les de *fals*, la qual cosa seria una contradicció i el programa no funcionaria de la manera que volem.

2 Exercici 2

En el segon exercici de la pràctica implementem el càlcul del terme enèssim de la successió de Fibonacci (entrem un nombre natural n i el programa guarda a memòria justament Fn, l'enèssim terme de la successió) amb una estructura while per tal de produïr un bucle.

El codi proporcionat en llenguatge d'alt nivell (llenguatge C) és el següent:

```
int comptador = 10;
int resultat;

int a = 0;
int b = 1;
while (comptador > 0) {
  int t = a + b;
  a = b;
  b = t;
  comptador--;
}
resultat = a;
```

Figura 2.3: Càlcul del terme n-èssim en C

I si ho traduïm a assemblador: (Figura 2.4)
En la implementació vista anteriorment utiliztem la instrucció *mv* que permet moure ell contingut d'un registre a un altre, però en realitat el que està fent és carregar la suma a un altre més l'inmediat 0.
L'algorisme implementat en llenguatge assemblador calcula els termes de la successió sumant els dos anteriors, tenint en compte que el terme 0-èssim és 0 i el terme 1-èssim és 1. Això ho fem tants cops com indica la entrada de dades etiquetada com *comptador* tal que es va decrementant el contingut del registre a0 en una unitat a cada iteració i quan aquest és zero realitzem un salt condicional al final del programa (comprovant si és zero o més gran en la condició del

```
.data
comptador:   .word 10
resultat:    .word 0

.text
    la a0, comptador
    lw a0, 0(a0)
    addi a1, zero, 0
    addi a2, zero, 1
loop:
    # condició: comptador > 0 -> em quedo en el loop, comptador = 0 -> surto del loop
    beqz a0, end
    # cos del bucle
    add a3, a1, a2
    mv a1, a2
    mv a2, a3
    addi a0, a0, -1
    j loop
end:
    la a0, resultat
    sw a1, 0(a0)
```

Figura 2.4: Càlcul del terme n-èssim en assemblador

bucle).

Quina estructura de control de flux indica normalment un salt cap enrere?
Aquest tipus d'estructura és una estructura *while*, un bucle que executa un bloc de codi repetidament sempre que es satisfaci una determinada condició (en el nostre cas que el contingut del registre a0 sigui no nul i major que zero).

Omple la següent taula executant el programa (modifiquem la 1ª columna).

Valor desitjat	Valor a "resultat"	# cicles	# instruccions
F_0	0	15	9
F_1	1	23	15
F_2	1	31	21
F_3	2	39	27
F_4	3	47	33
F_5	5	55	39
F_6	8	63	45
F_{25}	75025	215	159
F_{46}	1836311903	383	285
F_{47}	-1323752223	391	291

Figura 2.5: Taula de nombre de cicles i instruccions

Cal destacar que el nombre de cicles i instruccions es determinen usant el R5SP ja que, usant el processador d'un sol cicle tindríem el mateix nombre de cicles que d'instruccions. Cal destacar, també, que els cicles i les instruccions que porta el programa des de l'inici es poden trobar a la finestra del processador.

Que pasa amb el resultat F_{47}?
Es pot comprovar que el resultat obtingut és un nombre negatiu, resultat el qual no és l'esperat, perquè el 47-èssim terme de la successió de Fibonacci és la suma dels dos termes anteriors (com tota la successió), és a dir, $F_{47} = F_{45} + F_{46} = 1.134.903.170 + 1.836.311.903 = 2.971.215.073$. En aquest cas ens trobem amb un *overflow* ja que el contignut dels registres és de 32 bits i sabent que els nombres es guarden amb signe podem guardar als registres aquells nombres enters tals

siguin majors o iguals a -2^{31} i menors o iguals a $2^{31}-1$. Com el màxim nombre positiu que podem representar és el $2^{31}-1 = 2.147.483.647$ i $F_{47} < 2^{31}-1$, es suma l'excedent començant per -2^{31}. Per això s'obté un valor negatiu.

3 Exercici 3

Descriu el programa que has implementat. Quins salts has usat? Quins són condicionals i quins són incondicionals, i per què?

En el tercer i últim exercici d'aquesta pràctica hem d'implementar l'algorisme d'Euclides en llenguatge assemblador, el qual determina el mcd (màxim comú divisor) de dos nombres naturals.
Per programar l'algorisme emprarem una estructura *while if* (bucle on es realitza internament una acció o una altra depenent dels valors dels registres).
Donats a i b, la condició de bucle és comprovar que són diferents, llavors si a > b, assignem a **a** el valor de la diferència a - b; en cas contrari assignem a **b** el valor de la diferència b - a. Al finalitzar el bucle el mcd dels dos nombres serà el valor de la variable **a**.

La implementació de l'algorisme d'Euclides en llenguatge d'alt nivell (C) és la següent:

```
int a = 252;
int b = 105;
int resultat;

while (a != b) {
  if (a > b)
    a = a - b;
  else
    b = b - a;
}
resultat = a;
```

Figura 2.6: Algorisme d'Euclides en C

Una de les possibles implementacions de l'algorisme d'Euclides en llenguatge assemblador és la següent (Figura 2.7)
Com es pot comprovar en el codi mostrat anteriorment, **a** es guarda al registre a1, i **b** al registre a2. Si el contingut del primer és més petit o igual que el del segon (**ble a1, a2, fals**, i.e. a <= b), saltem a la branca falsa; en cas contrari (a > b), saltem a la branca certa. Un cop finalitza l'execució del bucle, guardem a memòria el contingut del registre a1, que guarda el valor del mcd dels dos nombres entrats.

Hem utilitzat dos salts condicionals i dos d'incondicionals. Els condicionals han estat per comprovar si els registres compleixen la condició per iterar un altre cop al bucle i per executar la branca falsa de dins del bucle (b = b – a) en cas de que la condició del *if* no es compleixi. Els salts incondicionals s'han programat amb la instrucció *j loop* i serveixen per saltar a la comprovació del bucle després d'executar qualsevol de les dues branques.

Finalment el programa retorna el mcd dels dos valors descrit al guió de pràctiques assumint d'aquesta manera que l'algorisme s'ha implementat correctament.

```
.data
a:   .word 252
b:   .word 105
resultat:    .word 0

.text
    la a0, a
    lw a1, 0(a0)  # [a1] = [@a]
    lw a2, 4(a0)  # [a2] = [@b]
loop:
    # condició while: a != b, saltem si a == b
    beq a1, a2, end
    # condició if: a > b, saltem a fals si a <= b
    ble a1, a2, fals
cert:
    # branca certa: a = a - b
    sub a1, a1, a2
    j loop
fals:
    # branca falsa: b = b - a
    sub a2, a2, a1
    j loop
end:
    sw a1, 8(a0)
```

Figura 2.7: Algorisme d'Euclides en assemblador

3 CONCLUSIONS

Aquesta pràctica ha servit per adentrar-nos més en el simulador *Ripes*, en el llenguatge assemblador i en el proceessador *Risc-V*, també hem tingut una primera toma de contacte amb les estructures iteratives i amb salts condicionals i no condicionals, programant i dissenyant diversos algorisme que empren aquestes eines per resoldre els problemes que plantejen.

Ha sigut de molta utilitat el com estaba plantejada la pràctica, ja que el primer exercici era amb condicionals, el segon utilitzant un bucle *while* i el tercer combinava aquestes dues estructures per resoldre el problema plantejat, de manera que eren tres exercicis acumulatius per tal de facilitar l'aprenentatge en relació als temes descrits en aquesta pràctica.