Pràctica 2

Objectius

L'objectiu d'aquesta pràctica és familiaritzar-nos amb el funcionament dels registres que hi ha a la CPU.

A més de veure estructures condicionals i analitzar la operació de multiplicar per veure de quina manera es pot fer més eficient.

Explicació de la practica

Aquesta segona practica consisteix en fer 3 exercicis i respondre a les preguntes plantejades a més dels 2 exercicis del pdf Teoricopràctic 2 on s'havien de fer dos programes en el Ripes.

Informe:

Exercici 1:

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

El contingut de A5 un cop acabat el programa és 12.

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

El contingut de A6 un cop acabat el programa és 4.

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

El contingut de A3 un cop acabat el programa és -1.

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)

La instrucció real per bgez és: si n >= 0 el flux del programa saltarà a la etiqueta que hi hagi associada.

La instrucció real per nop és: No operació, es a dir, que el programa no realitzarà cap instrucció.

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 en les posicions de memòria 0x10000010 i 0x10000014.

El contingut del registre A6 es guarda en la posició de memòria 0x10000014.

Quan executem bgez, quin registre es veu afectat?

El registre que es veu afectat quan executem bgez és el PC ja que aquest ha de apuntar a la següent instrucció, per tant depenent de si es compleix o no la condició del bgez el PC haurà d'apuntar a la següent línia on s'executarà sw a5, 16(a0), o tres línies cap endarrere per executar add a5,a1,a2.

Exercici 2:

```
1  .data
2  A: .word 3083            #Vector de 1 enter
3  B: .word 17              #Vector de 1 enter
4
5  .text
6      la a0, A                #a0 atua com un punter i apuntara a la direccio de A
7      lw a1, 0(a0)            #Es carrega el contingut al que apunta a0 en a1. a1 = 3082
8      lw a2, 4(a0)            #Es carrega el contingut al que apunta a0 desplaçat 4 bytes en a2. a2 = 17
9      sub a0, a0, a0          #S'inicialitza a0 = 0
10
11 loop:
12     beqz a2, end            #Si a2 es igual a 0 es salta a end
13     add a0, a0, a1          #Es suma a0 mes a1 i es guarda el resultat en a0
14     addi a2, a2 -1          #a2--
15     j loop
16
17 end:
18     nop
```

Quina operació matemàtica està realitzant aquest codi?

La operació matemàtica que s'està realitzant es una multiplicació, en el loop es fa 3083 x 17.

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

A0 = 52411

A1 = 3083

A2 = 0

Quantes iteracions del bucle executa el codi?

Com que a2 = 17 i cada vegada que es realitza el bucle decrementa una unitat, s'executaran 17 iteracions.

Exercici 3:

Ens demanen calcular un algoritme 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.

Per la multiplicació és més eficient fer la instrucció mul ja que només fa un cicle per executar-la. En canvi fer el càlcul iteratiu trigarà molt més ja que ha de realitzar les instruccions que hi han dins del bucle tantes vegades com aquest es repeteixi.

Codi amb la instrucció mul per fer la multiplicació

```
.data
A: .word 2          #Vector de 1 enter
B: .word 2          #Vector de 1 enter

.text
    la a0, A                #a0 atua com un punter i guardara el contingut de A
    lw a1, 0(a0)            #Es carrega el contingut al que apunta a0 en a1. a1 = 5
    lw a2, 4(a0)            #Es carrega el contingut al que apunta a0 desplaçat 4 bytes. a2 = 3
    bgt a1, a2, cas1        #Si a1 > a2 el programa salta al cas1
    bgt a2, a1, cas2        #Si a2 > a1 el programa salta al cas2
    beq a1, a2, cas3        #Si a1 == a2 el programa salta al cas3

cas1:
    add a3, a1, a2          #Es suma a1 mes a2 i es guarda el resultat en a3
    sw a3, 36(a0)           #Es guarda el contingut de a3 en la adreça on apunta a0 desplaçat 8 bytes
    j end

cas2:
    sub a3, a2, a1          #Es resta a2 menys a1 i es guarda el resultat en a3
    sw a3, 36(a0)           #Es guarda el contingut de a3 en la adreça on apunta a0 desplaçat 8 bytes
    j end

cas3:
    mul a3, a1, a2          #Es multiplica a1 per a2 i es guarda el resultat en a3
    sw a3, 36(a0)           #Es guarda el contingut de a3 en la adreça on apunta a0 desplaçat 8 bytes
    j end

end: nop
```

Codi amb el bucle per fer la multiplicació

```
.data
A: .word 2          #Vector de 1 enter
B: .word 2          #Vector de 1 enter

.text
    la a0, A                #a0 atua com un punter i guardara el contingut de A
    lw a1, 0(a0)            #Es carrega el contingut al que apunta a0 en a1. a1 = 5
    lw a2, 4(a0)            #Es carrega el contingut al que apunta a0 desplaçat 4 bytes. a2 = 3
    bgt a1, a2, cas1        #Si a1 > a2 el programa salta al cas1
    bgt a2, a1, cas2        #Si a2 > a1 el programa salta al cas2
    beq a1, a2, cas3        #Si a1 == a2 el programa salta al cas3

cas1:
    add a3, a1, a2          #Es suma a1 mes a2 i es guarda el resultat en a3
    sw a3, 36(a0)           #Es guarda el contingut de a3 en la adreça on apunta a0 desplaçat 8 bytes
    j end

cas2:
    sub a3, a2, a1          #Es resta a2 menys a1 i es guarda el resultat en a3
    sw a3, 36(a0)           #Es guarda el contingut de a3 en la adreça on apunta a0 desplaçat 8 bytes
    j end

cas3:
    beqz a2, end            #Si a2 es igual a 0, es salta al end
    add a3, a3, a1          #Es fa la suma de a3 mes a1 i es guarda el resultat en a3
    addi a2, a2, -1         #a2--
    j cas3

end:
sw a3, 36(a0)           #Es guarda el contingut de a3 en la adreça on apunta a0 desplaçat 8 bytes
nop
```

Expliqueu el funcionament del programa que heu implementat.

El programa comença amb 2 valors a memòria que es carregaran en a1 i a2, després es farà una estructura condicional per comparar els 2 valors. Si a1 > a2, es farà la seva suma, si a2 > a1, es farà la seva resta, i si a2 = a1, es farà la seva multiplicació. El resultat de l'operació realitzada es guardarà en a3 i també en memòria.

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

Veure en les captures anteriors.

Exercicis addicionals

Teòrico-Pràctica 2:

1. Feu un programa en assemblador que carregui dos valors de una determinada posició de memòria en dos registres: a1 i a0, faci la suma i ho guardi en a2. Finalment guarda el resultat en memòria.

```
.data
xx: .word 10              #Etiqueta que guarda 32 bits per emmgatzmar una paraula que conte un 10
yy: .word 20              #Etiqueta que guarda 32 bits per emmgatzmar una paraula que conte un 20
resultat: .word 0         #Etiqueta que guarda 32 bits per emmgatzmar una paraula que conte un 0

.text
    la a3, xx             #a3 fa de punter que apunta a la direccio de memoria de xx que conte un 10
    lw a0, 0(a3)          #Es carrega en a0 el contingut al qual apunta a3. a0 = 10
    lw a1, 4(a3)          #Es carrega en a1 el contingut al qual apunta a3 desplaçat 4 bytes. a0 = 20
    add a2, a0, a1        #Es suma a0 mes a1 i es guarda en a2. a2 = 30
    sw a2, 8(a3)          #Es carrega el contingut de a2 en la posicio de memoria que apunta a3 desplaçada 8 bytes
```

2. Feu un programa en assemblador que carregui dos valors de la memòria en dos registres. Si els valors són iguals, que faci el producte, si un és negatiu que faci la suma, si els dos són positius que faci la resta i si tots dos són negatius, que els passi a positiu i els guardi en memòria.

```
.data
A: .word -2, -3           # Vector de 2 enters

.text
    la a0, A              #a0 actuara com a punter
    lw a1, 0(a0)          #Es carrega el contingut al que apunta a0 en a1
    lw a2, 4(a0)          #Es carrega el contingut al que apunta a0 desplaçat 4 bytes en a2

    beq a1, a2, cas1      #Si a1 es igual a a2 es salta al cas1
    blez a1, aux1         #Si a1 es negatiu es salta a aux1
    blez a2, aux2         #Si a2 es negatiu es salta a aux2

cas3:
    sub a3, a1, a2        #Es resta a1 menys a2 i es guarda el resultat en a3
    j end
cas1:
    mul a3, a1, a2        #Es multiplica a1 per a2 i es guarda el resultat en a3
    j end

cas2:
    add a3, a1, a2        ##Es suma a1 mes a2 i es guarda el resultat en a3
    j end

cas4:
    not a1, a1            #Es nega a1 i es guarda en a1
    not a2, a2            #Es nega a2 i es guarda en a2
    addi a1, a1, 1        #Es suma 1 a a1 i es guarda en a1
    addi a2, a2, 1        #Es suma 1 a a2 i es guarda en a2
    sw a1, 8(a0)          #Es guarda el contingut de a1 en memmoria
    sw a2, 12(a0)         #Es guarda el contingut de a2 en memoria
    j end

aux1:
    blez a2, cas4         #Si a2 es negatiu salta al cas4 ja que a1 i a2 son negatius
    j cas2
aux2:
    blez a1, cas4         #Si a1 es negatiu salta al cas4 ja que a1 i a2 son negatius
    j cas2
end:
nop
```

Conclusions

Amb aquesta practica m'ha quedat més clar com funciona el Ripes ja que havíem de realitzar diversos programes.

Encara que m'ha costat una mica, he acabat entenent les estructures condicionals amb els salts a diferents etiquetes per fer les operacions corresponents en funció de la condició imposada.

Nacho Rivera

Grup B