taller 03 arquitectura

Ejercicios 2

```
li a0, 4228
li a1, 2114
jal ra, resta

fin: beq zero, zero, fin
resta:
prologo: addi sp, sp, -4
         sw ra, 0(sp)
sub a0, a0, a1
beq a0, zero, epilogo
sigo: jal ra, resta
epilogo: lw ra, 0(sp)
         addi sp, sp, 4
         ret
```

Observaciones

1. la pila son los distintos valores de **ra** guardados en la adress **sp**.
2. **ra** se **modifica** y se pierde, por eso se guardan en la pila sus diferentes valores.

- **ra** guarda una dirreccion de memoria
- **sp** es una direccion de memoria
- **ret** vuelve al ra_actual

Como Funciona ?

main

- asigna **a0** y **a1** constantes
- guarda **ra_inicial(fin)**, ignora **fin** y ejecuta **resta**
- pasa a resta

resta

prologo

- escribe memoria y apila (reserva espacio):
  - guarda **ra** en memoria
  - pasa a body

body:

- resta `a0 - a1` y vuelve a restar hasta que `a0 = 0`
- cada vez que se ejecuta resta se modifica **ra** y vuelve a **resta**
- pasa a **epilogo**

epilogo:

- escribe en registro (consulta a memoria) y desapila (libera espacio):
  - sobre-escribe el registro **ra** por los valores en memoria (pila) y vuelve al **ra_actual**
  - "se busca volver a ra_inicial."
  - pasa a **fin**

fin :

- se cumple condicion y termina

Pila y SP

si suponemos que sp esta en `0x00` podemos ver su comportamiento
ra_i representa los distintos estados de **ra**

prologo: escribo en memoria

sp	sp op	apila	estado pila
0x08	sp		$[\ ]$
0x04	sp-4	ra_0	$[ra_0]$
0x00	sp-4	ra_1	$[ra_0, ra_1]$

epilogo : consulto memoria(escribo registro)

sp	sp op	desapila	estado pila
0x00	sp+4	ra_1	$[ra_0, ra_1]$
0x04	sp+4	ra_0	$[ra_0]$
0x08	sp		$[\ ]$

Ejercicio 3

Ejercicio 3A.

Solo Sabemos que hay en las adress donde estan cargadas las instrucciones:
`0x08, 0x0c, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30, 0x34, 0x38`
las demas valen 0 por enunciado. `0x00, 0x04`

mem[x] es cuando se accede a memoria en direccion x

```
main:
addi x11 x0 4              #        x11 = 4
lw x12 0 x11               #        x12 = 0 <--- x12 = mem[4]  <-- 0
addi x13 x0 4              #        x13 = 4
lw x13 0 x13               #        x13 = 0 <--- x13 = mem[4]  <-- 0
lw x13 0 x13               #        x13 = 0 <--- x13 = mem[0]  <-- 0
                           #
beq x12 x13 -20            #        como x12 == x13, siempre vuelve a main
                           #
guardar:                   #
lui x14 0xfffa6            #
addi x14 x14 -1539         #
add x12 x14 x12            #
sw x11 40 x12              #
fin_programa:              #
addi x10 x0 0              #
addi x17 x0 93             #
ecall
```

Seguimiento

```
00000000 <root>: al agregar esto no se afecta el Ejercicio (para ver en ripes).

    00:          ????????          sw zero 0 zero
    04:          ????????          sw zero 4 zero

00000008 <main>:
    08:          00400593          addi x11 x0 4                  x11 = 0x04
    0c:          0005a603          lw x12 0 x11                   x12 = 0x00
    10:          00400693          addi x13 x0 4                  x13 = 0x04
    14:          0006a683          lw x13 0 x13                   x13 = 0x00
    18:          0006a683          lw x13 0 x13                   x13 = 0x00
    1c:          fed606e3          beq x12 x13 -20 <main>

00000018 <guardar>:
    20:          fffa6737          lui x14 0xfffa6
    24:          9fd70713          addi x14 x14 -1539
    28:          00c70633          add x12 x14 x12
    2c:          02b62423          sw x11 40 x12

00000028 <fin_programa>:
    30:          00000513          addi x10 x0 0
    34:          05d00893          addi x17 x0 93
    38:          00000073          ecall

 # 12 ciclos del PC

x11 = 0x04 0x04
x12 = 0x00 0x00
x13 = 0x04 0x00 0x04 0x00

PC = 0x08, 0x0c, 0x10, 0x14, 0x18, 0x1c, 0x08, 0x0c, 0x10, 0x14, 0x18, 0x1c
```

Observaciones :

el programa consulta a memoria fuera de rango `0x00, 0x04` que vale 0 por enunciado
haciendo que `x13 = x12 = 0` , luego nunca pasa de la instruccion `beq x12 x13 -20` ,
que vuelve a main en un bucle sin salida.

Ejercicio 3B

Solo Sabemos que hay en las adress donde estan cargadas las instrucciones:
`0x00, 0x04, 0x08, 0x0c, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x2c, 0x30`
las demas valen 0 por enunciado. `0x40, 0x0005a603`

- mem[x] es cuando se accede a memoria en direccion x

```
main:
addi x11 x0 4              #        x11 = 4
lw x12 0 x11               #        x12 = mem[4]  -->  x12 = lw x12 0 x11  = 0x0005a603 (ir)
addi x13 x0 4              #        x13 = 4
lw x13 0 x13               #        x13 = mem[4]  -->  x13 = lw x12 0 x11  = 0x0005a603 (ir)
lw x13 0 x13               #        x13 = 0       <-- x13 = mem[0x0005a603]   <-- 0
                           #
beq x12 x13 -20            #        como x12 != x13, no vuelve a main
                           #
guardar:                   #
lui x14 0xfffa6            #        x14 = 0xfffa6000 --> carga los ultimos 20 bits
addi x14 x14 -1539         #        x14 = 0xfffa6000 - 0x7FD00000 =  0xfffa59fd  ---> carga los ultimos 20bits
add x12 x14 x12            #        x12 = 0xfffa59fd + 0x0005a603 = 0x00000000 ?
sw x11 40 x12              #        mem[40+0] = 0
fin_programa:              #
addi x10 x0 0              #        x10 = 0
addi x17 x0 93             #        x17 = 93
ecall
```

Seguimiento

```
00000000 <main>:
    0:           00400593          addi x11 x0 4                  x11 = 0x00000004
    4:           0005a603          lw x12 0 x11                   x12 = 0x0005a603
    8:           00400693          addi x13 x0 4                  x13 = 0x00000004
    c:           0006a683          lw x13 0 x13                   x13 = 0x0005a603
    10:          0006a683          lw x13 0 x13                   x13 = 0x00000000
    14:          fed606e3          beq x12 x13 -20 <main>

00000018 <guardar>:
    18:          fffa6737          lui x14 0xfffa6                x14 = 0xfffa6000
    1c:          9fd70713          addi x14 x14 -1539             x14 = 0xfffa59fd
    20:          00c70633          add x12 x14 x12                x12 = 0x00000000
    24:          02b62423          sw x11 40 x12                  x11 = 0x00000000

00000028 <fin_programa>:
    28:          00000513          addi x10 x0 0                  x10 = 0x00000000
    2c:          05d00893          addi x17 x0 93                 x17 = 0x0000005d    pc => 00400593 => 05d00893


x11 = 0x04 0x04
x10 = 0x00
x12 = 0x5a603 0x00
x13 = 0x04 0x5a603 0x00
x14 = 0xfffa6000 0xfffa59fd
x17 = 0x5d


PC = 0x00, 0x04, 0x08, 0x0c, 0x10, 0x14, 0x18, 0x1c, 0x20, 0x24, 0x28, 0x00, 0x2c, 0x30
```

¿Cambia la ejecución del programa? ¿De qué manera? ¿Por qué?

- Ahora Sabemos que hay en la adress 4, esta cargada la instruccion `lw x12 0 x11`
  → mem[4] = `lw x12 0 x11`
- a `x12` se le asigna `4` , luego en `x12` se carga la adress `x12` , osea `x12 = mem[x12] = mem[4] = "lw x12 0 x11"`
  → x12 = `lw x12 0 x11`
- a x13 se le asigna 4, luego en x13 se carga la adress x13, osea x13 = mem[x13] = mem[4] = "lw x12 0 x11"
  → x13 = `lw x12 0 x11`
- luego se vuelve a cargar en x13 la adress x13, osea `x13 = mem[x13] = mem[lw x12 0 x11] = 0` , 0 porque esta fuera de nuestro rango.
  → x13 = 0
- AHORA se ejecuta `beq x12 x13 -20` , la cual no se cumple la condicion pues `x13 != x12` , y pasa al resto del programa

....