Informe de la práctica 3

Introducción

En esta 3ª práctica de Introducció als Ordinadors encontraremos los objetivos, los ejercicios planteados y conclusiones:

- *Utilización de las sentencias de control de flujo.*
- *Generación de bucles y saltos condicionales e incondicionales.*
- *Solidificar los conocimientos adquiridos en las prácticas anteriores.*

Problema 1

Los saltos nos permiten ejecutar diferentes bloques de código y así podemos implementar estructuras de control de flujo.

Ejemplo if: distancia entre dos números enteros.

Para calcular la distancia entre dos números enteros, haremos la resta y entonces el valor absoluto. $|a - b|$

Ejemplos: $|5 - (-6)| = 11$ $|-6 - 5| = 11$

El programa siguiente en C calcula esta distancia. ¿Cómo se haría en RISC V?

Nota important: Fixa't que la condició per al codi d'alt nivell ($a \geq b$) és la contrària que fem servir en el codi en llenguatge assemblador. Això passa perquè en el codi d'alt nivell la condició indica que s'executa la branca certa ($a \geq b$). En canvi, en el codi màquina de l'exemple anterior indica que saltem a la branca falsa ($a < b$). També és important recordar que després d'executar la branca certa cal saltar-se el bloc de la branca falsa.

Alto nivel (C)	RISC V

int a = 5;
int b = -6;
int resultat;
 	
if (a >= b)
 	
resultat = a - b;
 	
else
 	
resultat = b - a;
	.data
	a: .word 5
	b: .word -6
	resultat: .word 0
	
	.text
	
	la a0, a
	
	lw a1, 0(a0)
	
	lw a2, 4(a0)
	
	# condición
	
	cert:
	
	# rama certa
	
	fals:
	
	# rama falsa
	
	end:
	
	sw a3, 8(a0)

Código en Ripes:

```
.data
a: .word 5                # establecemos los valores de los que queremos medir la distancia
b: .word -6
resultat: .word 0         # y donde guardaremos el valor resultante
.text
        la a0, a          # cargamos la dirección de a en a0
        lw a1, 0(a0)      # cargamos los valores de a y b en los registros a1 y a2 respectivamente
        lw a2, 4(a0)
        blt a1, a2, fals  # si el contenido de a1 es menos que el de a2, saltaremos a false; sino a cert
cert:
        sub a3, a1, a2    # restamos a3 => a1 – a2 y hacemos un salto incondicional a end
        j end
```

```
fals:
        sub a3, a2, a1      # restamos a3 => a2 – a1 y hacemos un salto incondicional a end
end:
        sw a3, 8(a0)        # guardamos el contenido de a3 en la dirección de memoria 8(x10)
```

Informe del ejercicio 1

a) ¿Qué instrucciones de salto condicional usamos? ¿En qué caso saltamos?

Hemos usado la instrucción BLT (blt a1, a2, fals) para saltar a la condición “fals” del ejercicio si el contenido de a1 es menor que el de a2. También podríamos haber usado la instrucción BGT para ir a la condición “cert”, pero, si no se cumple la condición de salto de BLT, el código avanza directamente hasta ahí.

b) ¿Qué hace las instrucciones de salto condicional cuando la condición no se cumple?

Si la condición de salto de una instrucción no se cumple, no realizará el salto y pasará a la siguiente línea de código. En este caso, si BLT no se cumple, entraremos en “cert” y se hará la resta del SUB.

c) ¿Por qué hemos utilizado las instrucciones de salto incondicional?

Cada condición if debe tener un final, sino la máquina seguiría leyendo cada instrucción línea por línea. En el “fals” no hace falta un salto incondicional, porque directamente la siguiente instrucción a leer es el final del programa (end).

Problema 2

Ejemplo while: Fibonacci

La succesión de Fibonacci es una succesión matemática de números naturales tal que cada uno de ellos, sus términos son iguales a la suma de los dos anteriores.

Fn = Fn−1 + Fn−2

F0 = 0, F1 = 1

Ejemplos:

F2 = 1

F3 = 2

F4 = 3

...

F9 = 34

F10 = 55

El programa siguiente calcula el término de la serie de Fibonacci indicado por la variable “comptador” (Fcomptador), y lo guarda a la variable “resultat”. ¿Cómo se haría en RISC V?

Alto nivel (C)	RISC V

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;
	.data
	comptador: .word 10
	resultat: .word 0
	
	.text
	
	la a0, comptador
	
	lw a0, 0(a0)
	
	addi a1, zero, 0
	
	addi a2, zero, 1
	
	loop:
	
	# condición
	
	# cuerpo del bucle
	
	end:
	
	la a0, resultat
	
	sw a3, 8(a0)

Código en RISC V:

```
.data
comptador: .word 10             # establecemos el valor del contador
resultat: .word 0               # y donde guardaremos el resultado

.text
        la a0, comptador        # cargamos la dirección de contador en a0
        lw a0, 0(a0)            # cargamos el valor del contador en 0(x10)
        addi a1, zero, 0        # inicializamos el resgistro a1 a 0
        addi a2, zero, 1        # inicializamos el resgistro a2 a 1

loop:
        beqz a0, end            # mientras el contador no haya llegado a 0, hacemos el loop
        add a3, a1, a2          # sumamos a3 => a1 + a2
        mv a1, a2               # a1 ahora contiene el valor de a2
        mv a2, a3               # a2 ahora contiene el valor de a3
        addi a0, a0, -1         # restamos a0 => a0 - 1
        j loop                  # salto incondicional a loop

end:
        la a0, resultat         # cargamos la dirección de resultat en a0
        sw a1, 0(a0)            # guardamos el resultado de a1 en 0(x10)
```

Informe del ejercicio 2

a) ¿Qué tipo de estructura de control de flujo indica normalmente un salto hacia atrás ?

Un salto incondicional, como el j loop del código. Hace que repita el apartado loop mientras no se cumpla la condición del beqz.

b) Rellena la siguiente tabla ejecutando el programa.

Valor inicial	Valor a «resultat»	# ciclos	# instrucciones
F0	0	15	9
F1	1	23	15
F2	1	31	21
F3	2	39	27
F4	3	47	33
F5	5	55	39
F6	8	63	45
F25	75025	215	159
F46	1836311903	383	285
F47	-1323752223	391	291

c) ¿Quá pasa con el resultado F47?

No es la solución correcta. La máquina Ripes guarda como resultado -1323752223 pero el resultado real es 2971215073. Esto se debe a que el resultado de F47 en binario alcanza los 32 bits, por lo tanto usa el bit de signo par intentar representar el número (de ahí el negativo). Para comprenderlo mejor podemos observar la siguiente tabla:

Fx	Valor decimal	Valor binario (32 bits gracias al .word)
F46	1836311903	0110 1101 0111 0011 1110 0101 0101 1111 (positivo)
F47	2971215073	1011 0001 0001 1001 0010 0100 1110 0001 (negativo)

Problema 3

Para hacer en casa: implementa las partes que falten del programa ensamblador. Ejemplo while con if: máximo común divisor

El máximo común divisor (mcd) de dos o más números enteros positivos es el mayor divisor posible de todos ellos: mcd (a, b).

Ejemplos: mcd (252, 105) = 21 ; mcd (13, 31) = 1

A continuación tenemos la implementación en C usando el algorismo de Euclides. Dentro de un bucle “while”, con cada iteración se hace una comparación en un “if”. ¿Cómo se haría en RISC V?

Tingues en compte el flux del programa assemblador, i el que hem explicat en els exercicis anteriors sobre com avaluem de forma diferent les condicions en llenguatge d’alt nivell i en llenguatge màquina, i que algunes estructures de flux necessiten també salts incondicionals.

Alto nivel (C)	RISC V

int a = 5;
int b = -6;
int resultat;
 	
while (a != b){
 	
if (a > b)
 	
resultat = a – b;
 	
else
 	
b = b – a;
 	
}
 	
resultat = b - a;
	.data
	a: .word 252
	b: .word 105
	resultat: .word 0
	
	.text
	
	la a0, a
	
	lw a1, 0(a0)
	
	lw a2, 4(a0)
	
	loop:
	
	# condición while
	
	# condición if
	
	cert:
	
	# rama certa
	
	fals:
	
	# rama falsa
	
	end:
	
	sw a1, 8(a0)

Código en Ripes:

```
.data
a: .word 252                    # establecemos los valores de los que queremos ver el mcd
b: .word 105
resultat: .word 0               # y donde guardaremos el resultado
.text
        la a0, a                # cargamos la dirección de a en a0
        lw a1, 0(a0)            # y los valores de a y b en a1 y a2 respectivamente
        lw a2, 4(a0)
loop:
        beq a1, a2, end         # si a1 y a2 son iguales, se va al final del programa
        bgt a1, a2, cert        # si a1 es mayor que a2, pasamos al apartado «cert»
        blt a1, a2, fals        # si a1 es menor que a2, pasamos al apartado «fals»

cert:
        sub a1, a1, a2          # restamos a1 => a1 - a2
        j loop                  # salto incondicional al apartado «loop»
fals:
        sub a2, a2, a1          # restamos a2 => a2 - a1
        j loop                  # salto incondicional al apartado «loop»
end:
        sw a1, 8(a0)            # guardamos el resultado de a1 en 8(x10)
```

Informe ejercicio 3

Describe el programa que has implementado. ¿Qué saltos has usado? ¿Cuáles son condicionales y cuáles son incondicionales, y por qué?

Este programa en Ripes realiza el máximo común divisor. Para ello hacemos una condición while y una condición if.

Cada una se realiza con las siguientes instrucciones:

- BEQ: instrucción de salto que utilizaremos para finalizar el loop una vez el contenido de los 2 registros sea igual. Este caso corresponde con la condición while en C.
- BGT: instrucción de salto en la que, si el contenido del registro a1 es mayor estricto que el contenido de a2, nos derivará al apartado “cert”. Allí realizaremos la resta a = a - b (SUB a1, a1, a2) y regresaremos al loop.
- BLT: instrucción de salto en la que, si el contenido del registro a1 es menor estricto que el contenido de a2, nos derivará al apartado “fals”. Allí realizaremos la resta b = b - a (SUB a2, a2, a1) y regresaremos al loop.

Los saltos condicionales son los descritos anteriormente (beq a1, a2, end // bgt a1, a2, cert // blt a1, a2, fals) y los incondicionales son los 2 “j loop”. Los primeros son condicionales porque, tal y como indica el nombre, para realizar el salto se debe cumplir una condición: =, >, < ...

En cambio las incondicionales no necesitan ningún requisito para volver al loop.

Conclusiones

En esta práctica hemos consolidado conocimeintos en Ripes, como hacer bucles o utilizar sentencias de control de flujo, a través :

- *Realizar los ejercicios planteados, tanto guiados en los directos de youtube como los que son a realizar en casa.*

- *Mirar la guia de Ripes subida al campus para consultar las condiciones de salto y poder completar los informes de los ejercicios propuestos.*

- *Observar el funcionamiento de la máquina del Ripes: resultados guardados en memoria y diversas pruebas con el código.*

En resumen, doy por cumplidos los objetivos propuestos más arriba.