PRÁCTICA 3 IO

Carlos Raso Alonso

Índice

Objetivos

En esta práctica se pretende que podamos transformar sentencias de control de flujo en lenguaje asemblador con la utilización de saltos condicionales e incondicionales.

1.

1, 2 y 3

La primera instrucción de salto que podemos ver es de tipo branch y es blt. En esta instrucción, si a1 < a2 (correspondiente, según hemos asignado los registros, a a < b) se salta a la instrucción etiquetada como “falso” para que se guarde la operación b – a (a2 - a1)en el registro a3 (que almacenará el resultado final, |a - b|). En caso contrario se pasa a la siguiente instrucción (con etiqueta “cierto”). Dentro de la sección “cierto”, se ejecuta la operación a – b (a1 – a2) y se guarda en el registro solución, a3. Al final de esta sección, nos encontramos el otro único salto del código. En este caso es una instrucción de salto incondicional (j). Cuando se ha ejecutado el código que empieza en la etiqueta “cierto” (en la que solo se entra si a1 >= a2, gracias a la instrucción “blt a1, a2, falso”), se usa la instrucción de salto incondicional j end, que hace que el registro pc se sitúe directamente en la posición de memoria etiquetada como “end” que contiene la instrucción final del programa, en la que se almacena el resultado . El motivo de poner al final de esta sección un salto incondicional es para no ejecutar la instrucción con etiqueta “falso” si el pc está en la sección “verdadero”, e ir directamente al final del programa. Adjunto foto del código del programa:

```
1  .data
2  a: .word -3
3  b: .word -6
4  resultat: .word 0
5  .text
6      la a0, a
7      lw a1, 0(a0)
8      lw a2, 4(a0)
9      blt a1, a2, falso
10 cierto:
11     sub a3, a1, a2
12     j fin
13 falso:
14     sub a3, a2, a1
15 fin:
16     sw a3, 8(a0)
```

2.

1.

Un salto hacia detrás suele indicar una estructura de control de flujo de tipo for o while

Valor inicial	Valor en “resultado”	# ciclos	# instrucciones
F 0	0h	0	9
F 1	1h	1	15
F 2	1h	2	21
F 3	2h	3	27
F 4	3h	4	33
F 5	5h	5	39
F 6	8h	6	45
F 25	12511h	25	159
F 46	6d73e55fh	46	285
F 47	2fh	47	291

2. Lo que ocurre con el resultado F 47 es que es un número que no se puede representar con los 4 bytes que tiene un registro porque es muy grande. Esto hace que cuando se sume F 46 con F45 (almacenados en los registros a1 y a2 en el penúltimo bucle), se produzca un error de desbordamiento y el resultado de la suma no sea coherente (se suman dos números enteros positivos y da otro número entero positivo más pequeño que ellos). En este programa no tenemos en cuenta este tipo de errores de desbordamiento al sumar y por eso el valor de F47 no tiene sentido. Se podría arreglar este error viendo si en la suma se produce un error de desbordamiento.

   Adjunto una captura de pantalla de mi programa:

```
.data
comptador: .word 47
resultat: .word 0
.text
    la a0, comptador
    lw a0, 0(a0)
    addi a1, zero, 0
    addi a2, zero, 1
loop:
    blez a0, end
    add a3, a2, a1
    addi a1, a2, 0
    addi a2, a3, 0
    addi a0, a0, -1
    j loop
end:
    la a0, resultat
    sw a1, 0(a0)
```

3.

Mi programa primero guarda en las dos primeras posiciones de memoria de .data los dos números input, que se etiquetan como a y b. También se almacena en la tercera posición de memoria el resultado del mcd, al principio este valor se inicializa en 0. En las primeras instrucciones se carga en el registro a0 la dirección de memoria etiquetada con a y se cargan en los registros a1 y a2 el contenido de la primera y segunda posición de memoria de .data. Una vez hecho esto se entra en el bucle que empieza en la instrucción etiquetada por loop. Al principio de este loop se comprueba la condición de salida. Si a1 == a2 al principio del bucle se produce un salto de pc a la instrucción etiquetada por end, al final del programa. Si a1 != a2 al principio del bucle entonces el mcd aún no se ha calculado por lo que el bucle se ejecuta esa vez. Al principio del bucle nos encontramos con la condición del if. Si a1 <= a2 se salta a la instrucción etiquetada con falso gracias a la segunda instrucción de tipo branch del bucle loop. En este caso, en la instrucción etiquetada con falso se opera con los registros a1 y a2 correspondientemente si a1 <= a2 y luego se salta al principio de bucle con una instrucción de salto incondicional a la etiqueta loop (j loop). En caso que la instrucción de salto tipo branch del if no se cumpla, el pc salta a la siguiente instrucción, la etiquetada como cierto. En este caso se opera a1 y a2 como se deben operar si a1 > a2. Por último en esta sección del if, de nuevo, se realiza un salto incondicional a la instrucción loop para repetir el bucle. Una vez el bucle ha realizado suficientes iteraciones, de tal modo que a1 ya alberga el mcd entre a y b, es decir, cuando a1 == a2, se va a la instrucción etiquetada con end. En la última instrucción se guarda en la posición de memoria de .data con etiqueta resultado (a la que se accede sumando 8 a la dirección de memoria almacenada en a0) el valor del registro a1, que en este punto del programa guarda mcd(a, b), como queríamos. Como ya he explicado, en el programa hay dos saltos incondicionales y dos saltos condicionales (de tipo branch). Los dos saltos incondicionales corresponden al las dos posibilidades de final del bucle, es decir, si a1 > a2 o a1 <= a2. En cada uno de estos casos, cuando ya hemos operado con a1 y a2, necesitamos siempre volver a empezar el bucle para que se continúe el proceso de calcular el mcd y por ello se usa un salto incondicional. Los dos saltos condicionales del código se encuentran al principio del bucle loop, con la finalidad de salir del bucle loop en cuanto no se necesiten más iteraciones porque ya se ha obtenido el mcd (a1 == a2). En este caso se salta al final del bucle que corresponde a la instrucción end. El segundo salto condicional, inmediatamente posterior al primero, corresponde a la condición del if de dentro del bucle while. Con este salto condicional se separan los dos casos que corresponden a tratamientos distintos de a1 y a2. Lógicamente estos dos saltos deben ser condicionales porque si se realizan o no dependen directamente del valor de a1 y a2.

```
1  .data
2  a: .word 252
3  b: .word 105
4  resultado: .word 0
5  .text
6      la a0, a
7      lw a1, 0(a0)
8      lw a2, 4(a0)
9  loop:
10     beq a1, a2, end
11     ble a1, a2, falso
12 cierto:
13     sub a1, a1, a2
14     j loop
15 falso:
16     sub a2, a2, a1
17     j loop
18 end:
19     sw a1, 8(a0)
```

Conclusiones

En esta práctica he profundizado en mi conocimiento sobre saltos condicionales e incondicionales para poder generar sentencias de control de flujo en el código, lo que me permitirá construir códigos más complejos en lenguaje ensamblador.