DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2020)

Ayudantía 4

14 de Septiembre de 2020

1. Ex - 2019 - 1

Para cada uno de los siguientes programas escritos en assembly del computador básico, indica si durante la ejecución se produce el salto o no. Justifica con los valores de las señales Z, N, C y V entregados por la ALU.

I.
```
MOV A,3
CMP A,4
JMP Fibonacci
...
```

II.
```
MOV A,1
MOV B,-9
SHR A, A
JEQ Arquiyudantes
...
```

III.
```
MOV A,5
MOV B,-5
XOR A,B
JNE BonusLealtad
...
```

IV.
```
MOV A,7
MOV B,-2
INC B
JLT SuHansidad
...
```

V.
```
MOV A, 12
SUB A, 1
JCR JurgenMandaSaludos
...
```

VI.
```
MOV A,127
ADD A,127
JOV Germoji
...
```

Solución:

(a) I. Como JMP es un salto condicional, independiente de los condition codes, siempre salta.

(b) II. La última instrucción es $1 >> 1$, que es igual a 0, por lo que si salta.

(c) III. -5 XOR $5 = -2$.$Z = 1$ por lo que JNE salta.

(d) IV. $-2 + 1 = -1$, entonces $N = 1$ y $Z = 0$. Por lo tanto, JLT salta.

(e) V. Programamos $12 - 1$, pero la ALU hace $12 + 255$, lo que provoca un carry, por lo que JCR salta.

(f) VI. Aquí tenemos una suma entre números positivos, que da como resultado un número negativo. Esto significa que hay overflow, por lo que JOV salta.

2. I1 2018 - 1

Considera el código en assembly que se muestra más abajo.

(a) Explica el propósito de, y cómo funciona, la secuencia de instrucciones PUSH B, CALL func y POP B; y cuál es el efecto de que no haya una instrucción PUSH A (justo antes) ni una instrucción POP A (justo después).

(b) Explica el propósito de, y cómo funciona, la secuencia de instrucciones CMP A,0 y JEQ end, y en particular qué ocurre cuando el contenido del registro A es 0.

Tanto en (a) como en (b), identifica las componentes del computador básico que están involucradas y explica qué rol juegan.

```
DATA:                    func :
    r 3                      s h i f t :
                                 MOV  A,B
CODE :                           CMP  A, 0
    MOV  A, ( r )                JEQ end
    MOV  B, 2                    DEC  A
    PUSH  B                      MOV  B,A
    CALL  func                   MOV  A, ( r )
    POP  B                       SHL  A,A
    JMP  f i n i s h             MOV  ( r ) ,A
                                 JMP  s h i f t
finish:                      end :
                                 RET
```

Figura 1: Código pregunta 2.

Solución:

(a) Como podemos darnos cuenta en el código, la instrucción **CALL func** ejecuta la subrutina del mismo nombre, sin embargo, al tener PUSH B y POP B justo antes y después de la llamada a la subrutina, nos permitirá utilizar el registro B, sin que sus actualizaciones se vean reflejadas en nuestro código principal. Esto es posible por que la instrucción PUSH B guarda el valor de B en el stack, y luego este mismo valor es recuperado con POP B.

Por esto mismo, al no tener las instrucciones PUSH A y POP A, los cambios que se hagan en el registro A durante la ejecución de nuestra subrutina, se verán reflejados en nuestro código.

(b) El objetivo de CMP A,0 y JEQ end, es terminar la subrutina func en el caso de que A sea igual a 0. La instrucción CMP A,0 hará la resta A-0. No se guardará el resultado de esta operación, pero sí se activará Z = 1 en el registro Status en caso de A = 0.

Luego, la instrucción JEQ ejecutará el salto hasta el label *end* (y por lo tanto finalizando la ejecución) si el bit Z es 1.

3. I1 2019-2

El siguiente programa escrito en python implementa un algoritmo de conteo de los bits distintos entre dos números. Traduzca el programa al assembly del computador básico, cumpliendo lo siguiente:

- Su programa debe ser una traducción directa del programa en python, es decir, además de traducir la funcionalidad (conteo de los bits distintos) debe hacerlo implementando el mismo algoritmo descrito en el programa, incluyendo las funciones como subrutinas.
- Su programa debe definir (al menos) las variables var_a, var_b que se ocuparán como operandos para el conteo de los bits distintos, y la variable var_c que almacenará el resultado.

```
1  def contar_bits_seteados(n):
2    cuenta = 0
3    while n!=0:
4      cuenta += n & 1
5      n >>= 1
6    return cuenta
7
8  def contar_bits_distintos(var_a, var_b):
9    return contar_bits_seteados(var_a^var_b)
10
11
12 # Programa principal
13 def main():
14     var_a = 10
15     var_b = 20
16     var_c = 0
17     var_c = contar_bits_distintos(var_a,var_b)
18
19 if __name__ == "__main__":
20     main()
```

Solución:

```
DATA:
var_a       10
var_b       20
var_c       0
cuenta      0
CODE:
  MOV A,(var_a)
  MOV B,(var_b)
  CALL contar_bits_distintos
  MOV (var_c),A
  JMP end

contar_bits_distintos:
  XOR A,B
  CALL contar_bits_seteados
  RET

contar_bits_seteados:
  CMP A,0
  JEQ no_while
  MOV B,A
  AND A,1
  ADD A,(cuenta)
  MOV (cuenta), A
  MOV A,B
  SHR A,A
  JMP contar_bits_seteados
  no_while:
    MOV A, (cuenta)
    RET
end:
```