Guía de ejercicios # 8

Organización de Computadoras 2017

UNQ

Arquitectura Q4

Características

- Tiene 8 registros de uso general de 16 bits: R0..R7.
- La memoria utiliza direcciones de 16 bits.
- Contador de programa (*Program counter*) de 16 bits.
- *Stack Pointer* de 16 bits. Comienza en la dirección `0xFFEF`.
- Flags: Z, N, C, V (Zero, Negative, Carry, oVerflow). Instrucciones que alteran Z y N: ADD, SUB, CMP, DIV, MUL, AND, OR, NOT. Las 3 primeras además calculan C y V.

Instrucciones de dos operandos

Formato de Instrucción

CodOp (4b)	Modo Destino (6b)	Modo Origen (6b)	Destino (16b)	Origen (16b)

Tabla de instrucciones

Operación	Cod Op	Efecto
MUL	0000	Dest ← Dest * Origen [1]
MOV	0001	Dest ← Origen
ADD	0010	Dest ← Dest + Origen
SUB	0011	Dest ← Dest - Origen
CMP	0110	Dest - Origen
DIV	0111	Dest ← Dest % Origen [2]

Instrucciones de un operando origen

Formato de Instrucción

CodOp (4b)	Relleno (000000)	Modo Origen (6b)	Operando Origen (16b)

Tabla de instrucciones

Operación	Cod Op	Efecto
CALL	1011	[SP] ← PC; SP ← SP - 1; PC ← Origen
JMP	1010	PC ← Origen

[1] El resultado de la operación `MUL` ocupa 32 bits, almacenándose los 16 bits menos significativos en el operando destino y los 16 bits mas significativos en el registro `R7`

[2] El caracter % denota el cociente de la división entera

Instrucciones sin operandos

Formato de Instrucción

CodOp (4b)	Relleno (000000000000)

Tabla de instrucciones

Operación	CodOp	Efecto
RET	1100	SP ← SP + 1; PC ← [SP]

Saltos condicionales

Formato de Instrucción

Cod_Op (8)	Desplazamiento(8)

Los primeros cuatro bits del campo `Cod_Op` es la cadena 1111_2. Si **al evaluar la condición de salto** el resultado es 1, se le suma al PC el valor del desplazamiento, representado en $CA2(8)$. En caso contrario la instrucción no hace nada.

Codop	Op.	Descripción	Condición de Salto
0001	JE	Igual / Cero	Z
1001	JNE	No igual	not Z
0010	JLE	Menor o igual	Z or (N xor V)
1010	JG	Mayor	not (Z or (N xor V))
0011	JL	Menor	N xor V
1011	JGE	Mayor o igual	not (N xor V)
0100	JLEU	Menor o igual sin signo	C or Z
1100	JGU	Mayor sin signo	not (C or Z)
0101	JCS	Carry / Menor sin signo	C
0110	JNEG	Negativo	N
0111	JVS	Overflow	V

Modos de direccionamiento

Modo	Codificación
Inmediato	`000000`
Directo	`001000`
Registro	`100rrr`[3]

[3] `rrr` es una codificación (en 3 bits) del número de registro

1 Flags y saltos

A veces cuando programamos nos surge la necesidad de bifurcar el flujo de nuestro programa, esto quiere decir que en algunos casos queremos ejecutar una acción u otra dependiendo si determinada condición se cumple o no.
Debido a que nuestros programas se ejecutan de manera secuencial siguiendo el registro PC, para lograr este efecto necesitamos un mecanismo que nos permita modificar el valor del PC en el medio de la ejecución, por este motivo surgen las instrucciones conocidas como **Saltos**.

Saltos

Los **Saltos** son instrucciones que nos permiten llevar el flujo del programa hacia determinada dirección de memoria que nosotros queramos (O sea, cambia el PC), esta debe estar señalada por una etiqueta. Existen dos tipos de saltos:

- **Saltos incondicionales** La actualización de PC se lleva a cabo siempre que se ejecute el salto.
- **Saltos condicionales** La actualización de PC se lleva a cabo si se cumple determinada condición.

Ahora en Q4 tenemos un nuevo conjunto de instrucciones que nos permite realizar saltos condicionales e incondicionales. La instrucción de salto incondicional es **JMP**, esta funciona similar a un **CALL**, tiene la forma: `JMP etiqueta` y al ejecutarse carga el PC con la dirección de memoria donde está `etiqueta`. La diferencia que tienen es que el textbfJMP no modifica la pila.

Las instrucciones de Salto condicional son **JE, JNE, JLE, JG, JL, JGE, JLEU, JGU, JCS, JNEG** y **JVS**. Cada una de estas evalúa una condición distinta (*ver cuadro de saltos de la página 1*), tienen la forma: `JE etiqueta` y modifica el PC si la condición del salto se cumple, de lo contrario no hace nada. Ahora lo que nos falta es saber como podemos representar una condición, esto lo logramos a través de los **Flags**.

Flags

Los **Flags** se encuentran en el CPU y son bits que se usan para caracterizar el resultado de la ALU. Cuando se ejecuta una instrucción **CMP, ADD, SUB, MUL** o **DIV**, la ALU tiene la tarea de realizar a operación correspondiente y de dar el resultado, a partir de este resultado se determinan los valores de los **Flags**. Los Saltos condicionales ven los valores de los **Flags** y actúan en consecuencia.

Los flags de nuestra arquitectura son 4, cada uno indica una condición distinta:

- **Z (Zero)** Esta señal se enciende cuando el resultado es una cadena completa de ceros.
- **N (Negative)** Esta señal se encienda cuando el primer bit del resultado es 1.
- **C (Carry)** El flag de carry se enciende cuando a resta o la suma tuvieron acarreo o borrow.
- **V (Overflow)** El flag de Overflow (V) se enciende para indicar una situación de error en un sistema con signo.
  1. `positivo + positivo = negativo`
  2. `negativo + negativo = positivo`
  3. `negativo - positivo = positivo`
  4. `positivo - negativo = negativo`

CMP

El **CMP** es una instrucción que se incluye entre las instrucciones de dos operandos, cuyo objetivo es comparar los operandos por medio de una resta y modifica los flags en base al resultado. Pero como la resta se lleva a cabo sólo a los fines de comparar, su resultado se descarta y esto implica que no se modifique el destino (a diferencia de las demás instrucciones de dos operandos).

Ejercicios:

1.a Realizar las siguientes operaciones en $BSS(4)$ y calcular los flags a partir de los resultados de las mismas.

(a) `1010 + 1001`

(b) `1011 - 1011`

(c) `0010 + 1101`

(d) `0010 - 0111`

(e) `1100 - 1000`

1.b Dar los valores de R3 y R4 (y calcular los flags de la primer instrucción) que hagan que se ejecute la instrucción CALL:

(a)
```
rutina: CMP  R3, R4
        JLE fin
        CALL boom
   fin: RET
```

(b)
```
rutina: CMP  R3, R4
        JLEU fin
        CALL boom
   fin: RET
```

(c)
```
rutina: CMP  R3, R4
        JL fin
        CALL boom
   fin: RET
```

(d)
```
rutina: CMP  R3, R4
        JCS fin
        CALL boom
   fin: RET
```

1.c Diseñar un circuito que calcule el **flag Z** a partir de una suma en $BSS(4)$. Considerar que se tiene disponible un sumador de 4 bits.

1.d Diseñar un circuito que calcule el **flag N** a partir de una suma en $BSS(4)$. Considerar que se tiene disponible un sumador de 4 bits.

1.e Diseñar un circuito que calcule el **flag V** a partir de una suma en $BSS(4)$. Considerar que se tiene disponible un sumador de 4 bits.

1.f Diseñar un circuito que calcule el **flag V** a partir de una resta en $BSS(4)$. Considerar que se tiene disponible un restador de 4 bits.

1.g Indique verdadero o falso. Justifique.

(a) La instruccón JMP no modifica el SP.

(b) Los flags se modifican con todas las instrucciones de dos operandos.

(c) La instruccón JE es un salto incondicional.

2 Ensamblado de saltos

Antes de mostrar como se ensamblan los saltos hay que mencionar otra característica de estos. Dijimos que los saltos modifican el valor de PC, hay dos formas lograr esto:

- **Salto absoluto:** El nuevo valor de PC se expresa en términos de una dirección de memoria.
- **Salto relativo:** El nuevo valor de PC se expresa en términos de un desplazamiento con respecto a la siguiente instrucción.

El **JMP** es un Salto absoluto con lo cual el valor del origen debe ser la dirección de memoria a donde queremos saltar, nosotros expresamos esta dirección como una etiqueta, por esta razón el **JMP** al igual que el **CALL** ocupará dos celdas, una que contiene el código de operación + relleno + modo inmediato y la segunda la dirección de memoria a la que se quiere saltar.

El resto de los **Saltos**, osea los condicionales, son saltos relativos. Este tipo de saltos no pisa en valor del PC con una dirección de memoria totalmente nueva, sino que calculan a donde tienen que saltar utilizando un desplazamiento, este desplazamiento se va a expresar como un número en $CA2(8)$ que va a venir ensamblada en la misma celda del salto. Se utiliza **CA2** porque al disponer de números negativos podemos realizar saltos tanto para adelante como para atrás. El formato de estas instrucciones es

Cod_Op (8)	Desplazamiento(8)

, donde el código de operación de 8 bits siempre empieza con el prefijo `1111` y los siguientes 4 bits son el código de operación del salto que se haya elegido, los últimos 8 bits son el desplazamiento expresado en $CA2(8)$.

Ejercicios:

2.a Considere el siguiente programa.

```
prog:  SUB R0, 0x0001
       JE pisar
       MOV R3, [0A0A]
pisar: MOV R3, 0xFFFF
       RET
```

(a) Ensamblar a a partir de la celda `FF0E`.

(b) ¿Que valor tiene el desplazamiento del salto `JE`?

(c) ¿A que celda queda asociada la etiqueta `pisar`?

(d) Explique que hace el programa

2.b Considere el siguiente programa.

```
     prog: SUB R0, 0x0001
           JE escero
           ADD R0, 0x000B
           SUB R0, 0x000A
           JLEU mayoradiez
   escero: MOV R3, 0xFFFF
mayoradiez:MOV R3, 0xFFFF
           RET
```

(a) Ensamblar a a partir de la celda `FF0E`.

(b) ¿Que valor tiene el desplazamiento del salto `JE`?

(c) ¿Que valor tiene el desplazamiento del salto `JLEU`?

(d) ¿A que celda queda asociada la etiqueta `escero`?

(e) ¿A que celda queda asociada la etiqueta `mayoradiez`?

(f) Explique que hace el programa

2.c Dado el siguiente programa:

```
prog: MOV R0, R2
      ADD R0, [1C00]
      JE alla
      SUB R3, 0x0001
alla: MOV R1, [1B00]
      RET
```

(a) Ensamblar a a partir de la celda `2001`.

(b) ¿Que valor tiene el desplazamiento del salto `JE`?

(c) ¿A que celda queda asociada la etiqueta `alla`?

2.d Dado el siguiente mapa de memoria, simule la ejecución de un programa que comienza en la celda A893, asumiendo que R0 = 0000 y R1 = F000

	...
A893	6821
A894	FC04
A895	1980
A896	FFFF
A897	A000
A898	A89B
A899	1980
A89A	AAAA
A89B	C000
	...

2.e Simule la ejecución del programa en 2.d, asumiendo que R0 = F000 y R1 = 0000

2.f Simule la ejecución del programa en 2.d, reemplazando el valor de la celda A894 por el valor FA04 y asumiendo que R0 = 0000 y R1 = F000.

2.g Dado el siguiente mapa de memoria, simule la ejecución de un programa que comienza en la celda A899, asumiendo que el valor de R0 es 2.

	...
A893	3800
A894	0001
A895	F102
A896	A000
A897	A893
A898	C000
A899	B000
A89A	A893
	...

3 Estructura condicional

Nota: Documente todas las rutinas y programas.

3.a Escribir un programa que, si el valor en R0 es igual al valor en R1, ponga en R2 un 1 ó 0 en caso contrario

3.b Escribir un programa que, si el valor en R7 es negativo, le sume 1, o le reste 1 en caso contrario

3.c Escribir un programa que ponga en R2 el máximo valor entre R0 y R1. Considerar que los valores están en $BSS()$

3.d Escribir un programa que ponga en R2 el máximo valor entre R0 y R1. Considerar que los valores están en $CA2()$

3.e Escribir un programa `absCA2` que determina si el valor almacenado en el registro `R7` es negativo (en $CA2()$). En caso de serlo, lo reemplaza por su valor absoluto.

3.f Escribir un programa que utilice la rutina `avg` de la práctica número 4 para determinar si el promedio de los valores almacenado en las celdas `1000` y `1004` es mayor al valor `1010`, en dicho caso ponga 1 en `R1`

3.g Escribir un programa que utilice la rutina `sumaDos` de la práctica número 4 y determine si el valor resultado es mayor al contenido de `R4`, en ese caso poner un 1 en la celda `00FE`

3.h Escribir un programa que, si el valor en `R1` es 0 ponga en `R7` el valor almacenado en la celda `0FFE`. En caso contrario guarde en `R7` la suma de `R2` y `R3`

3.i Escribir una rutina `init` que, si el valor almacenado en `R1` es `FFFF` , inicialice con 0 el resto de los registros.