Guía de ejercicios # 4
Q3, Rutinas y Pila

Organización de Computadoras 2016

UNQ

Arquitectura Q3

Características

- Tiene 8 registros de uso general de 16 bits: R0..R7.
- La memoria utiliza direcciones de 16 bits.
- Tiene un contador de programa (*Program counter*) de 16 bits.
- *Stack Pointer* de 16 bits. Comienza en la dirección `0xFFEF`.

Instrucciones de dos operandos

Formato de Instrucción				
CodOp (4b)	Modo Destino (6b)	Modo Origen (6b)	Destino (16b)	Origen (16b)

Tabla de códigos de operaciones		
Operación	Cod Op	Efecto
MUL	0000	Dest ← Dest * Origen [1]
MOV	0001	Dest ← Origen
ADD	0010	Dest ← Dest + Origen
SUB	0011	Dest ← 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 códigos de operaciones		
Operación	Cod Op	Efecto
CALL	1011	[SP] ← PC; SP ← SP - 1; PC ← Origen

Instrucciones sin operandos

Formato de Instrucción	
CodOp (4b)	Relleno (000000000000)

Tabla de códigos de operaciones		
Operación	CodOp	Efecto
RET	1100	SP ← SP + 1; PC ← [SP]

[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

Modos de direccionamiento

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

Ejercicios

1 Creación e invocación de rutinas

Una rutina es un programa acotado que queremos usar más de una vez en combinación con otros programas. Esto nos sirve para reutilizar código partiendo problemas más grandes en otros más pequeños.

1.1 Q3: CALL y RET

Q3 agrega nueva funcionalidad a las arquitecturas Q permitiéndonos ahora ser capaces de crear e invocar rutinas. Esto se hace a través de las intrucciones CALL (de un operando origen) y RET (sin operandos).
Para llamar una rutina existente utilizamos la instrucción CALL cuyo operando será una etiqueta ¿Qué es una etiqueta? es un nombre asociado a dicha rutina que tiene el valor de la dirección de memoria donde inicia.
Es necesario, cuando terminamos de ejecutar una rutina, retornar al punto del programa donde nos hemos quedado (la siguiente instrucción al `CALL etiqueta`). Para esto Q3 agrega la instrucción RET, que al ejecutarse vuelve al punto del programa luego del último CALL invocado.

1.2 Creación de rutinas

Supongamos que queremos crear una rutina que simplemente sume el número tres al valor almacenado en `R0` y lo guarde en `R1` .

- Código de la rutina
  En el cuerpo de la rutina escribiremos las instrucciones para que la computadora realice las operaciones indicadas para lograr el efecto que deseamos (en este caso que luego de invocar la rutina `R1` sea igual a `R0` + 3). Y luego finalizaremos con la instruccion RET que nos permitirá continuar con la ejecución del programa desde el cual invocamos la rutina.

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

El código de la rutina podría ser el siguiente:

```
MOV R1, R0
ADD R1, 0x0003
RET
```

- Etiqueta de la rutina
  Al crear una rutina para identificarla y luego poder invocarla cuando nos sea necesario la nombraremos con una etiqueta (nombre descriptivo).
  Esa etiqueta tendra como valor asociado la dirección de memoria a partir de donde esta ensamblada dicha rutina.
  Siguiendo con el ejemplo anterior, a nuestra rutina podríamos asignarle la etiqueta **sumaTres** asociandolo al código de la siguiente manera:

```
sumaTres:  MOV R1, R0
           ADD R1, 0x0003
           RET
```

- Documentación de la rutina
  Es importante para usar una rutina contar con una documentación que especifique como hacerlo ya que no debería ser necesario conocer como esta programada para invocarla.
  Por eso cada vez que creamos una rutina debemos **documentarla** especificando:
  - **Requiere**: Si la rutina recibe párametros en que celdas de memoria o registros lo hace y cuáles son las precondiciones para invocar la rutina.
  - **Modifica**: Las celdas de memoria o registros cuyo valor antes de invocar la rutina serán diferentes luego de hacerlo, es decir, las celdas de memoria o registros cuyos valores son "pisados".
  - **Retorna**: En que celda de memoria o registro retorna el/los resultado/s.

  Entonces la documentación de la rutina **sumaTres** podría ser la siguiente:
  - **Requiere**: Valor al que se desea sumarle tres almacenado en `R0`
  - **Modifica**: `R1`
  - **Retorna**: En `R1` el resultado de sumarle tres al número almacenado en `R0`

1.3 Invocación de rutinas

Supongamos que necesitamos crear un programa que calcule el área de dos cuadrados cuyo tamaño de lado en cm se encuentran almacenados en las celdas de memoria 0xA123 y 0xA124 y guardar el resultado en ambas celdas respectivamente.
Contamos con la rutina existente **areaCuadrado** que calcula el área de un cuadrado cuya documentación es la siguiente:

- **Requiere**: La medida de lado del cuadrado en cm almacenada en el registro `R3` .
- **Modifica**: `R3` y `R4` .
- **Retorna**: En `R4` el área de un cuadrado de medida de lado almacenado en el registro `R3` .

Entonces podríamos invocar a la rutina **areaCuadrado** para calcular ambas áreas aunque no sepamos como esta programada teniendo en cuenta la documentación de la siguiente manera:

- Modifica

  Este item de la documentación nos va a indicar que registros y/o celdas de memoria se modifican durante la ejecución de la rutina, por lo cual, si no queremos perder los datos que tenemos almacenados en ellos deberíamos resguardarlos copiándolos en otro lado.
  La documentación dice textualmente: "*Modifica: `R3` y `R4` *"; entonces, para empezar con la porción del programa que calculará el área del cuadrado cuyo lado en cm esta almacenado en la celda 0xA123, suponiendo que no quiero perder los datos almacenados en `R3` y `R4` debería copiarlos a otros registros por lo que mi programa podría comenzar de la siguiente manera:

  ```
  MOV R1, R3
  MOV R2, R4
  ```

  De esta manera tengo resguardados los datos almacenados en `R3` y `R4` para usarlos luego.

- Requiere

  Este item de la documentación nos va a indicar dónde poner los datos antes de invocar la rutina utilizando la instrucción CALL.
  La documentación dice textualmente: "*Requiere: La medida de lado del cuadrado en cm almacenado en el registro `R3` *"; entonces, ahora que ya tenemos resguardados los datos que eran de nuestro interes, deberíamos mover al registro `R3` el contenido almacenado en la celda de memoria 0xA123 para luego invocar a la rutina **areaCuadrado**.

  ```
  MOV R1, R3
  MOV R2, R4
  MOV R3, [0xA123]
  CALL areaCuadrado
  ```

- Retorna

  Este item de la documentación nos va a indicar dónde son almacenados los datos que genera la rutina para luego poder usarlos.
  La documentación dice textualmente: "*Retorna: En `R4` el área de un cuadrado de medida de lado almacenado en el registro `R3` *". En este caso el área del cuadrado se almacena en el registro `R3` y como nuestro programa debe almacenar ese dato en la celda de memoria 0xA123, tendremos que moverlo.

```
MOV R1, R3
MOV R2, R4
MOV R3, [0xA123]
CALL areaCuadrado
MOV [0xA123], R3
```

De esta manera y teniendo en cuenta la documentación, pudimos invocar la rutina para calcular el área del primer cuadrado. Ahora para calcular el área del segundo podemos volver a invocarla siguiendo los mismos pasos:

```
MOV R1, R3
MOV R2, R4
MOV R3, [0xA123]
CALL areaCuadrado
MOV [0xA123], R3
MOV R3, [0xA124]
CALL areaCuadrado
MOV [0xA124], R3
```

1.4 Ejercicios

1. Escriba una rutina `mulPorDos`, que multiplique por 2 el contenido de `R1` y guarde el resultado en `R1`
   Documente la rutina especificando requiere, retonra y modifica.

2. Utilice la rutina anterior para escribir un **programa** que calcule 2 elevado a la 5.

3. Escriba una rutina `mulPorCuatro`, que multiplique por 4 el contenido de `R1` y guarde el resultado en `R1` . **Documente la rutina** especificando requiere, retonra y modifica.

4. Utilice la rutina anterior para escribir un **programa** que calcule 4 elevado a la 4.

5. Escriba una rutina `aLaQuinta`, que eleve a la 5ta potencia el contenido del registro `R0` . **Documente la rutina** especificando requiere, retonra y modifica..

6. Utilizando la rutina del ejercicio anterior, escriba un programa que calcule n^5 para los números que están almacenados en la memoria desde la celda `0x1000` hasta la celda `0x1007` **inclusive**.

7. Escriba una rutina `esPar`, que determine si el contenido de `R0` es par o impar de la siguiente manera: Si es par, debe guardar un 0 en `R1` , en caso contrario, debe guardar un 1 en dicho registro. **Documente la rutina** especificando requiere, retonra y modifica.

8. Utilizando la rutina del ejercicio anterior, escriba un programa que grabe un 0 o un 1 en `0x0000`, ..., `0x0005` según si los números de `0xF000`, ..., `0xF005` son pares o no.

9. Escribir una rutina `swapR0R1` que intercambie los valores de los registros R0 y R1. Documente su rutina.

10. Escribir un programa que utilice la rutina anterior para intercambiar los valores de las celdas consecutivas entre la 0x3000 y la 0x3009. Esto es: intercambiar el valor de **X** con el valor de **X**+1, siendo **X** una celda cuya dirección es par, en el rango [0x3000, 0x3009].

11. Escribir una rutina `avg` que calcule el promedio entre R1 y R2, guardando el calculo en R3. **Documente la rutina** especificando requiere, retonra y modifica.

12. Sabiendo que en R1, R2, R4 y R5 se encuentran almacenadas las edades de los profes de ORGA, calcular el promedio total utilizando la rutina del ejercicio anterior.

13. Se cuenta con la subrutina `maxInt` que calcula el maximo entre los valores $BSS(16)$ de R6 y R7, dejando el resultado en R6.
    Escribir un programa que calcule el maximo de los registros R1 al R7 **inclusive**.

14. Se cuenta con las siguientes documentaciones de la rutinas `sumados` y `aplicarDescuento`:

```
; SUMADOS
; REQUIERE: Dos valores a sumar almacenados
            en R1 y R2
; MODIFICA: R1
; RETORNA: En R1 la suma de los valores
           almacenados en R1 y R2.
```

```
; APLICARDESCUENTO
; REQUIERE: El precio unitario en la celda 0xA000
;           El porcentaje a aplicar
;                 en la celda 0xA001
; MODIFICA: R0
; RETORNA: El precio con el descuento
;          aplicado en R0.
```

    Utilizando las rutinas `aplicarDescuento` y `sumaDos`, escriba un programa que calcule el precio final a pagar por una persona que compra dos productos, cuyos precios unitarios están almacenados en R6 y R7 y se le debe aplicas un descuento del 10%.

15. Dada la siguiente secuencia de números 2, 4, 6, 8, 10 escribir un programa que calcule la sumatoria utilizando la rutina `sumaDos`.

16. Teniendo en cuenta lo ejercitado en esta sección, dado un programa que quiere invocar a una rutina:

    (a) ¿Quién debe hacer el call: el programa o la rutina?

    (b) ¿Quién debe hacer el ret: el programa o la rutina?

2 Implementación del CALL y RET

Teniendo en cuenta la implementación del CALL y el RET:

1. ¿Que relación existe entre los registros IR y PC?

2. Explique detalladamente como funciona el CALL y el RET.

3 Simular la ejecución de rutinas

3.1 Tabla de ejecución

Durante la ejecución de un programa se realizan cambios en los registros especiales PC, SP y en la pila que dependerán de las instrucciones que lo conformen. Para seguir paso a paso estos cambios completaremos la siguiente tabla simulando ejecución instrucción por instrucción:

PC	Instruccion	PC-BI	Efecto	PC-Ex	SP	Pila

Donde **PC-BI** es el pc luego de la Busqueda de Instrucción y **PC-Ex** es el PC luego de la ejecución.

Por ejemplo, supongamos que estamos completando el cuadro de la ejecución del siguiente programa:

```
programa: MOV [0x0001], R0
          RET
```

Y tenemos los siguientes datos:

- El programa esta ensamblado a partir de la celda 0x00F0
- SP = 0xFFEE
- La pila tiene un solo elemento: 0xEFF0
- El contenido de R0 es 0xC0C0

Con estos datos más el código del programa podemos comenzar a llenar la tabla y tenemos que empezar simulando la ejecución de la primer instrucción, `MOV [0x0001], R0`, de la siguiente forma:

- Por lo datos sabemos que el programa esta ensamblado a partir de la celda 0x00F0, entonces en la columna **PC** el valor será 0x00F0.
- La columna **Instrucción** simplemente se rellena con la instrucción actual: `MOV [0x0001], R0`
- La columna **PC-BI** es el pc luego de la Busqueda de Instrucción. Recordemos que luego de la búsqueda de instrucción el PC contiene la dirección de memoria donde comienza la siguiente instrucciónа ejecutar. Para obtener su valor tenemos que fijarnos en la columna **PC** y saber cuantas celdas ocupa la instrucción actual. Siendo `MOV [0x0001], R0` y sabiendo que ocupa dos celdas, sumamos esa cantidad a 0x00F0. Entonces **PC-BI** es 0x00F2.
- La columna **Efecto** contiene la descripción de lo que sucede al ejecutar esa instrucción tal y como lo define la arquitectura Q.
  En este caso al ser un `MOV` el efecto es el siguiente:
  Dest ← Origen
  Teniendo en cuenta esto y que sabemos que el contenido de R0 es 0xC0C0 en este caso el efecto de `MOV [0x0001], R0` sería:
  `[0x0001]` ← `0xC0C0`
- La columna **PC-Ex** es el PC luego de la ejecución y para completarla tenemos que tener en cuenta la columna **PC-BI** (ya que es el último valor de PC que conocemos) y el efecto de ejecutar la instrucción actual, la columna **Efecto**.
  En este caso al ser un `MOV` en el efecto no se involucra al registro especial PC, por lo que, el PC seguirá siendo el mismo que en la columna **PC-BI** : 0x00F2.
- Según la información dada el SP al comenzar a ejecutar el programa tiene como valor 0xFFEE. Para completar en este caso tenemos que mirar el valor anterior del SP (ya dijimos que es 0xFFEE, nos lo da la información) y la columna **Efecto**.
  En este caso al ser un `MOV` en el efecto no se involucra al registro especial SP, por lo que, el SP seguirá siendo el mismo que el anterior : 0xFFEE.
- Por último, según la información dada, la pila al comenzar a ejecutar el programa tiene un solo elemento: 0xEFF0. Para completar la columna **Pila** tenemos que mirar lo que contiene la pila y nuevamente la columna **Efecto**.
  En este caso al ser un `MOV` en el efecto no se involucra la pila, por lo que, la misma seguirá siendo igual : [0xEFF0].

Para seguir simulando la ejecución tenemos que repetir el proceso con las siguientes instrucciones, una por una en orden, hasta haber terminado el programa, incluyendo el llamado a subrutinas.
La tabla terminada con la simulación de ejecución del programa anterior sería la siguiente:

PC	Instruccion	PC-BI	Efecto	PC-Ex	SP	Pila
0x00F0	MOV [0x0001],R0	0x00F2	..	0x00F1	0xFFEE	[0xEFF0]
0x00F2	RET	0x00F3	..	0xEFF0	0xFFEF	[]

3.2 Ejercicios

1. Sabiendo que:
   - La rutina `mulPorCuatro` se encuentran ensamblada a partir de la celda `0x2000`
   - El programa del punto anterior está ensamblado a partir de la celda `0xF000`
   - La pila está vacía [4]

   Simule los cambios que ocurren en el PC, en el SP y en el contenido de la pila durante la ejecución del mismo completando la 3.1tabla de ejecución:

2. Sabiendo que:
   - Las rutina `avg` se encuentra ensamblada a partir de la celda `0x2000`
   - El programa del ejercicio anterior está ensamblado a partir de la celda `0xF000`
   - La pila está vacía[5]

   Simule los cambios que ocurren en el PC, en el SP completando la tabla de ejecución (denifina en el ejercicio 3.1).

[4] Es decir, el valor de SP es 0xFFEF
[5] Es decir, el valor de SP es 0xFFEF

3. Considere la siguiente rutina:

```
rutina1: MOV R1, R0
         RET
```

y el siguiente programa:

```
programa: CALL rutina1
          CALL rutina1
```

Sabiendo que:

- `rutina1` está ensamblada a partir de la celda `0x00E0`
- el programa está ensamblado a partir de la celda `0x1000`
- PC=1000
- La pila está vacía[5]

Simule los cambios que ocurren en el PC, en el SP completando la tabla de ejecución (denifina en el ejercicio 3.1).

4. Considere las siguientes rutinas:

```
rutina1: MOV R1, R0
         RET

rutina2: RET
```

y el siguiente programa:

```
programa: CALL rutina1
          CALL rutina2
```

Sabiendo que:

- `rutina1` está ensamblada a partir de la celda `0x00E0`
- `rutina2` está ensamblada a partir de la celda `0x00A1`
- el programa está ensamblado a partir de la celda `0x1000`
- PC=`0x1000`
- la pila está vacía[5]

Simule los cambios que ocurren en el PC, en el SP completando la tabla de ejecución (denifina en el ejercicio 3.1).

5. Considere las siguientes rutinas:

```
rutina1: MOV R0, R1
         CALL rutina2
         RET

rutina2: MOV R2, R3
         RET
```

Y el siguiente programa:

```
principal: CALL rutina2
           CALL rutina1
```

Sabiendo que:

- `rutina1` está ensamblada a partir de la celda `0x00F0`
- `rutina2` está ensamblada a partir de la celda `0x00A1`
- el programa está ensamblado a partir de la celda `0x0E00`
- PC=`0x0E00`
- la pila está vacía[6]

¿Cómo varía la pila durante la ejecución del programa `principal`?

6. Considere las siguientes rutinas:

```
rutina1: MOV R1, R0
         CALL rutina2
         RET

rutina2: CALL rutina3
         RET

rutina3: MOV R2, R1
         RET
```

y el siguiente programa:

```
programa: CALL rutina1
          CALL rutina2
          CALL rutina3
```

Sabiendo que:

- `rutina1` está ensamblada a partir de la celda `0x00E0`
- `rutina2` está ensamblada a partir de la celda `0x00A1`
- `rutina3` está ensamblada a partir de la celda `0x0101`
- el programa está ensamblado a partir de la celda `0x1000`
- PC=0x1000
- la pila está vacía[6]

Simule los cambios que ocurren en el PC, en el SP completando la tabla de ejecución (denifina en el ejercicio 3.1).

7. Dado el siguiente programa:

```
programa: SUB R0, 0x0001
          CALL rutina
          DIV R5, 0x0002
```

La siguiente rutina:

[6] Es decir, el valor de SP es 0xFFEF

```
rutina: MUL R0, 0X0003
        MOV R5, R0
        RET
```

Y teniendo los siguientes datos:

- El programa esta ensamblado a partir de la celda 0x0020
- La rutina esta ensamblada a partir de la celda 0x0E00
- La pila esta vacía

Encuentre los errores en la siguiente tabla de ejecución:

PC	Instruccion	PC-BI	PC-Ex	SP	Pila
0x0020	SUB R0,0x0001	0x0021	0x0022	0xFFEF	[]
0x0022	CALL rutina	0x0024	0x0E00	0xFFEE	[0x0024]
0x0E00	MUL R0,0X0003	0x0E02	0x0E02	0xFFEE	[0x0024]
0x0E02	RET	0x0E03	0x0024	0xFFEE	[]
0x0E02	DIV R5,0x0002	0x0E04	0x0E04	0xFFEF	[]

4 Ensamblado de programas

1. Ensamblar el siguiente programa a partir de la celda `0xFF0E`

```
SUB  R0, 0x0001
CALL restarTriple
MOV  R3, [0x0A0A]
ADD  R3, R0

restarTriple: MOV R1, R0
              MUL R1, 0x0003
              SUB R0, R1
              RET
```

   Sabiendo que `restarTriple` se encuentra ensamblado a partir de la celda `0x1000`.

2. Ensamblar el siguiente programa a partir de la celda `0xA010`

```
MUL  R3, 0x0005
CALL restarMitad
SUB  R3, 0X0079

restarMitad: MOV R2, R3
             DIV R2, 0x0002
             SUB R3, R2
             RET
```

   Sabiendo que `restarMitad` se encuentra ensamblado a partir de la celda `0xF120`.

5 Ejercicios para utilizar QSim

1. Utilizando QSim, ejecute paso a paso el siguiente programa, observando el comportamiento de la pila y el registro SP. Indique con qué valor queda el registro R2 al finalizar la ejecución de la cuarta instrucción (`MOV R2, R0`):

```
MOV R0, 0x000A
MOV R1, 0x000B
CALL sumamult
MOV R2, R0

sumamult:
   ADD R0, 0x000C
   ADD R1, 0x000C
   CALL mult
   RET

mult:
   MUL R0, R1
   RET
```

2. En el ejercicio anterior, ¿cómo se traducen las etiquetas *sumamult* y *mult*?

3. Pruebe en QSim, qué efecto tiene utilizar CALL con modos de direccionamiento registro o directo en lugar de una etiqueta, por ejemplo:

```
MOV [0x000A], 0x001A
CALL [0x000A]


MOV R0, 0x000A
CALL R0
```

4. Utilizando QSim, pruebe el siguiente programa:

```
MOV R0, 0x0001
MOV R1, 0x0002
CALL factInfinito

factInfinito:
   MUL R0, R1
   ADD R1, 0x0001
   CALL factInfinito
```

   Luego responda las siguientes preguntas:

   - ¿Qué se está calculando en el registro R0?
   - ¿Qué defecto tiene el programa?
   - ¿Cómo crece la pila? ¿Qué valores se apilan?
   - ¿Dada la arquitectura de 16 bits de QSim, hasta qué punto tiene sentido que el programa continue calculando?