Práctica 3: Arquitectura del CPU

Programación en assembler de ORGA1

Organización del Computador I
DC - UBA

Verano 2018

Ejercicio 1

Escribir una función en assembler de ORGA1 que tome como parámetro un entero de 16 bits en el registro `R1` y devuelva en `R0` un valor según el siguiente código en C:

```
int categoria(int n) {
        int ret;
        if (n >= 0 && n < 7 && n % 2 == 1)
                ret = 4; // caso 1
        else if (n >= 7 && n <= 127)
                ret = 7; // caso 2
        else
                ret = -1; // caso 3
        return ret;
}
```

Ejercicio 1: Solución

```
categoria: MOV R0, -1 ; caso 3 por default
           CMP R1, 0 ; n < 0? caso 3
           JL fin
           CMP R1, 7 ; n >= 0 && n < 7? potencial caso 1
           JL esImpar
           CMP R1, 127 ; n > 127? caso 3
           JG fin
           MOV R1, 7 ; n >= 7 && n <= 127 -> caso 2
           JMP fin
  esImpar: AND R1, 1 ; n >= 0 && n < 7
           JZ fin ; n & 1 == 0? caso 3
           MOV R0, 4 ; caso 1
      fin: RET
```

Ejercicio 2

Escribir una función en assembler de ORGA1 que tome como parámetro una matriz de 8×8 enteros de 16 bits y utilice la función `print` para imprimir cada elemento de la misma. El orden en el que deben imprimirse los valores es el resultante de recorrer la matriz de a columnas.

Notas

- *La matriz se encuentra almacenada a partir de la posición `0x1234` como un arreglo de filas contiguas.*
- *La función `print` toma como parámetro un único número a imprimir en el registro `R0`, y no modifica ningún registro.*

Ejercicio 2: Solución

```
imprimirMatriz: MOV R1, 0
                MOV R3, 0
                MOV R2, 8*8
          loop: CMP R2, 0
                JE fin
                MOV R0, [R3 + 0x1234]
                CALL print
                ADD R3, 8
                CMP R3, 8*8
                JL sigue
                SUB R3, 8*8-1
         sigue: SUB R2, 1
                JMP loop
           fin: RET
```

ORGA1s

Se tiene una máquina cuya arquitectura es una extensión de ORGA1, llamada ORGA1s, la cual incorpora *shifters* hacia la izquierda y hacia derecha en el conjunto de instrucciones. Las descripción de las nuevas instrucciones es la siguiente:

- `SHL destino, c5`: modifica destino shifteando sin signo `c5` bits a la izquierda su valor original (si $0 \leq$ `c5` ≤ 16; el resultado es 0 en otro caso).
- `SHR destino, c5`: modifica destino shifteando sin signo `c5` bits a la derecha su valor original (si $0 \leq$ `c5` ≤ 16; el resultado es 0 en otro caso).

En ambos casos, `destino` es cualquiera de los modos de direccionamiento soportados por la arquitectura ORGA1.

Ambas instrucciones modifican las flags `Z`, `N` y `C`. En el caso de la flag `C`, su valor es el del último bit que se "cae" del número.

Ejercicio 3

Implementar en assembler de ORGA1s los siguientes programas:

a) `contarUnos`: devolver en `R0` la cantidad de bits encendidos del número indicado por `R1`. ¿Es posible hacer un programa en ORGA1 que resuelva el mismo problema?

b) `espejar`: devolver en `R0` un número cuyo valor sea el resultado de espejar bit a bit `R1`. Por ejemplo:
`1000 0000 0000 0000` $\rightarrow$ `0000 0000 0000 0001`
`0001 1101 1100 1111` $\rightarrow$ `1111 0011 1011 1000`

Ejercicio 3a: Solución (ORGA1s)

```
int contarUnos(int v) {
    int c = 0;
    while (v) { c += (v & 1); v >>= 1; }
    return c;
}
```

```
contarUnos: MOV R0, 0
      loop: CMP R1, 0
            JE fin
            MOV R2, R1
            AND R2, 1
            ADD R0, R2
            SHR R1, 1
            JMP loop
       fin: RET
```

Ejercicio 3a: Solución (ORGA1)

```
int contarUnos(int v) {
    int c = 0;
    while (v) {
        c += (v & 0x8000) ? 1 : 0;
        v <<= 1;
    }
    return c;
}
```

```
contarUnos: MOV R0, 0
      loop: CMP R1, 0
            JE fin
            ADD R1, R1 ; SHL R1, 1
            ADDC R0, 0 ; si C, MSB(R1) era 1
            JMP loop
       fin: RET
```

Ejercicio 3a: Solución (Kernighan)

```
int contarUnos(int v) {
        int c = 0;
        while (v) { v &= (v-1); c++; }
        return c;
}
```

```
contarUnos: MOV R0, 0
      loop: CMP R1, 0
            JE fin
            MOV R2, R1
            SUB R2, 1
            AND R1, R2
            ADD R0, 1
            JMP loop
       fin: RET
```

Ejercicio 3b: Solución (ORGA1s)

```
int espejar(int n) {
    int ret = 0, count = 16;
    while (count) {
        ret <<= 1; ret |= (n & 1); n >>= 1; count--;
    }
    return ret;
}
```

```
espejar: MOV R0, 0 ; ret = 0
         MOV R2, 16 ; count = 16
   loop: CMP R2, 0 ; guarda del while
         JZ fin
         SHL R0, 1 ; ret <<= 1
         MOV R3, R1 ; r3 = n
         AND R3, 1 ; r3 = (n & 1)
         OR R0, R3 ; ret |= (n & 1)
         SHR R1, 1 ; n >>= 1
         SUB R2, 1
         JMP loop
    fin: RET
```

Ejercicio 3b: Solución (SWAR)

```
espejar: MOV R1, R0
         MOV R2, R0
         AND R1, 0x5555
         SHL R1, 1
         AND R2, 0xAAAA
         SHR R2, 1
         OR R1, R2
         MOV R0, R1 ; n = ((n & 0x5555) << 1) | ((n & 0xAAAA) >> 1);
         MOV R2, R1
         AND R1, 0x3333
         SHL R1, 2
         AND R2, 0xCCCC
         SHR R2, 2
         OR R1, R2
         MOV R0, R1 ; n = ((n & 0x3333) << 2) | ((n & 0xCCCC) >> 2);
         MOV R2, R0
         AND R1, 0x0F0F
         SHL R1, 4
         AND R2, 0xF0F0
         SHR R2, 4
         OR R1, R2
         MOV R0, R1 ; n = ((n & 0x0F0F) << 4) | ((n & 0xF0F0) >> 4);
         MOV R2, R0
         AND R1, 0x00FF
         SHL R1, 8
         AND R2, 0xFF00
         SHR R2, 8
         OR R1, R2
         MOV R0, R1 ; n = ((n & 0x00FF) << 8) | ((n & 0xFF00) >> 8);
         RET
```