ORGANIZACION DE COMPUTADORAS
PARCIAL 2 2014

1. En el marco de la norma IEEE 754, considerando la representación en punto flotante de media presición: mantisa fraccionaria en signo magnitud con hidden bit, exponente en exceso y base 2 y la siguiente distribución de bits:

Sig (1b)	Exponente(5bits)	Mantisa(10bits)

Dados los números:

X= (1 10110 0101110011) Y= (0 00111 0011011001)

Realizar el producto X x Y aplicando redondeo hacia -∞, explicando cada uno de los pasos involucrados e indicando claramente que se hace con los bits G, R y S del resultado y con R y S al redondear. El resultado debe ser expresado según la representación enunciada.

2. En el marco de la norma IEEE 754, considerando la representación en punto flotante de simple presición: mantisa fraccionaria en signo magnitud con hidden bit, exponente en exceso y base 2 y la siguiente distribución de bits:

Sig (1b)	Exponente(8bits)	Mantisa(15bits)

Dados los números:

X= (1 01111010 001110011001110) Y= (1 10000011 101001100111101)

Realizar la suma X + Y aplicando redondeo por proximidad unbiased(hacia los pares), explicando cada uno de los pasos involucrados e indicando claramente que se hace con los bits G, R y S del resultado y con R y S al redondear. El resultado debe ser expresado según la representación enunciada.

3. Asumiendo que se cuenta en todos los casos con las instrucciones add y mpy. Encontrar una secuencia de instrucciones que resulten óptimas en tiempo de ejecución(que minimice la cantidad de accesos a memoria) y cuya ejecución tenga como resultado la evaluación de la siguiente expresión aritmética:

$$B= D \times C + A \times A + B \times D \times C$$

Las etiquetas denotan las direcciones que contienen los valores sobre los que se quiere operar.

1. Asumiendo una arquitectura de 0-direcciones(tipo PILA), con las instrucciones push y pop para acceder a memoria y la instrucción dup que duplica el tope de la pila.
2. Asumiendo una arquitectura tipo INTEL con operaciones 1-dirección mas registros, sin limitaciones en cuanto a los registros disponibles, que cuenta con la instrucción

mov para acceder a memoria y donde las operaciones aritméticas operan con tres operandos(dst,fte,fte).

3. Asumiendo una arquitectura de tipo VAX con operaciones que admiten hasta 3 direcciones de memoria, donde las operaciones aritméticas operan con tres operandos (dst,fte,fte).

4. Para cada una de las implementaciones de los tres incisos anteriores, determinar:
   -el numero de instrucciones
   -el numero de accesos a memoria realizados por lectura/escritura de datos
   -el espacio en memoria ocupado:
      -las instrucciones tipo PILA ocupan 3 bytes las de 1 dirección, y 1 byte las de 0 direcciones.
      -las de tipo INTEL ocupan 2 bytes las de 0 direcciones(solo registros), 4 bytes las de 1 dirección(sin registros) y 6 bytes las de 1 dirección mas registros.
      -las de tipo VAS ocupan 1 byte las de 0 direcciones, 2 bytes las de 1-dirección, 3 bytes las de 2 direcciones, y 4 bytes las de 3 direcciones.

1. Considerando el siguiente programa para la arquitectura OCUNS, en la que toda lectura/escritura en la dirección FFh es atrapada y redireccionada a la entrada/salida estándar y el registro RF está cableado a 0:

```
        lda     RA, FF
        load    R5, 0(RA)
        load    R6, 0(RA)
        add     R7, R5, R6
        dec     R7
        call    R3, rut
        hlt
rut:    sub     R2, R5, R7
        jg      R2, fin
        load    R0, 0(R5)
        load    R1, 0(R7)
        sub     R2, R1, R0
        jg      R2, lbl
        store   R1, 0(R5)
        store   R0, 0(R7)
lbl:    inc     R5
        dec     R7
        jz      RF, rut
fin:    jmp     R3
```

OPCODE	DESCRIPCION	FORMATO	PSEUDOCÓDIGO
0	add (add)	**I**	R[d] ← R[s] + R[t]
1	sub (subtract)	**I**	R[d] ← R[s] – R[t]
2	and (and)	**I**	R[d] ← R[s] & R[t]
3	xor (xor)	**I**	R[d] ← R[s] ^ R[t]
4	lsh (left shift)	**I**	R[d] ← R[s] « R[t]
5	rsh (right shift)	**I**	R[d] ← R[s] » R[t]
6	load (load)	**I**	R[d] ← mem[offset + R[s]]
7	store (store)	**I**	mem[offset + R[d]] ← R[s]
8	lda (load address)	**II**	R[d] ← addr
9	jz (jump zero)	**II**	if (R[d] == 0) PC ← PC + addr
A	jg (jump greater)	**II**	if (R[d] > 0) PC ← PC + addr
B	call (jump and link)	**II**	R[d] ← PC; PC ← addr
C	jump (jump register)	**III**	PC ← R[d]
D	inc (increment)	**III**	R[d] ← R[d] + 1
E	dec (decrement)	**III**	R[d] ← R[d] – 1
F	hlt (halt)	**III**	exit

FORMATO	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
I	0	x	x	x	dest.d				src.s				src.t/offset			
II	1	0	x	x	dest.d				address addr							
III	1	1	x	x	dest.d				-							

1. Ensamblar el programa a partir de la direccion 00h.
2. Indicar cómo recibe los argumentos de entrada rut y que consideraciones debe tener en cuanto a registros el programa que la utilice.
3. Describir claramente que hace el programa. A partir del ensamblado del inciso 1, realizar la traza considerando que se ingresan por teclado, en el orden dado, los números 00h y 0Ch. Indicar cuántas veces se realiza un swap.
4. Reubicar el código máquina obtenido en el inciso 1 a partir de la direccion A0h. Justificar adecuadamente cuáles de las referencias a memoria requieren ser ajustadas y cuáles no.

5. Considerando la arquitectura OCUNS descripta en el ejercicio anterior, indicar una secuencia de instrucciones de dicha arquitectura equivalente a cada uno de los siguientes códigos:

a.
```
If (RA > 5) R3++;
else R3+= R5;
```

b.
```
While (RA > 0){
    R3+= RA;
    RA--;
}
```

c.
```
do{
    R3 -= RA;
    RA++;
}while( R3 < 100);
```