Ejercicio 1:

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

$$B = (A \times (C+D)\ ) + (A\ x\ (C + D))^3$$

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

a) 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. Determinar la cantidad de instrucciones y la profundidad de la pila alcanzada.

b) Asumiendo una arquitectura estilo RISC con operaciones registro a registro, sin limitaciones en cuanto a los registros disponibles, y las instrucciones ld (load) y st (store) para acceder a memoria, y la instrucción lda (load address). Las operaciones aritméticas operan con dos operandos (dst/fte , fte). Indicar la cantidad de accesos a memoria requeridos.

c) Asumiendo una arquitectura tipo INTEL con operaciones 1–dirección más registros, sin limitaciones en cuando a los registros disponibles, que en lugar de load y store cuenta con la instrucción mov para acceder a memoria y donde las operaciones aritméticas operan con tres operandos (dst, fte, fte). Indicar la cantidad de accesos a memoria realizados.

a) Pila

Inst 1: PUSH A
Inst 2: PUSH C
Inst 3: PUSH D
Inst 4: ADD
Inst 5: MPY
Inst 6: DUP
Inst 7: DUP
Inst 8: DUP
Inst 9: MPY
Inst 10: MPY
Inst 11: ADD
Inst 12: POP B

Cantidad de instrucciones: 12
Profundidad de la pila alcanzada: 4

Pila 1: inst 1 a 3	Pila 2. Inst 4
A	A
C	C+D
D	
Pila 3: Inst 5	**Pila 4: Inst 6**
A*(C+D)	A*(C+D)
	A*(C+D)
Pila 4: Inst 7 y 8	**Pila 5: Inst 9**
A*(C+D)	A*(C+D)
A*(C+D)	A*(C+D)
A*(C+D)	(A*(C+D))^2
A*(C+D)	

	Pila 6: Inst 10 A*(C+D) (A*(C+D))^3) Pila 7: Inst 11 (A*(C+D))+ (A*(C+D))^3
b) RISC registro a registro 1: LDA R0,C 2: LD R1,(R0) 3: LDA R0,D 4: LD R2,(R0) 5: LDA R0,A 6: LD R3,(R0) 7: ADD R4,R1,R2 8: MUL R5,R4,R3 9: MUL R6,R5,R5 10: MUL R6,R6,R5 11: ADD R7,R6,R5 12: LDA R8, B 13: ST (R8),R7 Cantidad de accesos a memoria: 4	R1 <- C R2 <- D R3 <- A R4 <- C+D R5 <- A*(C+D) R6 <- (A*(C+D))^3 R7 <- (A*(C+D)) + (A*(C+D))^3 B <- R7
c) INTEL 1 dir + reg 1: mov R0,[C] 2: ADD R0,[D] 3: MUL R0,[A] 4: MOV R1,R0 5: MUL R0,R0 6: MUL R0,R1 7: ADD R0,R1 8: MOV [B],R0	Cantidad de instrucciones: 8 Accesos a memoria: 4

Ejercicio Nº 2:

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

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

Sig (1b)	Exponente (8 bits)	Mantisa (10 bits)

Dados los números:
X = (1 10000100 0011111001) = -1 x 2^5 x 1.2431640625 = -39,78125
Y = (0 01110101 1000111100) = 1 x 2^{-10} x 1.55859375 = 0.0015220642090
$X*Y$ = -39,78125 * 0.0015220642090 = 0.060549617

Realizar el producto $X \times Y$ aplicando redondeo hacia $+\infty$ y proximidad pares, explicando cada uno de los pasos involucrados e indicando claramente qué se hace con los bits **G, R y S** del resultado y **con R y S** al redondear. El resultado debe ser expresando según la representación enunciada.
Solución:

X = (-1) * $^{(132-127)}$ * 1,0011111001

Y = (1) * $2^{117-127}$ * 1,1000111100

Sumo exponentes:
132 +117 - 127 = 122

= 0111 1010

Multiplicamos mantisas:

										1	0	0	1	1	1	1	1	0	0	1
										1	1	0	0	0	1	1	1	1	0	0
									0^{11}	0^{11}	0^{10}	0	0	0	0	0	0	0	0	0
								1^{10}	0	0	0	1	1	1	1	0	0	0	1	
							1^{10}	0	0	1	1	1	1	1	0	0	0	1		
						1^{10}	0	0	1	1	0	1	1	1	0	0	1			
					0^{10}	0	0	0	0	0	0	0	0	0	0	1				
				0^1	0	0	0	0	0	0	0	0	0	0	0					
			0	0	1	1	1	1	1	0	0	1								
		1	0	0	1	1	1	1	0	0	1									
1																				
1,	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0

1,	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0

G = 0, R = 0, S = 1
Está normalizado.
1,1111000000 => R = 0 S = 1.

Redondeo $+\infty$ R = 0 S = 1
+1 LSB

	1, 1 1 1 1 0 0 0 0 0 1 Sig (1b) | Exponente (8 bits) | Mantisa (10 bits) 1 | 01111001 | 1111000001 <u>Redondeo proximidad pares</u>: R = 0 S = 1 (no hay cambio) 1, 1 1 1 1 0 0 0 0 0 0 Resultado: Sig (1b) | Exponente (8 bits) | Mantisa (10 bits) 1 | 01111001 | 1111000000
	Resultado decimal: -39,78125 * 0.0015220642090 = - 0.060549617 Resultado Norma: $-1 \times 2^{-6} \times 1.9375 = - 0{,}0302734375$ Error: 0.060549617 - 0,0302734375 = 0,0302761795

Ejercicio 3:

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

Sig (1b)	Exponente (8 bits)	Mantisa (10 bits)

Dados los números:
X = (0 01111100 0010110101) = 1 x 2^{-3} x 1.1767578125 = 0,147094727
Y = (1 01111101 1101000110) = -1 x 2^{-2} x 1.818359375 = 0,454589844

Realizar la suma *X + Y* aplicando redondeo por proximidad, explicando cada uno de los pasos involucrados e indicando claramente qué se hace con los bits **G, R y S** del resultado y **con R y S** al redondear. El resultado debe ser expresando según la representación enunciada.
Solución:

Igualar Exponentes X = 1 * 2^{-3}* 1.0010110101 >> 1 X = 1 * 2^{-2}* 0.10010110101 Y = (-1) * 2^{-2} * 1.1101000110	
Complementar y Y=01.1101000110 10.0010111001 + 1 . 10.0010111010	
Sumar mantisas X 00.1001011010|1 Y 10.0010111010| . 10.1100010100|0 0 0 G R S Complemento: 01.0011101011 + 1 Resultado = 01.0011101100 Esta normalizado 1.0011101110|0 0 0 Valores R = 0, S = 0. Redondeo proximidad unbiased (pares): p_0=0 R=0 S=0 No sumar nada. Resultado final: 1 | 01111101 | 0011101110	Es negativo, con lo cual hay que complementar.

Suma en decimal: 0,147094727 - 0,454589844 = -0,307495117 Suma IEEE (decimal) : 2^{-2} x 1.23046875 = -0,3076171875 Error: 0,0001220705	

Ejercicio Nº 4:

(1) mov R1,#0200	#xxxx Inmediato		***R1***	***R2***	***R3***	***M[200]***	***M[300]***
(2) mov (R1), #0100	R Registro	***1***	**200**	-	-	-	-
(3) mov 0100(R1), R1	(R) Registro indirecto	***2***	200	-	-	**100**	-
(4) mov R2, #0500	xxxx Absoluto	***3***	200	-	-	100	**200**
(5) mov @0100(R1), #0500	xxxx(R) Indexado	***4***	200	**500**	-	100	200
(6) mov (0200), 0300	(xxxx) Memoria indirecto	***5***	200	500	-	**500**	200
(7) mov R3, 0200	@xxxx(R) Pre-indexado indirecto	***6***	200	500	-	**300**	200
(8) mov R3, @0100(R3)		***7***	200	500	**200**	300	200
		8	200	500	**300**	300	200

Ejercicio Nº 5:

Programa	***Inciso A: Ensamblado***		***Inciso B:***
LDA R0, FFh	00:	80FF	Al tener que reubicar el código máquina, se deberán ajustar todas las referencias que direccionen de forma directa a posiciones de memoria, no siendo así las referencias que direccionen desplazamientos relativos dentro del programa. Luego, se deberá ajustar la dirección de la instrucción (LDA R4, lbl3; 8420) por (LDA R4, lbl3; 8440).
LOAD R1, 0(R0)	02:	6100	
LOAD R2, 0(R0)	04:	6200	
XOR R3, R3, R3	06:	3333	
LDA R4, lbl3	08:	8420	
JZ R1, lbl3	0A:	9114	
JZ R2, lbl3	0C:	9212	
SUB R5, R1, R2	0E:	1512	
JG R5, lbl2	10:	A508	
lbl1: ADD R3, R3, R2	12:	0332	
DEC R1	14:	E1XX	
JG R1, lbl1	16:	A1FA	
JMP R4	18:	C4XX	
lbl2: ADD R3, R3, R1	1A:	0331	
DEC R2	1C:	E2XX	
JG R2, lbl2	1E:	A2FA	
lbl3: STORE R3, 0(R0)	20:	7030	
HLT	22:	FXXX	

Inciso C:

R0	***R1***	***R2***	***R3***	***R4***	***R5***	***PC***
--	--	--	--	--	--	**00**
FF	--	--	--	--	--	**02**
FF	**04**	--	--	--	--	**04**
FF	04	**02**	--	--	--	**06**
FF	04	02	0	--	--	**08**
FF	04	02	0	20	--	**0A**
FF	04	02	0	20	--	**0C**
FF	04	02	0	20	--	**0E**
FF	04	02	0	20	**02**	**10**
FF	04	02	0	20	02	**12-1A**
FF	04	02	**04**	20	02	**1C**
FF	04	01	04	20	02	**1E**
FF	04	01	04	20	02	**20-1A**
FF	04	01	**08**	20	02	**1C**
FF	04	00	08	20	02	**1E**
FF	04	00	08	20	02	**20**
FF	04	00	08	20	02	**22**
FF	04	00	08	20	02	**--**

El programa lee por consola dos valores A y B, y retorna el resultado de la multiplicación A*B.

Inciso D:

El orden de los operandos A y B no cambia el resultado: el programa se encarga de realizar X sucesivas sumas de un valor Y; el valor de X será el MIN(A,B) y el valor de Y=MAX(A,B) para realizar la menor cantidad de sumas posibles, sin alterar el resultado.

La restricción en cuanto a los valores de entrada es que ambos sean positivos. Si se ingresan valores negativos, el programa se comporta de forma anómala (bucle infinito hasta señalización de overflow).