Evaluación de aprendizaje #2

Arquitectura de Computadoras - UNLaM

Aclaraciones

Los siguientes puntos deben ser resueltos con programas escritos en ASM para el 68HC11. Los mismos deben poder ser ensamblados con el simulador THRSIM11. Se considera **grave** el entregar programas que generen errores de ensamblado.

Por cada punto del examen se debe entregar un archivo .asm con el código correspondiente. El formato del nombre del archivo será:
`EA2-COMISION-DNI-ApellidoNombre-PuntoX.asm`.
Ejemplo: Alumno Carlos Maidana (comisión 2) con DNI 12.345.678 entrega el punto 1 como:
`EA2-C2-12345678-MaidanaCarlos-Punto1.asm`
El conjunto de archivos .ASM debe ser entregado por MIeL en un archivo .zip con el siguiente formato de nombre:
`EA2-C2-12345678-MaidanaCarlos.zip`

Cada programa debe tener comentarios que sean suficientes como para describir lo que realiza el programa. Mientras mejor comentado se encuentre, más fácil es para el docente entender lo que hicieron. Si el docente no puede comprender algo entonces está mal.

En algunos puntos se brindan valores de prueba como ejemplo, pero sus programas serán probados con diversos lotes de prueba. El mismo debe funcionar para cualquier valor posible, no solo para los valores de prueba dados como ejemplo. En el caso particular de algún valor problemático haga las aclaraciones pertinentes.

En algunos puntos se entregan esqueletos de programa (plantillas a ser completadas). Respete las direcciones y etiquetas de estos.
Siéntase libre de emplear subrutinas para resolver los ejercicios. Por favor comente en un par de líneas el objetivo de la subrutina y la forma en que está pasando los parámetros y devolviendo valores. Por ejemplo, si emplea registros, la pila, etc.

El último ejercicio requiere que aprendan un algoritmo nuevo y lo implementen en ASM.

En cada punto hay diversos casos que cambian los elementos del ejercicio. Busque los valores que le corresponden a usted en la tabla publicada e indique la opción resuelta en un comentario al comienzo de cada programa.

1- Escribir una subrutina que compare dos números (N1 y N2) . La cantidad de bytes que ocupa cada número se informa por el registro B cuando se llama a la subrutina (como mínimo 2 bytes). Las direcciones de memoria de comienzo de cada número se informan por los registros índices X=N1 e Y=N2. El resultado de la comparación se informa mediante el registro A, con el cual se indica:

- **A = 0** si **N1==N2**
- **A > 0** si **N1 > N2**
- **A < 0** si **N1 < N2**

Los números N1 y N2 pueden tener los siguientes formatos:

- **Caso A**: Ambos son signados (negativos en CB) en formato big endian
- **Caso B**: Ambos son sin signo en formato big endian
- **Caso C**: Ambos son signados (negativos en CB) en formato little endian
- **Caso D**: Ambos son sin signo en formato little endian

Se provee a continuación el esqueleto del programa, incluyendo el programa principal, además de datos de ejemplo:

```
;Mi caso es el: (complete el caso asignado)
            ORG $0000
;;Reserve memoria de ser necesario
            ORG $C000
Main
            LDX   #N1
            LDY   #N2
            LDAB  #3
            JSR   ComparaN
FIN         BRA   FIN

N1          DB    $32,$34,$BC
N2          DB    $98,$6D,$55

ComparaN
;;Complete la subrutina

            ORG   $FFFE
            DW    Main
```

2- Dados dos valores (NUMA y NUMB), debe escribir un programa que realice la suma de ambos números y almacene el resultado en RES teniendo en cuenta

- **Caso A**: NUMA y NUMB son números signados (Negativos en CB) de 16 bits
- **Caso B**: NUMA y NUMB son números sin signo de 16 bits pero están almacenados en formato little endian. Se espera que RES esté almacenado en formato big endian.
- **Caso C**: NUMA y NUMB son números sin signo de 24 bits.
- **Caso D**: NUMA y NUMB son números sin signo de 32 bits.

Se provee a continuación el esqueleto del programa para completar. Tenga en cuenta que los valores provistos de NUMA y NUMB son ejemplos, el programa debe funcionar para cualquier valor.

```
;Mi caso es el: (complete el caso asignado)
                ORG $0000
RES             RMB
;;Complete el valor de RES y declare las variables que considere necesarias

                ORG $C000
Main
;;Comience su programa aquí abajo

;;Descomente el caso que corresponda
;Caso A
;NUMA            DB  $30, $45
;NUMB            DB  $92, $FA

;Caso B
;NUMA            DB $32, $80
;NUMB            DB $45, $C8

;Caso C
;NUMA            DB $23,$67,$89
;NUMB            DB $DF,$45,$AE

;Caso D
;NUMA            DB $32,$E6,$F1,$00
;NUMB            DB $45,$78,$CB,$DB

                ORG  $FFFE
                DW Main
```

3- Dado el vector **DESORDENADO** que contiene:

- **Caso A:** Elementos de 8 bits sin signo
- **Caso B:** Elementos de 8 bits signados (Negativos en CB)
- **Caso C:** Elementos de 16 bits sin signo
- **Caso D:** Elementos de 16 bits signados (Negativos en CB)

Genere un nuevo vector ORDENADO el cual copie los elementos del vector DESORDENADO en orden:

- **Caso A:** Menor elemento en la mitad del vector, ascendiendo hacia el final. Cuando llega al final continúa por el primer elemento del vector hasta llegar a la mitad. Esto es Ordenamiento circular ascendente.
- **Caso B:** Mayor elemento en la mitad del vector, descendiendo hacia el final. Cuando llega al final continúa por el primer elemento del vector hasta llegar a la mitad. Esto es Ordenamiento circular descendente.
- **Caso C:** De menor a mayor comenzando por el tope del vector.
- **Caso D:** De mayor a menor comenzando por el tope del vector.

Se provee a continuación el esqueleto del programa para completar. Se proveen valores como ejemplo para DESORDENADO, pero tenga en cuenta que la verificación del examen se hará con otros valores de prueba. Esto quiere decir que su programa debe funcionar para cualquier valor dentro del caso asignado. El tamaño del vector estará en todos los casos limitado a 128 bytes.
NOTA: El programa debe estar debidamente comentado para alguien que tiene 5 minutos para ver su programa y debe entender como funciona.

```
;Mi caso es el: (complete el caso asignado)
                ORG   $0000
ORDENADO        RMB   32
;; Puede declarar variables que considere necesarias

                ORG   $C000
Main
;; Comienza el programa:


DESORDENADO     DB    $01,$3F,$80,23…

                ORG   $FFFE
                DW    Main
```

4- Dado un vector de elementos de 8 bits del que conoce la dirección de inicio y la cantidad de elementos, siendo la cantidad siempre menor o igual a 255, genere el código ensamblador tal que cuente los siguientes elementos dentro del vector, según corresponda a su caso:

DNI terminado en	Ocurrencias y nombre de la reserva para indicarlo
0	Ceros (cero), múltiplos de 4 (mul4), códigos ascii de letras minúsculas (min)
1	Múltiplos de 2 (mul2), códigos ascii de letras mayúsculas (may), impares (impar)
2	Pares (par), Negativos en Cb(negcb), códigos ascii de letras minúsculas (min)
3	Ceros (cero), códigos ascii de letras mayúsculas (may), impares (impar)
4	Múltiplos de 8 (mul8), impares (impar),códigos ascii de letras minúsculas (min)
5	Pares (par), múltiplos de 8 (mul8), Negativos en Cb (negcb)
6	Negativos en Cb (negcb), impares (impar), códigos ascii de letras mayúsculas (may)
7	Ceros (cero), códigos ascii de letras minúsculas (min), Negativos en MyS (negms)
8	Negativos en Cb (negcb), múltiplos de 4 (mul4), códigos ascii números (min).
9	Impares (impar), códigos ascii de letras mayúsculas (may), Negativos en MyS (negms)

```
;Mi DNI termina en: (complete)
                ORG $0000
DIRINI          EQU   [complete]
CANT            RMB   [complete]
[complete con las reservas para cada consigna segun el caso]

;;Declare las variables que crea necesarias

                ORG $C000
Main
;;Complete su programa aqui

                ORG $DIRINI
VECTOR          DB ….. [genere un juego de prueba significativo]
                ORG $FFFE
                DW  Main
```

5- Dado un vector que puede contener hasta 16 elementos

- **Caso A**: cada elemento de 8 bits signado con negativos en CB
- **Caso B**: cada elemento de 8 bits signado en módulo y signo
- **Caso C**: cada elemento de 16 bits signado con negativos en CB
- **Caso D**: cada elemento de 16 bits signado en módulo y signo

Almacene los mismos en un nuevo vector de forma tal que si el elemento original era de 8 bits, el mismo ocupe 16 bits en el nuevo vector. Si el elemento original era de 16 bits, el mismo será de 32 bits en el nuevo vector.
A continuación el esqueleto del programa:

```
;Mi caso es el: (complete el caso asignado)
                ORG $0000
NUEVOVECTOR     RMB
;;Complete el tamaño del nuevo vector
;;Declare las variables que crea necesarias

                ORG $C000
Main
;;Complete su programa aqui


VECTOR          DB $80,$00,$CA, $7F, $7F
                ORG $FFFE
                DW  Main
```

6- En una sección de la memoria cuya dirección inicial es MENSAJE, se ha almacenado una cadena de caracteres de índole confidencial codificados en ASCII (sólo letras mayúsculas). La cadena concluye con un carácter no-ascii, el valor 0x0 (cero). Se solicita aplique a la cadena un cifrado y la almacene a partir de otra posición de memoria (CRYPTO). La cadena cifrada debe concluir con el valor 0xFF. Almacene y utilice la clave (keyword) en KEY.

Caso A: Aplique cifrado César, con una key=[último dígito del DNI]
Caso B: Aplique cifrado Atbash (https://en.wikipedia.org/wiki/Atbash) utilizando el siguiente alfabeto:

Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Caso C: Aplique cifrado Vigenère (La keyword empleada deben ser las 3 iniciales del alumno, en caso de no tener 3, agregar una A como segunda inicial).

```
;Mi caso es el: (complete el caso asignado)
                ORG $0000

;;Declare las variables que crea necesarias
CRYPTO          RMB   [XX]    * complete la cantidad
KEY             RMB   [X]     * complete según necesidad y el caso

                ORG $C000
Main
;;Complete su programa aqui

MENSAJE         FCC “HABLEMASFUERTEQUETENGOUNATOALLA”
FINMENSAJE      DB 0

                ORG $FFFE
                DW  Main
```