Final 30-07-2018

Ejercicio 1

(Item a) Cuando tenemos una instruccion a ejecutarse, esta se procesa y se ejecuta todo el microcodigo relacionado a la operacion que estemos queriendo realizar. Para cada microinstruccion estan los campos i, imux (siendo i= {A,B}). Estos campos de la microinstruccion, se van a conectar con el multiplexorA y el multiplexorB respectivamente. Dicho multiplexor va a tener como entradas, en el caso de A, el rs1 del IR, el campo A del MIR y el campo Amux del MIR que sera quien indique si el muxA tiene que elegir la direccion 0-rs1 o la del campo A del MIR. Similar sucede en el caso del B, dicho multiplexor va a tener como entradas, el rs2 del IR, el campo B del MIR y el campo Bmux del MIR que sera quien indique si el muxB tiene que elegir la direccion 0-rs2 o la del campo B del MIR. Y la salida de estos multiplexores le indicaran a cada uno de los decodificadores, A y B respectivamente, el contenido de que registro debe ser copiada a los buses A y B respectivamente. Siendo los buses A y B las entradas de la ALU desde los registros. Es por eso que la ALU solo ve el contenido de uno de todos los registros conectados al BUS.

(Item c)
la c puse que es necesario que este incluida porque el microcodigo de la instruccion 2047 incrementa el program counter, al hacer eso el program counter es capas de realizar el seguimiento y ejecucion de las intrucciones del program.

Ejercicio 2

```
.begin
.org 2048

.macro push arg
        add %r14,-4,%r14
        st arg,%r14
.endmacro

.macro pop arg
        ld %r14, arg
        add %r14,4,%r14
.endmacro
add %r15,0,%r16 !back up del r15 por las dudas

pop %r1 !direccion de inicio del array
pop %r2 !largo del array (palabras de 4bytes)
add %r2,%r0,%r7 !back up del largo del array

A .equ D1000h
B .equ 160h
sethi A,%r3
sll %r3,2,%r3
add %r3,B,%r3 !almacena la direccion del periferico

mascara .equ 300000h
sethi mascara,%r4       !r4 tiene la mascara
```

```
add %r0,%r0,%r20        !r20 me acumulara la cantidad de elementos alterados
add %r27,1,%r27 !respuesta uno de la sub

recorrer_array:         andcc %r2,%r2,%r0
                        be fin
                        add %r2,%r1,%r5 !r3 es la pos dentro del array
                        ld %r5,%r6              !r6 almacena el contenido del
array
                        push %r16
                        add %r15,0,%r17 !back up r15 para salir del
recorrer_array
                        call verificar_condicion
                        pop %r10        !devuelve de la sub 1 o 0
                        addcc %r10,-1,%r0
                        be poner_en_cero
                        add %r2,-4,%r2
                        ba recorrer_array
andcc %r20,%r0,%r0
be copiar_valor_periferico

halt

!labels & subrutinas

verificar_condicion:    pop %r11
                        andcc   %r11,%r4,%r0
                        be no_son_1
                        push %r27
                        jmpl %r15,4,%r0

no_son_1:       push %r0
                !jmpl %r15,4,%r0

poner_en_cero:  and %r0,%r6,%r6
                        add %r20,1,%r20
                        !jmpl %r15,4,%r15

copiar_valor_periferico:        ld %r3,%r26
                                add %r7,%r1,%r18
                                ld %r18,%r19
                                st %r26,%r19
                                !jmpl %r15,4,%r0

fin:    jmpl %r16,4,%r0
.end
```

Ejercicio 3
(Item a)

```
.org 5000 !(directiva q no ocupa memoria)
.dwb 5000 !(5000x32)/8=20000 Bytes
V .equ 5000 !(directiva q no ocupa memoria)
```

```
add %r1,V,%r1 !4 bytes
or %r1,4,%r1 !4bytes
x:5000  !direccion q no ocupa memoria
```

En total se usan 20000+4+4=20008 bytes de RAM.

(Item b)
La necesidad de que un programa ensamblador guarde junto con el codigo de maquina, una tabla de simbolos donde indica si cada simbolo es o no relocalizable, ya que es responsabilidad del ensamblador indicar que simbolos son reubicales. En cualquier caso, la informacon de reubicacion se inclye e un diccionario de reubicacion, en el moludo de ensmablado, para que sea utilzizadp por en linker y/o el loader.
Ya que el linker tiene la tarea de combinar programas ensamblados por separado en un unico programa, y para ello debe resolver todas las refrencias globales y externas, y reubicar las direcciones de los diferentes modulos.
El programa unificado puede ser cargado en memoria por medio de un loader, que tambien puede necesitar modificar direcciones si el programa se carga en una direccion distinta de origen de carga usada por el linker. Por estos motivos, es fundamental saber si los simbolos pueden o no ser relocalizables.

Ejercicio 4
(Item A)
La necesidad de cache en los procesadores modernos se debe a que esta le brinda una mayor velocidad. Esto se debe a que en memoria cache se copian los bloques de memoria RAM mas utilizados por el programa (por el principio de localidad, que se divide en espacial: insinuando que estadisticamente si accedes a una direccion de memoria es mucho mas probable que proximamente accedas a una cercana que a una lejana y temporal: que si accedemos a una direccion de memoria, es muy probable que en poco tiempo volvamos a acceder a la misma).
Acceder desde el microprocesador a caché es mas veloz que acceder a RAM, el acceso a la memoria principal es muy lento. Por lo tanto con cache, la velocidad de operacion resulta mayor, y ademas, teniendo en cuenta cache+principio de localidad, el rendimiento de nuestros procesadores va a ser mayor.

(Item B)
El mapa de memoria es una estructura de datos que indica como esta distribuida la memoria.Brindandonos informacion del tamaño de la memoria y de los respecivos espacios. Es decir, de cuanta memoria (o que direcciones de memoria) se destina(n) al sistema operativo, cuanto (o que direcciones de memoria) se destina(n) a los programas ensamblados por el usuario, cuanto (o que direcciones de memoria) se destina(n) a la pila, y por ultimo cuanto (o que direcciones de memoria) se destina(n) a los perifericos de entrada y salida.