Final 30-07-2018 Estructura del Computador

1.

a) Respecto de la micro arquitectura ARC explique detalladamente de qué manera se logra que cada entrada de la ALU vea el contenido de solamente uno de todos los registros conectados al bus.
b) Respecto del bloque conocido como lógica de control de saltos Explique su lógica de funcionamiento, entradas, salidas y vínculos con otros componentes de la micro arquitectura.
c) El micro código del procesador ARC contempla en la posición 2047 una microinstrucción que incrementa el program counter justifique la necesidad de su inclusión como parte del micro código y refiera las instrucciones de Assembler cuya ejecución no requería de esa posición de memoria de control

2. Un programa recibe por stack la dirección de inicio y el largo de un array (el largo es medido en palabreas de 4 bytes).Su función es poner a cero todos los elementos de ese array cuyos dos bits más significativos son "1" dejando el resto de los elementos de ese array sin alterar. En caso de que ningún elemento de ese arreglo cumpla esa condición el programa escribe el valor D1000160h en el último de los elementos. La condición citada es evaluada por una rutina a la cual se le pasa por stack el valor a evaluar devolviendo un 1 si la condición se verifica a cero en caso contrario. Esta debe ser declarada en el mismo modulo.

3.

a) Indique la cantidad de memoria RAM ocupa por cada una de las siguientes líneas de código:

```
.org  5000
.dwb   5000
V .equ  5000
add  %r1,V,%r1
or  %r1,4,%r1
x:  5000
```

b) Justifique la necesidad de que un programa ensamblador guarde junto con el código de máquina, una tabla de símbolos donde indica si cada símbolo es o no relocalizable.

4.

a) Justifique la necesidad de memoria cache en los procesadores modernos
b) Que entiende por mapa de memoria

Final 30-07-2018

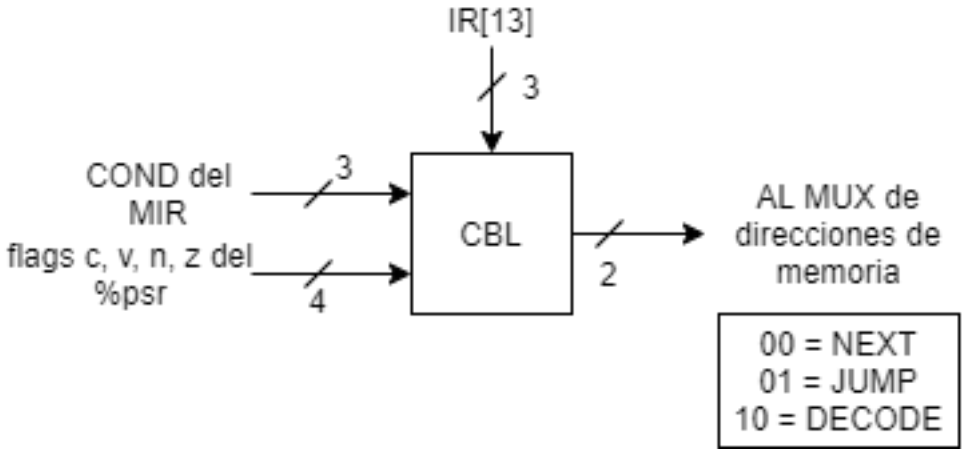
1. Ejercicio 1

   a) Los buses A y B cuentan cada uno con sus respectivos decodificadores, que reciben de la unidad de control el numero de registros a habilitar en el bus.
   Dichos decodificadores cuentan con una entrada de 6 bits que indica el numero de registro(son en total 38 registros) y la salida de 38 bits que habilita/deshabilita cada uno.

   b) El bloque de lógica de control de saltos es el que decide si la siguiente instrucción a ejecutar debe ser literalmente la siguiente en la ROM, si debe decodificarse del IR o si es un JMP al JMP ADDR del MIR. Esta unidad es necesaria debido a los diferentes tipos de saltos condicionales que existen(be, bneg, etc). Sus inputs son:

   - El campo COND de 3 bits del MIR que indica de done tomar la próxima microinstrucción: NEXT de la ROM, DECODE de IR o JUMP del MIR.
   - El bit 13 del IR que sirve como condición para ejecutar la próxima microinstrucción.
   - Los 4 bits de condición(n, c, v, z) del %psr.

   Con todos estos inputs, el CBL resuelve cual de los tres escenarios tomará lugar, enviandole el output al MUX de memoria que se encarga de elegir con esta información de donde actualizar la próxima dirección para luego disponerla en el MIR.



   c) La línea `2047: R[pc] <-- INCP(R[pc]); GOTO 0;`

   - Es la encargada de que el program counter apunte a la instrucción siguiente del código.
   - Es fundamental puesto que sin esta línea se ejecutaría siempre la misma instrucción.
   - Por último, el `GOTO 0;` hace que el microporgama luego guarde dicha instrucción en el IR y en la línea 1 la decodifica.

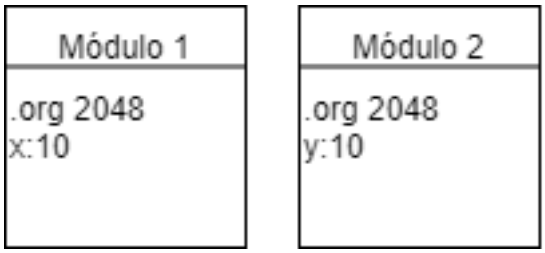
- Las instrucciones que no la necesitan son aquellas que al ejecutarse no deben seguir el flujo del programa secuencialmente, es decir son aquellas que ejecutan los saltos, como los branch y los call.

2. Ejercicio 2
   (al final)

3. Ejercicio 3

   a) `.org 5000` no ocupa nada
   `.dwb 5000` se reserva 5000 palabras de 32 bits
   `V .equ 5000` no ocupa nada
   `add %r1,V, %r1` 32 bits
   `or %r1,4, %r1` 32 bits
   `x: 5000` 32 bits
   Total = 160096 bits = 20012bytes

   b) La necesidad se debe a que puede suceder que la dirección en la que originalmente iba a estar un símbolo ya no este disponible. Por ejemplo, si al momento de linkear dos módulos independientes suceda:



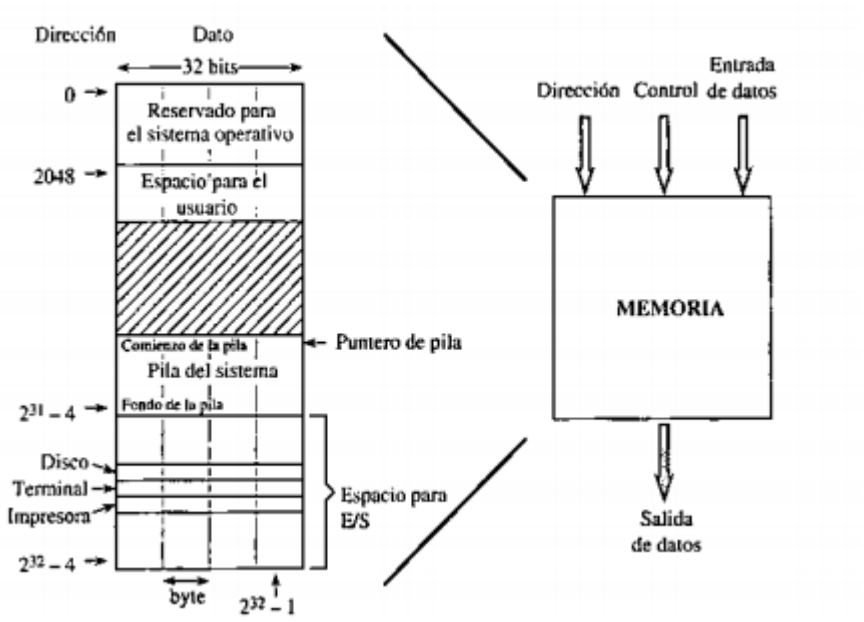
   En este caso, x e y están declarados en la misma posición y se pisarían. El linker tiene que resolver este problema, viendo en la tabla de símbolos cual símbolo es relocalizable para poder hacerlo y que no se pisen ambos.

4. Ejercicio 4

   a) La existencia de la caché y su utilidad se basa en el principio de localidad.
   - Localidad temporal: Estadísticamente, si se accedió a una posición de memoria la probabilidad de volver a a acceder a ella en el mismo programa es muy alta.
   - Localidad espacial: De igual manera, si se accedió a una posición es altamente probable que se acceda a posiciones contiguas, por ejemplo al almacenar datos se almacenan uno tras otro.

El fin de la memoria caché es una memoria que contiene una çopia"de estos bloques a los que se accede, la misma esta física y lógicamente cerca del CPU. Busca evitar la búsqueda en la memoria principal que es una de las operaciones mas costosas en cuanto a tiempo. Evita el bottleneck dado que la velocidad del CPU es mayor que la velocidad de la memoria.

b) El mapa de memoria es la distribución de los sectores de memoria para el sistema operativo, los dispositivos de entrada/salida, los programas del usuario y el stack. Existen muchas implementaciones distintas para el mismo procesador, en particular para ARC:



Un programa recibe por stack la dir de inicio y el largo de un array. Si arr[i] tiene en 1 los 2 bits más significativos, lo reemplaza por 0, en caso que no altere ningún elemento en el ultimo elemento carga D1000160h, la condición es evaluada por una subrutina a la que se le pasa por stack el valor a evaluar y devuelve 1 si se verifica la cond y debe ser declarada en el mismo modulo. (30-07-2018) (hay q definir push y pop, and que siempre no la recargo, r condición)

```
.begin
.org 2048
main : ld[dir], %r1              ! r1 puntero
   ld[tam], %r2                  ! r2 -> tamaño
   push %r1
   push %r2
   pop  %r2
   pop  %r1
   and  %r0, %r0, %r5            ! seteo en 0 a r5
   add  %r1, %r2, %r1            ! r1 al final del arreglo.
loop: andcc %r2, %r2, %r0
      be  fin_ciclo
      sub %r1, %r2, %r3          ! r3 -> dirección del dato
      ld  %r3, %r4               ! r4 -> dato a evaluar
      push %r4
      call evaluar
      pop %r4                    ! r4 -> el retun de evaluar (1/0)
      add %r5, %r4, %r5          ! r5 += r4
      andcc %r4, %r4, %r0
      be  sigo                   ! si la función devolvió 0 sigo
      st  %r0, %r3               ! si no, cargo un 0 en %r3
      sigo : add %r2, -4, %r2    ! avanzar pila
          ba loop
      fin_ciclo : andcc %r5, %r5, %r0
          be    bandera          ! si r5 es 0, no pise ningún dato
          ba    end
      bandera : sethi 3440000h, %r6
          add  %r6, 160h, %r6
```

```
        add %r1, -4, %r1    ! r1 -> dir ultimo elemento
        st  %r16, %r1
        ba  end
evaluar: for %r17
        srl %r17, 30, %r17  ! en 17 me quedo con los 2 bits mas significativos y 0
        subcc %r17, 3, %r10
        be  cambio
        and %r10, %r10, %r18    ! seteo a r18 en 0
        ba  return
cambio: or  %r10, 1, %r18       ! r18=1
return  push %r18
        jmpl %r15, 4, %r10
dir: 3000
tam: 16
end: halt
.end
```