4TO Final

Ejercicio ①

Escribir un programa que declara dos arreglos de 8 elementos cada uno y luego los completa con valores leidos de un periférico que está mapeado en la dirección A2A0123Ch. Cada lectura comprende 32 bits en los cuales se codifican 2 numeros ubicados en los 16 bits mas altos y en los 16 bits más bajos respectivamente. Ambos son enteros en complementos a 2.

La obtención de los numeros a partir de la lectura de 32 bits debe ser implementada en una sub declarada en el mismo modulo a la cual se le pasa por pila el valor leido y devuelve tamb por pila los 2 numeros en complemento a 2.

```
.begin
.org 2048
.macro  push reg
        add %r14, -4, %r14
        ST  reg, [%r14]
.endmacro
.macro  pop reg
        ld  [%r14], reg
        add %r14, 4, %r14
.endmacro
```

```
A   .equ   A2A01h
B   .equ   23Ch
SETH1  A,%r1
sll  %r1,2,%r1
add   %r1,B,%r1     ! almaceno en %r1 la dir del periferico

arrayA : .dwb 8
arrayB = .dwb 8
ld   arrayA , %r2    ! dir del arrayA (r2)
ld   arrayB , %r3    ! dir del arrayB (r3)
add  %r0 ,32,%r4     ! r4 es mi i para iterar en los array
call for
halt
for  : addcc %r4,-4,%r4
       bneg FIN
       ld %r1,%r10
       push %r10                 add %r15,0,%r20
       call obtener_numeros      (para no perder la ref y poder salir del for)
       pop %r7      ! 1er num
       pop %r8      ! 2do num
       add %r4,%r2,%r5
       ST  %r7,%r5          ! cargo en el array A
       add %r4,%r3,%r6
       ST %r7,%r6           ! cargo en el array B
       ba for
```

```
obetener_numeros:   pop %r12          ! recibo la lect del periferico
                    ld %r12, %r18     ! backup
                    srL %r12, 16, %r12
                    push %r12
                    sll %r18, 16, %r18
                    srL %r18, 16, %r18
                    push %r18
                    jmpl %r15+4, %r0

Fin:   jmpl %r20+4, %r0

.end
```

Ejercicio ②

Un procesador ARC esta ejecutando el siguiente programa:

```
.begin
        .org 2048
        cte    .equ   389 h
        Pd3DMS .equ   694 h
  2048  addcc  %r0, cte, %r20
  2052  subcc  %r20, Pd3DMS, %r10   → r10 =
  2056  ld  [a], %r12
  2060  norcc  %r10, %r12, %r1
  2064  call   3000
  2068  L1:  jmpl  %r15, 4, %r0
        a: 256
.end
```

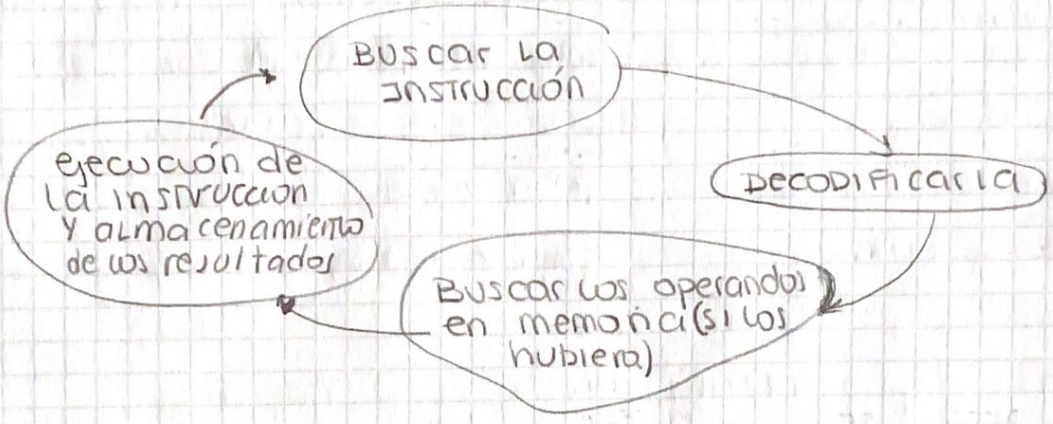
  389
- 694
_____

Ej ①

ⓐ Explicar de qué modo la instrucción es identificada con la entrada L1. es microprocesador (ejecutada por el) incluyendo cada uno de los pasos de microcódigo hasta completar el ciclo de FETCH

jmpl %r15,4,%r0

Ciclo de FETCH:



(IR)

10 00000 111000 01111 1 0000000000100

Decode

1 10 111000 00 → 1760

1760: IF R[IR[13]] then goto 1762,

1761: R[pc] ← add (R[rs1], R[rs2]), Goto 0

1762: R[tempO] ← SEXT13 (R[ir])

1763: R[pc] ← add (R[rd], R[tempO]) goto 0.

(b) el bit 13 es (1) → 3ck:

1763 R[pc] ← add(R[rs1], R[tempO]); Goto 0;

IR

10 00000 111000 01111 1 0000000000000

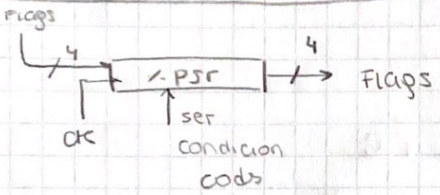
MIR

000000 1 100001 0 100000 0 00 1000 110 0000000000000

Program counter

PC = 2048 + 4.5 = 2068

Entradas y salidas del PSR



Valor de Γ10

Γ10 = FFFFFD25

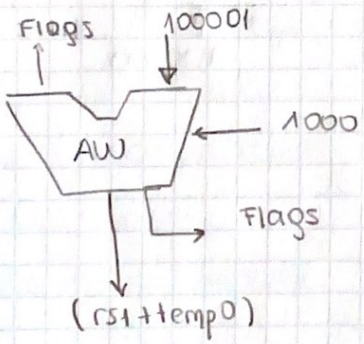
3B9 - 694 = 3B9 + Cm(694)

3B9 = 0000 0000 0000 0000 0000 0110 1001 0100

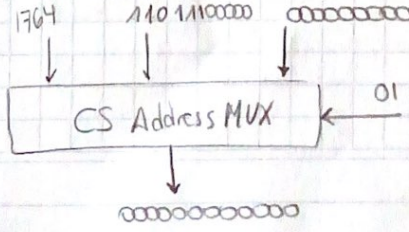
Cm (694) = FFFFF 1001 0110 1100

Γ10 = 00000 0110 1001 0100
FFFFF 1001 0110 1100
FFFFF D25

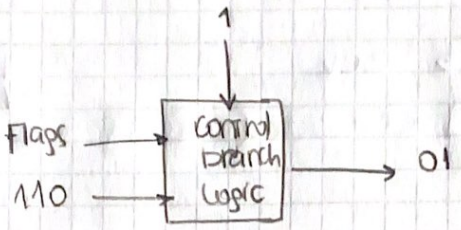
ALU



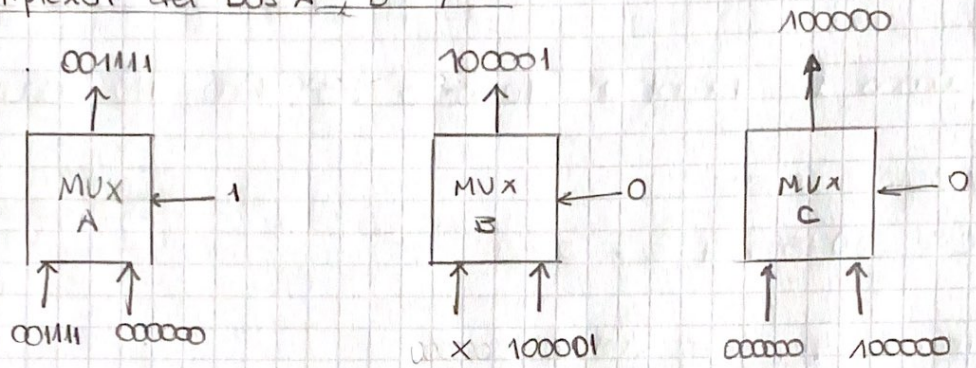
Entradas y salidas al multiplexor de memoria de control



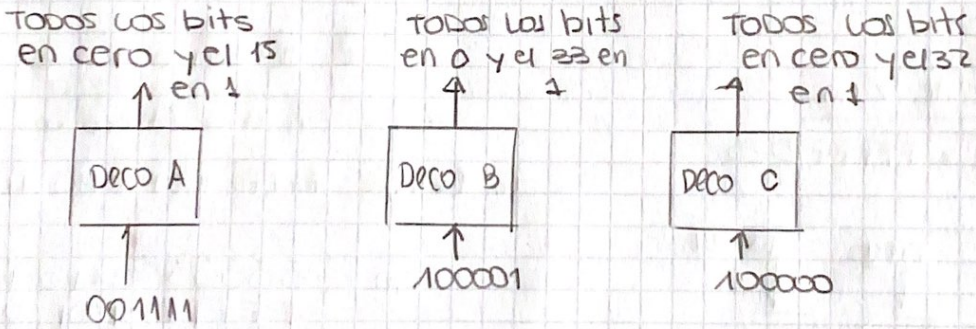
Entradas y salidas a la lógica de saltos



Multiplexor del bus A, B y C



Decodificador del BUS A, B y C



Ejercicio ③

③ Determine cuantos bytes de memoria RAM son necesarios para cargar el siguiente programa assembler Justifique detalladamente su respuesta.

```
.begin
        .org 2048
A       .dwb 2          → 12 palabras = 64 bits = 8 bytes
B       .equ 32
        ld [A], %r1          4
        addcc %r1, B, %r2    4
        be fin               4
        add %r1, %r1, %r1    4
Fin     jmpl %r15, 4, %r0    4
.end
```

④ ¿Que programa decide la localización (en reg, stack, etc) de cada variable declarada en un programa de lenguaje de alto nivel?

a) el compilador
b) el ensamblador
c) EL linker

Justifique y detalle su respuesta

NOTA

(a)

Contamos en este programa una única directiva que genera información en memoria que va es .dwb

.dwb 2 → 2 palabras = 32 bits x2 = 64 bits = 8 bytes

Y contamos con 5 instrucciones ARC, cada instrucción ocupa 32 bits = 4 bytes → 4·5 = 20 bytes

Por lo tanto, ocupa 28 bytes de RAM.

(b) El compilador es el encargado de pasar de lenguaje de alto nivel a lenguaje simbólico, el ensamblador una vez finalizado el proceso de compilación, es el encargado de traducir el lenguaje simbólico 1 a 1 a código de máquina.

El linker es el encargado de unir módulos distintos en Assembler, resuelve las referencias globales y externas y reubica las direcciones de los diferentes módulos para que no se pisen entre sí.