Apellido y Nombre

email:

Padrón:

Fecha: 24-julio-2018

Cuatr. cursada:

Turno de TP:

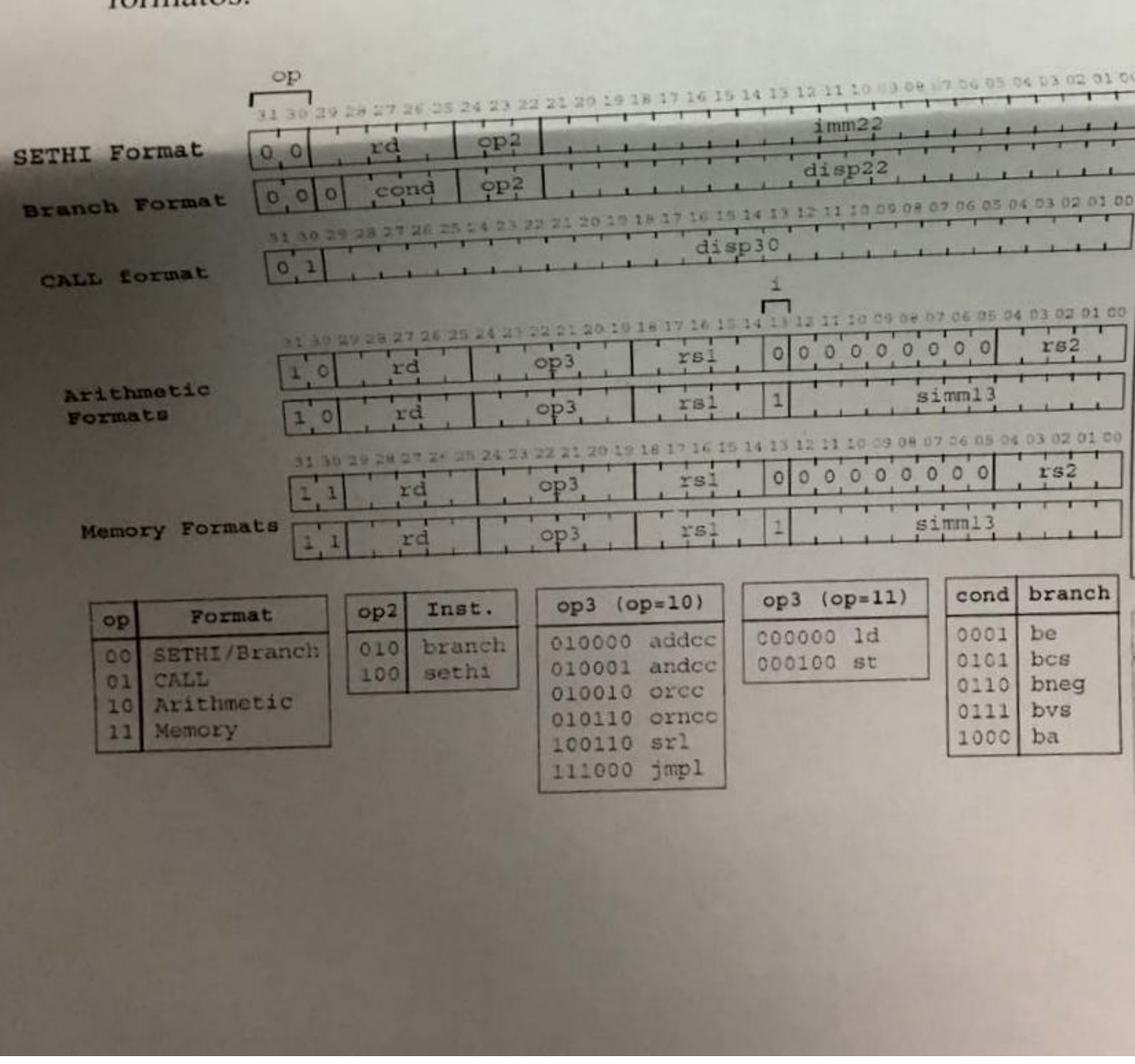
****

1) (a) Indique cuántos ciclos de reloj toma la ejecución de una microinstrucción y de qué manera se determina cuál es la microinstrucción que será ejecutada luego de la actual.
(b) Proponga un microcódigo que implemente la instrucción "jmpl" de Assembly ARC indicando su localización en la memoria de control. Incluya la expresión en 41 bits de las primeras dos líneas del microcódigo propuesto.
(c) Proponga un circuito para el registro 'program counter' indicando sus componentes y el detalle de sus conexiones con otros componentes de la microarquitectura.

2) Un programa lee números entregados por un dispositivo de entrada y los guarda en uno de dos arreglos según se trate de un entero positivo o negativo. Ambos arreglos, de 48 elementos cada uno, son declarados por el mismo programa. Una y otra vez repite este proceso hasta que se agote la capacidad de alguno de los dos arreglos y en ese caso termina.
Una rutina declarada en otro módulo tiene la función específica de leer cada valor entregado por ese dispositivo (que está mapeado en C0102030h) y devolverlo por vía de la pila.

3) Respecto del código anterior describa todos los archivos generados por el ensamblador y de qué manera estos son procesados en la etapa de linking.

4) Justifique la necesidad de definir los formatos "big endian" y "little endian". Describa ambos formatos.



F_3 F_2 F_1 F_0	Operation
0 0 0 0	ANDCC (A, B)
0 0 0 1	ORCC (A, B)
0 0 1 0	NORCC (A, B)
0 0 1 1	ADDCC (A, B)
0 1 0 0	SRL (A, B)
0 1 0 1	AND (A, B)
0 1 1 0	OR (A, B)
0 1 1 1	NOR (A, B)
1 0 0 0	ADD (A, B)
1 0 0 1	LSHIFT2 (A)
1 0 1 0	LSHIFT10 (A)
1 0 1 1	SIMM13 (A)
1 1 0 0	SEXT13 (A)
1 1 0 1	INC (A)
1 1 1 0	INCPC (A)
1 1 1 1	RSHIFT5 (A)

op	Format
00	SETHI/Branch
01	CALL
10	Arithmetic
11	Memory

op2	Inst.
010	branch
100	sethi

op3 (op=10)
010000 addcc
010001 andcc
010010 orcc
010110 orncc
100110 srl
111000 jmpl

op3 (op=11)
000000 ld
000100 st

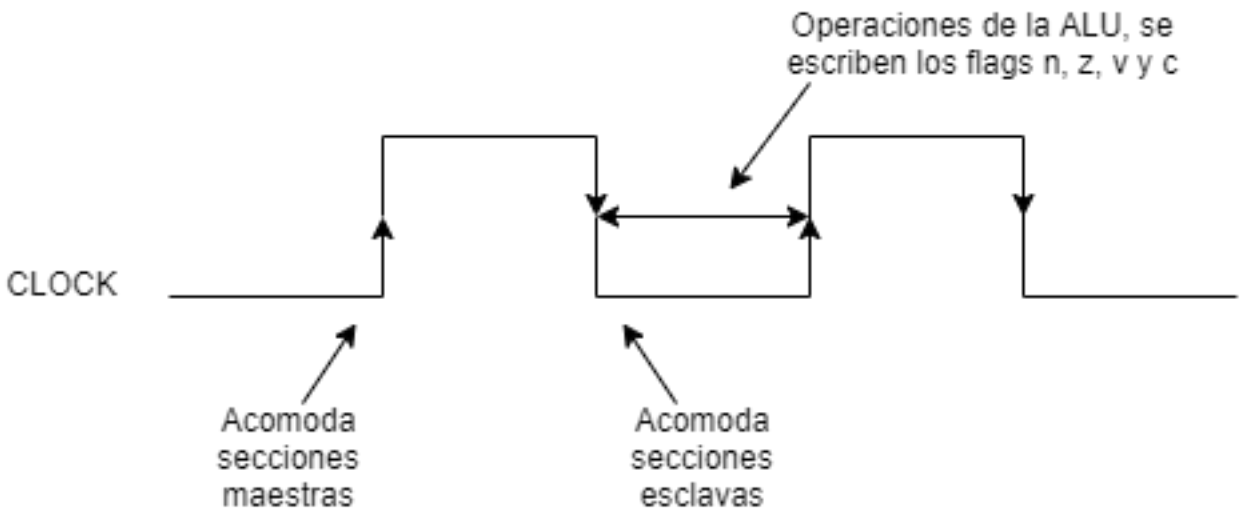
cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba

cond	Operation
0 0 0	Use NEXT ADDR
0 0 1	Use JUMP ADDR if n = 1
0 1 0	Use JUMP ADDR if z = 1
0 1 1	Use JUMP ADDR if v = 1
1 0 0	Use JUMP ADDR if c = 1
1 0 1	Use JUMP ADDR if IR[13] = 1
1 1 0	Use JUMP ADDR
1 1 1	DECODE

Final 24-07-2018

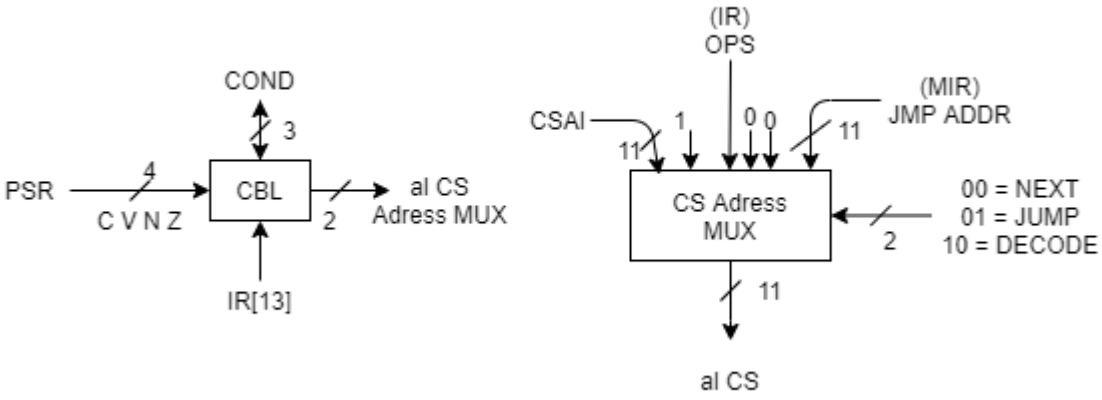
1. Ejercicio 1

   a) La microarquitectura opera sobre un ciclo de reloj de dos fases.
   - En el flanco positivo del clock cambian las secciones maestras de todos los registros.
   - En el flanco negativo la información almacenada en la sección maestra de cada FF se transfiere hacia la esclava(esto deja información disponible para las operaciones de la ALU).
   - Cuando el reloj esta en estado bajo, se llevan a cabo las funciones de ALU, MUX y CBL que deben completarse antes del próximo flanco positivo.



   La siguiente microinstrucción a ser ejecutada bien puede ser:
   - Inmediatamente siguiente en la memoria de control
   - Una a la que se salta en la memoria de control
   - La microinstrucción `DECODE`

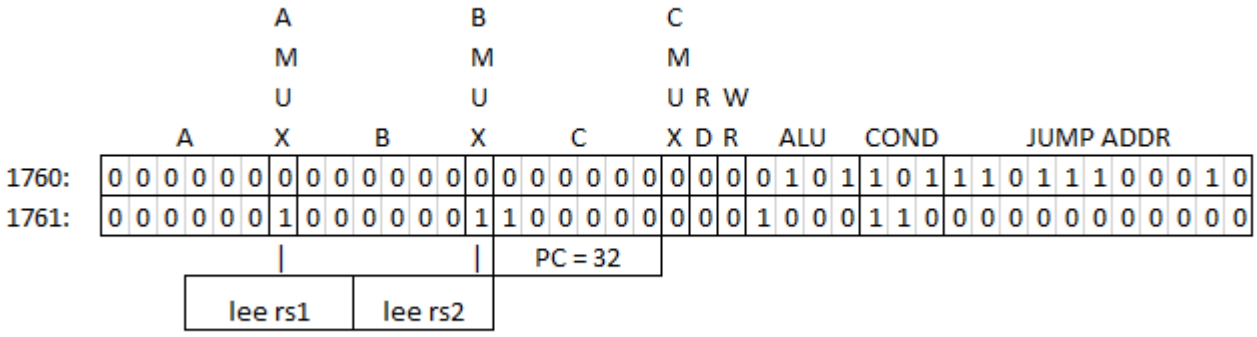
   Esto lo determina el CBL; a partir de sus entradas(el campo COND, los flags del PSR y IR[13]), determina si para la microinstrucción actual será NEXT, JMP o DECODE.
   Envía dicha información por 2 bits al CS Adress MUX que le dice precisamente la dirección que se requiere a la memoria de control(quien luego la cargará en el MIR)



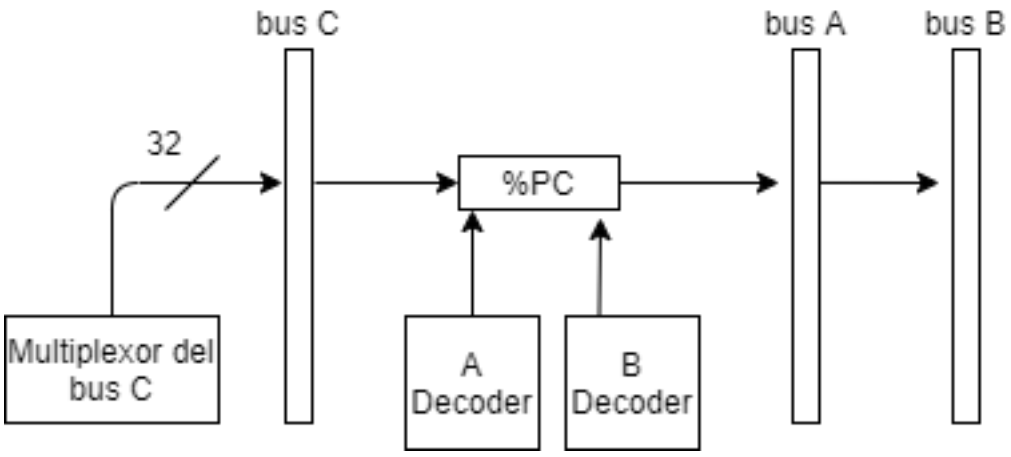
b) jmpl: op = 10 op3 = 111000
DECODE recibe 1101110000 = 1760

```
1760: IF R[IR[13]] THEN GOTO 1762;
1761: R[pc] <-- ADD(R[rs1], R[rs2]); GOTO 0;
1762: R[temp0] <-- SEXT13(R[IR]);
1763: R[pc] <-- ADD(R[rs1], R[temp0]); GOTO 0;
```

El MIR queda:



c)



2. Ejercicio 2

Un Programa lee números entregados por un dispositivo de entrada y guarda en dos arreglos según si el num es >ó< a 0. Ambos arreglos de 48 elementos cada uno, son declarados por el mismo programa. Una y otra vez se repite el proceso hasta que se agota la capacidad de alguno. Una rutina declarada en otro módulo tiene la función de leer y volver del periférico (mapeado en Cafe2030h) y devolver el valor via stack (24/07/18)

```
.begin
.org 2048
.extern leer
.macro Push arg                    !macroins para agregar elem a la Pila
      Add %r14,-4,%r14
      St arg,%r14
.endmacro
.macro Pop arg
      ld %r14,arg                  !macro ins para obtener elem de la Pila
      Add %r14,4,%r14
.endmacro
.macro poner dir,cont
    Sub dir,cont,%r8
    St %r15,%r8
    Add cont,-4,cont
.endmacro
DIR1  .equ 3000+192                !Vectores para los numeros negativos/positivos
DIR2  .equ 3192+192                !
main: Or %r0,DIR1,%r1              ! r1 → puntero a arreglos pos
      Or %r0,DIR2,%r2              !. r2 → Puntero a vec negs
      Sethi 294028h,%r20           ! dir del
      Add %r20,120h,%r20           ! periferico.
      Or %r0,192,%r3               !tamaños de
      Or %r0,192,%r4               !los arreglos
      Push %r15
loop  Andcc %r3,%r3,%r0            !si algun vector se quedo sin
      be end                       !capacidad termina
      Andcc %r4,%r4,%r0
```

```
be end
Call leer
POP %r15          !guardo el valor leído en %r15
Anddcc %r15, %r5, %r10   !veo si r5 es + ó -
bneg negativos
Poner %r11, %r13
 ba loop
 leer: ld %r20, %r15   !cargo en %r13 el 1° elem del vector
       Push %r15
       Jmpl %r15, 4, %r0
 end: Pop %r15
      Jmpl %r15, 4, %r0

.org 3000
Arreglo 1: .dwb 48
Arreglo 2: .dwb 48
.end
```

3. Ejercicio 3

- Archivo código objeto:
  - Código de máquina
  - Dirección de la primera instrucción
  - Símbolos declarados en otros módulos (externos)
  - Símbolos globales
  - Librerías externas

- Información sobre la reubicación del código

- Linking: para combinar módulos, el linker:
  a) Resuelve referencias externas
  b) Reubica cada módulo de forma apropiada
  c) Especifica símbolo principal de cada módulo
  d) Especifia los contenidos de los diferentes segmentos de cada módulo
- Loader: Toma el código objeto y reubica los módulos con offset para que entren todos el programa en memoria simultaneamente sin que se pisen los distintos módulos.

4. Ejercicio 4
   La necesidad de definir los formatos big y little endian es porque los bits de los registros pueden ser guardados tanto como el bit mas significativo en la posición mas baja de memoria como de la forma contraria.

   - Big endian: almacena el byte menos significativo en la posición mas baja de memoria.
   - Little endian: almacenar el byte más significativo en la dirección más baja de memoria

   Representación de 0x1A2B3C4D5E6F7080: