Apellido y Nombre: Fecha: **22-12-2015**

email: Cuatr. cursada:

Padrón: Turno de tp:

****

1) (a) Diseñar una unidad aritmético-lógica capaz de resolver apenas dos operaciones (NORCC y AND) con datos de 8 bits. Incluir la generación flags.

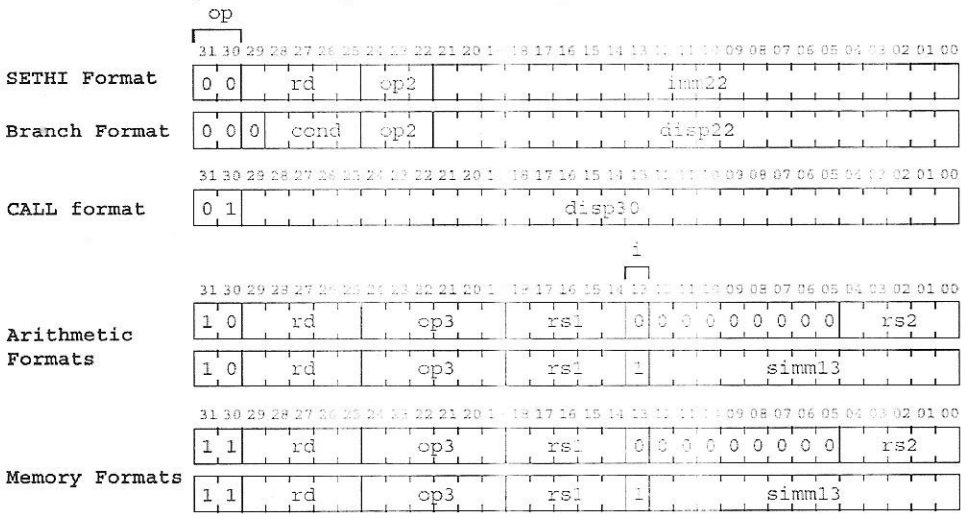
   (b) En la microarquitectura ARC el incrementador de direcciones de la memoria de control tiene conexión con la memoria RAM Explique los problemas de funcionamiento que pudieran surgir si esa conexión se corta.

   (c) Proponga un microcódigo para un procesador ARC que permita implementar la instrucción que implemente la instrucción de assembler "srl". Indique su dirección en la memoria de control. Detalle también otras microinstrucciones que resultan necesarias para su integración al ciclo de búsqueda-ejecución. Presente el contenido binario de la primera posición de memoria en que se encuentra almacenado el microcódigo propuesto para la instrucción srl.

2) Escribir un programa en código ARC tal que recibe a través de la pila un número de 32 bits en representación de punto flotante, lo multiplica por 2 y escriba el resultado en un dispositivo de salida que se encuentra mapeado en la dirección B0001010h.
   La operación de duplicar el valor en punto flotante es llevada a cabo por una rutina declarada en un módulo diferente al programa principal que el programa principal con quien intercambia argumentos via la pila. Si el resultado excede el rango de representación, devuelve "infinito"
   La operación de escribir el resultado en el periférico es llevada a cabo por un rutina declarada en el mismo módulo. Esta recibe a través de la pila dos argumentos: el valor de 32 bits y la dirección donde escribirlo.
   Se pide escribir código para el programa principal y las dos rutinas.

3) Describa detalladamente los pasos necesarios para que se genere un archivo ejecutable a partir del código propuesto en el punto anterior e explique de que modo ese archivo es ejecutado cuando el operador de la computadora así lo requiere.

4) Explique porqué se espera que un procesador que opera con memoria cache lo haga más rápidamente que otro idéntico pero sin memoria cache.



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

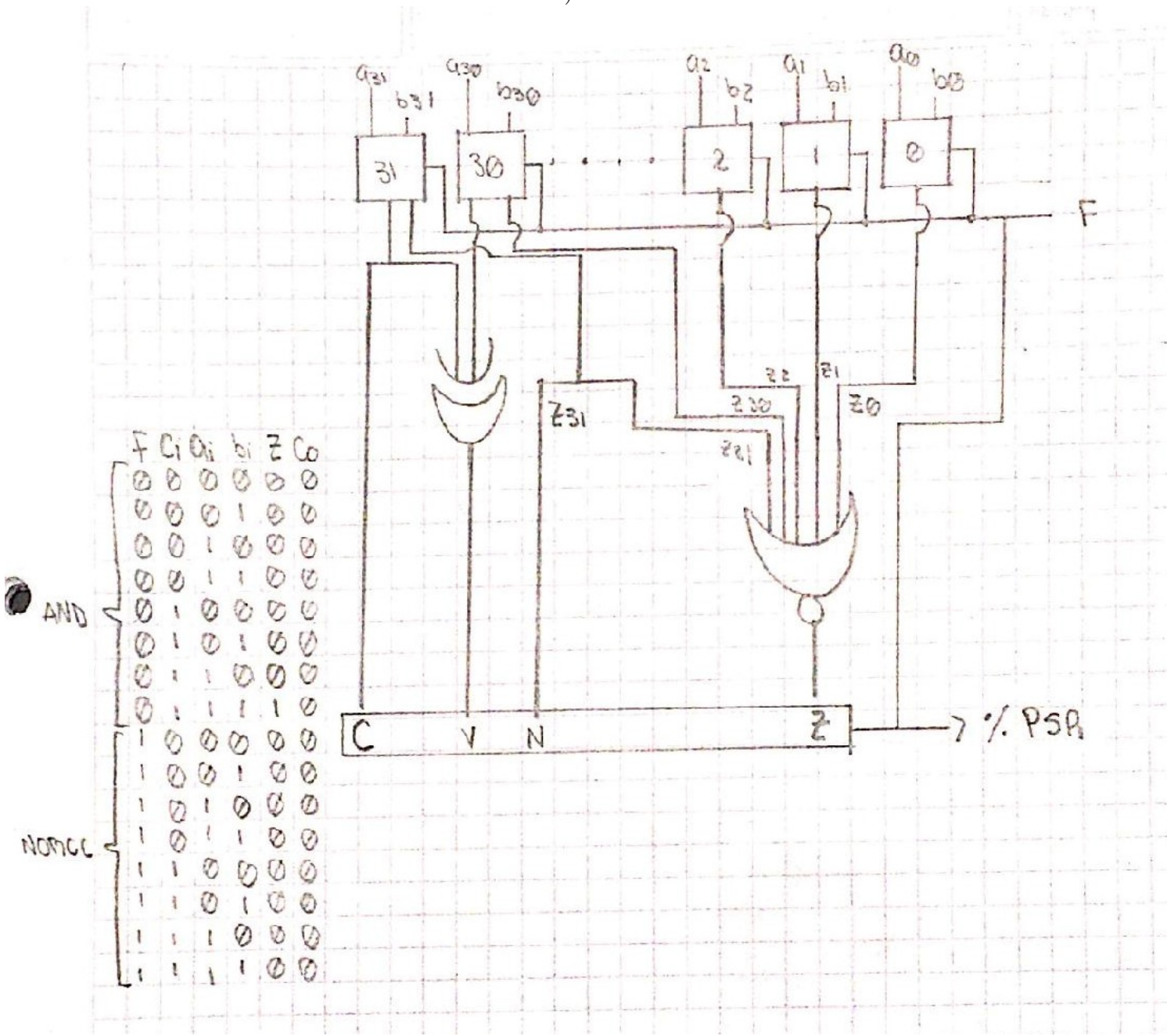
cond	branch
0001	be
0101	bcs
0110	bneg
0111	bvs
1000	ba

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)

C_2 C_1 C_0	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

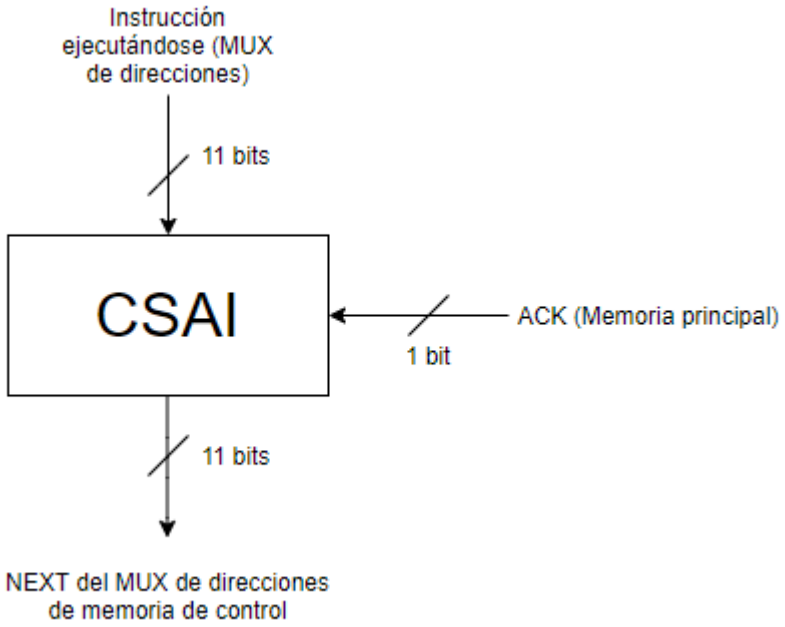
1. Ejercicio 1

a)



b) El CSAI tiene como input un bit ACK proveniente de la memoria principal. Este bit es enviado para indicar que termino un proceso de escritura/lectura en la memoria RAM y hasta que este bit no se envíe la misma no se incrementará. En caso que se corte la conexión el CSAI jamás se enteraría el cese de la operación, no pudiéndole así mandar la nueva instrucción al MUX de direcciones de memoria de control. Podría pasar que se pase a la siguiente instruc-

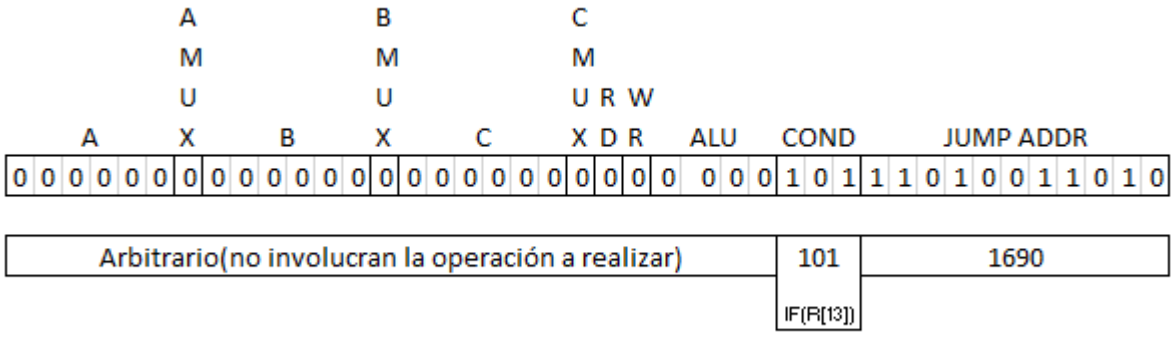
ción sin haber terminado la actual, almacenando/leyendo así cosas incorrectas.



c) srl: op = 10 op3 = 100110
1101001100 = 1688

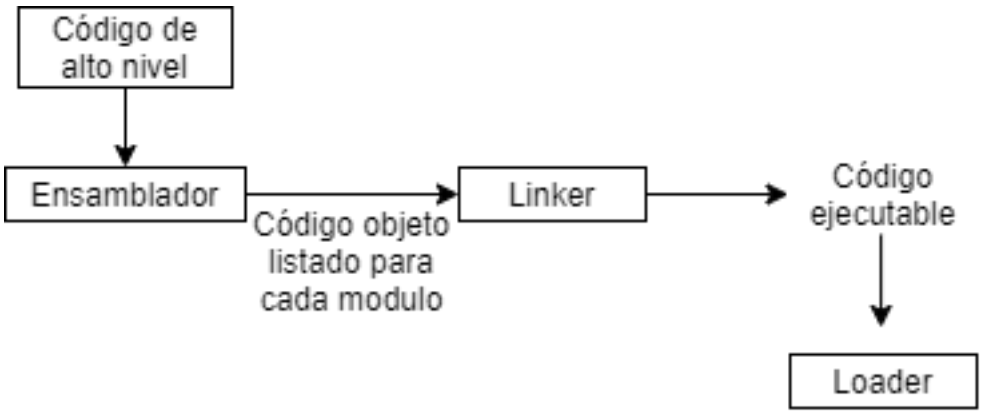
```
0: R[IR] <-- AND(R[PC], R[PC]); READ;
1: DECODE
1688: IF R[IR[13]] THEN GOTO 1690;
1689: R[rd] <-- SRL(R[rs1], R[rs2]); GOTO 2047
1690: R[temp0] <-- SIMM13(R[IR]);
1691: R[rd] <-- SRL (R[rs1], R[temp0]); GOTO 2047;
2047: R[pc] <-- INCPC(R[pc]); GOTO 0;
```

0: Pone el bus de direcciones de memoria con pc, pone el bus de datos desde memoria a IR lee memoria.
1: Decodifica la microinstrucción inicial 2047: Incrementa el program counter en 4(próxima linea) y llama a 0 para completar el ciclo de fecth.
MIR en 1688:



2. Ejercicio 2
   Ejercicio de assembler muy difícil el cual elijo creer que no me tomarán

3. Ejercicio 3



   El ensamblador genera un listado que incluye la tabla de símbolos y el código objeto que contempla: la primera instrucción a ejecutar, los módulos externos y globales e información de código realocable. Esto ultimo es util para el linker.
   El linker toma estos módulos y los junta en un código ejecutable solucionando las necesidades de relocalizar el código. Por ejemplo, en caso de que haya dos modulos declarados en la misma posicion, con la infotmación brindada por el ensamblador el linker se encarga de ver cuales simbolos son reubicables, es decir que cosas puede mover para evitar que los módulos se pisen.
   Por ultimo, el loader carga el programa del disco a memoria para que sea ejecutado.

4. Ejercicio 4

   Un procesador caché tiende a operar mas rápido por el principio de localidad.
   La caché es una memoria pequeña y rápida ubicada física y lógicamente cerca del

procesador por lo cual es mucho más rápida que la memoria RAM y disminuye el tiempo de acceso a los datos.
Almacena a los bloques de código a los que se acceden.
Localidad temporal: las palabras de memoria accedidas recientemente tienen una alta probabilidad de volver a ser accedidas en el futuro cercano. La localidad temporal de los programas viene motivada principalmente por la existencia de bucles.
Localidad espacial: las palabras próximas en el espacio de memoria a las recientemente referenciadas tienen una alta probabilidad de ser también referenciadas en el futuro cercano. Es decir, que las palabras próximas en memoria tienden a ser referenciadas juntas en el tiempo. La localidad espacial viene motivada fundamentalmente por la linealidad de los programas (secuenciamiento lineal de las instrucciones) y el acceso a las estructuras de datos regulares.
Como el acceso a los datos es una de las operaciones mas costosas en termino de tiempo, la memoria cache al almacenar los bloques de código que tienden a ser los mas usados en ella, disminuye este tiempo ahorrando la búsqueda de datos en memoria. Puede suceder un fallo de caché que se da cuando el contenido de la dirección no se encuentre en ningún bloque ubicado en alguna línea de la caché.