(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.

Microcódigo de la instrucción SRL

Primero, obtendré la dirección de memoria de la primera microinstrucción correspondiente al microcódigo de SRL.

DECODE

1 10 100110 00 = 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], R[PC]);
GOTO 0;

Las microinstrucciones en negrita son las necesarias para que se cumpla el ciclo de Fetch que ejecutará correctamente la instrucción SRL.

1688: IF R[IR[13]] THEN GOTO 1690;

MIR de esa microinstrucción

A AMUX B BMUX C CMUX RD WR ALU COND JUMP ADDR

000000 0 000000 0 000000 0 0 0 000 101 11111111111

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.

Recibe a través de stack un número de 32 bits en punto flotante, lo multiplica por 2 y escribe el resultado en un dispositivo en B0001010.

Rutina para duplicar el valor en módulo diferente, intercambian argumentos por pila.

Rutina para escribir el resultado en el periférico en el mismo módulo, recibe a través de la pila el valor de 32 bits y dónde escribirlo.

```
                !Módulo principal
                .begin
                .org 2048
                .extern duplicar
                .macro push arg
                add %r14, -4, %r14
                st arg, %r14
                .end macro
                .macro pop arg
                ld %r14, arg
                add %r14, 4, %r14
                .end macro
main:           ld [periferico], %r2
                call duplicar !ya esta el elemento cargado en pila
                push %r2
                call escribir
escribir:       pop %r3 !recupero direccion periferico
                pop %r4 !recupero elemento
                st %r4, %r3
                jmpl %r15+4, %r0
periferico:     0xB0001010
                .end

                !Módulo externo
                .begin
                .org 2048
                .global duplicar
                ld %r14, %r29 !tendría que definir nuevamente la macro pop y push
                add %r14, 4, %r14
```

```
srl %r29, 1, %r29
add %r14, -4, %r14
st %r29, %r14
jmpl %r15+4, %r0
.end
```

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.

Como ya tengo el código expresado en lenguaje ensamblador o Assembler, se debe seguir con el paso del ensamblado para convertir ese programa a código de máquina.

El ensamblador de ARC es de dos pasadas, es decir lee 2 veces el código, esto permite poder usar los símbolos de un programa antes de definirlos. Además de permitir usar macros, ubicar palabras en memoria, definir etiquetas para las direcciones de memoria constante, entre otras prestaciones.
Comienza con un preproceso donde se expanden las macros y se registran las definiciones y se reemplazan. Luego en la primera pasada se crea la tabla de símbolos la cual para cada uno contiene su dirección de memoria, si es reubicable o no y es global o extern.
En la segunda pasada se genera el código de máquina.

Luego seguimos con el enlace realizado por el linker que se encarga de combinar 2 o más módulos que hayan sido ensamblados de forma separada, además se encarga de reubicar direcciones internas en caso de que se superpongan en los módulos y de resolver las referencias globales y externas.

Seguimos con la carga que la realiza el loader, donde se cargan los segmentos de memoria y se deja el archivo listo para su ejecución, puede ocurrir una reubicación de direcciones para lograr la coexistencia de programas en memoria.

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

La memoria caché es un agregado que se crea para reducir la brecha en tiempos de acceso entre la memoria DRAM y el CPU.

La memoria caché es una memoria con poca capacidad del tipo RAM estática (SRAM) por lo cual es más rápida que la dinámica y tiene una conexión física con el CPU, es decir, no utiliza los buses para la transferencia de datos.

Esta memoria funciona en base a lo que se denomina Principio de Localidad:

- Localidad Temporal: si accedí a una dirección de memoria, es probable que vuelva a acceder a ella en poco tiempo.
- Localidad Espacial: si accedí a una dirección es memoria es probable que acceda a sus direcciones contiguas.

Utilizando este principio, se logra reducir notablemente el acceso a la memoria principal, ya que cuando se busca acceder a un dato o instrucción, se buscará en la memoria caché antes que en la memoria principal. La idea es ir añadiendo los bloques de memoria desde la principal a la caché a medida que vamos avanzando.

Para poder aprovechar al máximo esta ventaja que nos da este tipo de memoria, se pueden proponer algunas estrategias en alto nivel como:

- Recorrer de manera secuencial, ordenada y predictiva las estructuras de datos (favorecemos la localidad espacial).
- Empaquetar las funciones y los datos más usados en un mismo bloque (favorecemos la localidad temporal)

Podemos tener diferentes estructuras de caché como:

- Caché Especializado: tendremos un caché para instrucciones y para datos. Esto resulta ventajoso ya que tenemos las tareas divididas y cada bloque podrá usarse de la manera que le sea más útil, por ejemplo en el caso de las instrucciones, como la mayoría de las operaciones son de lectura, podríamos tener un bloque más simple, mientras que para los datos, podemos realizar uno más complejo.
- Caché Multinivel: tendremos varios niveles de caché desde el más cercano al CPU en adelante. La idea es que cuando el caché más próximo se llene, pueden guardarse datos en el próximo nivel. En general se usan 2 o 3 niveles. Esto retrasa aún más el acceso a la memoria principal.