

1.

(a) Explique cuáles son los componentes de la microarquitectura ARC que hacen que la ALU 'vea' en cada entrada sólo uno de todos los registros del procesador (este es el que fue elegido por el programador).

① a) La ALU únicamente ve uno (por cada bus de entrada de datos) de todos los registros del procesador ya que lo MIR (registro de micro instrucciones) le indica a los MUXes de datos cuál es selección efectiva.

De este forma, el MUX selecciona si el registro que se colocó en el bus es el indicado en la instrucción (rs1 y rs2 según corresponda) o el que indica lo cedido de lo MIR (A y B según corresponda).

Finalmente y gracias a que lo MIR controla el flujo de datos y de operación, la ALU opera con los registros seleccionados para los buses A y B según corresponda.

Microarquitectura

¿Qué es lo que falta explicar?

Me centré únicamente en la operatoria de la ALU, que fue lo que solicitaba el ejercicio.

(b) Proponga un microcódigo para la instrucción *jmpl*. asignándole ubicación en la memoria de control. Debería estar repetido en varias localizaciones?

b) decode = 1 1 0 1 1 0 00 0 0 = 1760d

Un microcódigo para *jmpl* podría ser:

1760: IF R[12][13] THEN GOTO 1762,

1761: R[pc]  $\leftarrow$  ADD (R[rs1], R[rs2]), GOTO 0;

1762: R[temp0]  $\leftarrow$  SEXT13 (R[ir])

1763: R[pc]  $\leftarrow$  ADD (R[rs1], R[temp0]), GOTO 0,

El microcódigo propuesto es elenco para la instrucción *jmpl* puesto que no existe otra instrucción en ARC que altere el contenido del program counter excepto las llamadas al valor específico. El "call" altera el program counter pero no permite que el programador decida cuál será el valor si no que lo hace como consecuencia de realizar en saltos o en subrutinas.

¿Qué estaría mal en esta respuesta? El decode apunta a la ubicación 1760, por lo que no puedo situar la instrucción en otro lugar y el microcódigo se despliega a partir de dicha ubicación.

(c) Explique de qué forma los flags de la ALU controlan el flujo del microprograma detallando los componentes circuitales que intervienen.

c) Los flags de la ALU se ven alterados por dichas operaciones que causan los cambios en el código de control (es decir los que terminan con "cc"). Estos se almacenan en el registro "psr" que a su vez es leído por el CBL (Control Branch Logic) que ejecuta las branquias en el código. Es por esta razón que, al utilizar branquias en código assembly, estos estarán sujetos a las operaciones basadas en los resultados de las operaciones que alteran los flags de la ALU.

Podemos decir que existen 4 flags:  
 C, V, N, Z. En sucesión, la ALU es <sup>la</sup>  
 compuesta de 32 LUT's <sup>(look up Table) = (Tabla de Verdad)</sup>, cada uno con  
 32 bits de entrada correspondientes al bus A  
 y 32 bits de entrada correspondientes al  
 bus B; cada LUT tiene un bit de  
 salida de los cuales los 31 menos signifi-  
 cativos están conectados a una barra  
 de desplazamiento que contiene un  
 selector para decidir la dirección de  
 dichos desplazamientos en caso de realizar  
 una operación del tipo SHIFT.

En detalle, diremos que el flag C (carry)  
 se puede cuando <sup>los 2</sup> bits más significativos  
 de salida de los LUT's (<sup>los 2</sup> más significativos)

En detalle, diremos que el flag C (carry)  
 se puede cuando <sup>los 2</sup> bits más significativos  
 de salida de los LUT's estén encendidos; <sup>con AND</sup>  
 el flag V (over flow) se prende cuando  
 los últimos 2 bits más significativos  
 de salida de los LUT's son distintos y  
 se implementa con una compuerta <sup>con AND</sup> IX-NOR;  
 el flag N (negative) se prende cuando  
 el bit más significativo es 1;  
 el flag Z (zero) se prende cuando  
 todos los bits son distintos a 0 con NAND.

NOTA

NOTA

No entiendo por qué la primera parte está mal.

Por otro lado, en la segunda parte expliqué cómo se generan estos flags para definir el flujo a futuro del programa. No veo por qué no correspondería explicar esto cuando es un punto importante.

- 2) Un programa recibe por stack la dirección y largo de un arreglo conformado por números enteros en complemento a 2 y se encarga de devolver por esa misma vía la suma de todos sus elementos positivos de dicho arreglo. Antes de terminar invoca una subrutina que escribe un 0 en la dirección A2010B10h en caso de que la suma anterior no sea representable en 32 bits o escribe un 1 en caso contrario. (a) subrutina y main forman parte del mismo módulo (b) indicar los cambios necesarios para que la rutina este declarada en un módulo aparte

2) a)

②. begin

```

.org 2048
A .equ A2010h
B .equ B10h
    
```

- 0) sethi A, %r1
- 1) slli %r1, 2, %r1
- 2) addl %r1, B, %r1 ! r1 < A2010B10h
- 3) addl %r1, %r0, %r1 ! backup del pc.
- 4) pop %r2 ! r2 < dirección arreglo
- 5) pop %r3 ! r3 < largo arreglo.
- 6) addl %r2, %r0, %r4 ! r4 < dir. arreglo
- 7) addl %r0, %r0, %r5 ! r5 < acum. pos
- 8) addl %r0, %r0, %r6 ! r6 < total
- 9) addl %r0, %r0, %r7 ! r7 < acum. dir
- 10) addl %r0, %r0, %r7 ! inicializo r10 en 4
- 11) for: subcc %r3, %r7, %r7 ! largo - i
- 12) bpos check-sign
- 13) beg fin ! i == largo.
- 14) fin: addl %r15, %r0, %r12
- 15) call resultado
- 16) jump %r13, 4, %r0
- 17) resultado: st %r10, %r1
- 18) jmp %r16, 4, %r0 ! vuelvo
- 19) check-sign: ld %r4, %r20
- 20) addl %r4, 4, %r4
- 21) addl %r15, 1, %r15
- 22) subcc %r20, %r0, %r0
- 23) bcc es-pos.

27) Es - pos.: addcc r4, r10, r14  
 bcs no - es - representable.

28) no - es - representable: addcc r10, r10, r14  
 call resultado. *Termino*  
*sal -*

29)
 

- . macro pop reg
- ld r14, reg
- addcc r14, -4, r14
- . end macro
- . enda

En este ejercicio usted me marcó 2 errores. El primero fue que la instrucción subcc no tiene el último parámetro, que es el registro que almacena el resultado de la instrucción. Está más que claro -de ver el resto del código- que fue producto de un olvido por mi parte, y no por desconocer la sintaxis de assembly. En esa línea, el parámetro correspondía a un %r0 pues no utilizo el resultado de la operación sino que utilicé únicamente los flags resultantes de la operación. El segundo comentario es, si el código finaliza en call resultado. Esto no es un error, puesto que una vez que la sumatoria supera la máxima representación posible, el programa debe finalizar, y es la lógica que replico llamando a la subrutina resultado.

- 3) Presente la tabla de símbolos del programa anterior (punto 2)) y explique detalladamente porqué esta información resulta fundamental para la aplicación sea correctamente ejecutada.

symbol	value	extern global	reloc
A	42010h	-	no
B	B10h	-	no
pop			
for	2048 + 4.13	-	sí
check-sign	2048 + 4.21	-	sí
fin	2048 + 4.40	-	sí
resultado	2048 + 4.19	-	sí
es-pos	2048 + 4.29	-	sí
no-es-representable	2048 + 4.28	-	sí

Esta información es fundamental para la ejecución del programa ya que es utilizada por el ensamblador para emplear los símbolos en los que fueron necesarios al inicio del ensamblado, y por el linker para juntar los programas de distintos módulos y que estos puedan ser ejecutados como uno sólo y enlazados con los otros entre los mismos.

*Muel*  
 El procesador ARC tiene un ensamblador de 2 pasos. Esto es: en la primera etapa lee todo con los símbolos que se acierten descompoci-

NOTA

do. En lo segundo guardo un archivo con los valores correspondientes en el programa.

Adicionalmente, al compilador se le pasa la tabla con valores que posteriormente utilizará el linker al enlazar los diferentes programas, tales como si el símbolo es externo, global y/o relocable.

La tabla de símbolos es fundamental para el ensamblador y el enlazado de los diferentes programas.

Muy  
mejor

En este ejercicio cometí el error de llamar al ensamblador de doble pasada como una funcionalidad propia del microprocesador ARC, cuando en realidad es un programa en sí. El resto del ejercicio creería que está completo, y revisando el material no encuentro información adicional que podría haber propiciado en el examen que no esté fuera de tema. Aunque entiendo que el error es llamativo, no creo que sea suficientemente grave como para descalificar por completo el ejercicio.

- 4) (a) ¿Qué entiende por mapa de memoria? (b) Queda este definido al elegir un determinado microprocesador? (c) Explique eventuales diferencias entre los mapas de memoria de sistemas con y sin memoria cache

④ a) El mapa de memoria define cómo están destinadas las distintas funciones que realizan en el procesador en su memoria principal.

El mapa nos da la información sobre dónde y cuánto espacio ocupan:

- El sistema operativo.
- El bloq que da al usuario.
- El slack (p.12).
- Dispositivos de Input / Output.

El mapa de memoria define cómo están destinadas las distintas direcciones de la memoria principal a los ojos del procesador.

c) Tener un sistema sin memoria cache implica que todas las operaciones que requieren de una lectura y/o escritura en memoria, deberán realizarse sobre la memoria principal. Esto, está conectado al CPU mediante un Bus de datos que permite la transferencia de datos entre los componentes. Ahorramos bien, no sólo que la memoria principal

lo tiene la característica de ser lo más rápido de todos los existentes, sino que además el la comunicación mediante el bus tiene todo la operación (yo sé que se está leyendo como que si esté escribiendo lo mismo).

Es por esto que se incorporan a los sistemas memoria cache. Estos son localizados entre el CPU y la memoria principal. ~~son más~~ Claro que la memoria cache es mucho más rápida que la memoria principal pero es mucho más veloz (además de ser más cara).

~~la memoria cache guarda un punto a tanto que esté~~

De este forma, el programador lee y escribe a la memoria cache según corresponda. La cache almacena las palabras que el CPU le solicita de la memoria principal. Si es lo suficiente vez que solicita esta información, accederá a la memoria principal mediante el bus de datos y almacenará la información correspondiente. Una vez finalizada la ejecución, escribe en memoria principal según corresponda.

Di Matteo, Carolina 103963

HOJA N° 7 / 7

FECHA

Este técnico se llama MAPEO DE MEMORIA CACHÉ y se implementa mediante slots en caché y blocks en memoria principal.

Siguiendo el mapa, cada slot es lo dirigido a uno o más bloques de memoria y de este forma se pueden tener una especie de "puertas" o memoria principal usando una caché.

En resumen podemos decir que el uso de memoria caché en un sistema debe optimizarse para aumentar significativamente la velocidad de ejecución de un programa, mientras que esto lleva la lentitud por el gran costo que tiene leer y escribir en memoria cada vez que se realiza una operación.

Más respuesta  
de pregunta  
gr

No entiendo por qué está incompleta.

El mapa de memoria en definitiva es un reflejo de la memoria principal, de la cual hablé abiertamente en todo el ejercicio y cómo ésta interactúa con la caché y cómo ambas operan con el CPU.

En caso que esto no fuera correcto, ¿en dónde puedo encontrar la respuesta correcta o material para leer sobre este tema?