DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (I/2020)

I_3

Arquitecturas y Pipeline.

6. Pregunta ISA

Los ayudantes del curso están creando una nueva ISA.[1] Pero no logran ponerse de acuerdo si construirla CISC o RISC, así que necesitan que les respondas algunas preguntas para ayudarles a tomar la decición.

En base a la materia de clases, responde:

1. ¿Podría la ISA RISC ser un subconjunto de las instrucciones de la ISA CISC? ¿Porqué?
   Lo esperado... Debes indicar "sí", "no" o "depende" y explicar los motivos, basándote en las características y propuestas de cada uno de los distintos tipos de ISA.

2. Si tuviéramos un programa escrito con la versión CISC de la ISA y quisiéramos convertir su código assembly a la ISA RISC. ¿Qué consideraciones deberíamos tener al hacer la conversión? Describe el proceso paso a paso.
   Lo esperado... Se espera que resaltes las dificultades a las que nos enfrentaríamos y cómo se resuelven, llegando a una serie de reglas con las que puedas crear tu algoritmo, basándote también en las características y propuestas de cada uno de los tipos de ISA.

7. Pregunta Microarquitecturas

Se te entregará una ISA, en base a ella implementa una micro-arquitectura **Von Neumann** de 8 *bits* que le de soporte. En tu implementación debes tener al menos:

- 4 registros (A, B, C y D)
  - Registros C y D para direccionamiento
- Una unidad aritmética
- Una unidad lógica
- Saltos y subrutinas
  - Se puede saltar a la posición que apunta D en memoria, también llamarla como subrutina.

Lo esperado... Se espera un diagrama claro, completo, legible y sin abstracciones. Utiliza nodos para las bifurcaciones o saltos para los cruces entre cables distintos. **No** uses componentes que no se han visto en clases sin explicar su funcionamiento ni detallar el circuito interno. No necesitas detallar el funcionamiento de la *control unit*. Lo más importante es que la arquitectura que diseñes sea funcional.

[1] Llamada "YadrISA".

7.1. ISA

Condition Codes	Name	Description
o	Odd	Impar
e	Even	Par
c	Carry	Carry
v	Overflow	Overflow
d	Warning	Intentar dividir por 0

	Op1	Op2	Descripción
MOV	R1	R2	Coloca del valor de R2 en R1
SWP	R1	R2	Intercambia los valores de R1 y R2
ADD	R1	R2	Suma R1 con R2
NEG	R1	R2	Multiplica R2 por -1 (inverso aditivo) y lo guarda en R1
MUL	R1	R2	Toma la mitad menos significativa de los bits de cada registro y los multiplica
DIV	R1	R2	Guarda el cociente en R1 y el módulo en R2, si R2 es 0, retorna R1 y R2 normalmente y pone en 1 el condition code d.
NAND	R1	R2	R1 = NOT (R1 AND R2)
NOR	R1	R2	R1 = NOT (R1 OR R2)
SHL	R1	R2	R1 << R2
SHR	R1	R2	R1 >> R2
JMP	LIT		Salto incondicional
JCR	LIT		Salto si hay carry
JOV	LIT		Salto si hay overflow
JOD	LIT		Salto si es impar (odd)
JEV	LIT		Salto si es par (even)
MOV	R1	(C)	
MOV	R1	(D)	
MOV	(C)	R1	
MOV	(D)	R1	
MOV	R1	(LIT)	
MOV	(LIT)	R1	
JMP	D		Usa el valor guardado en el registro D como dirección de salto
JCR	D		Usa el valor guardado en el registro D como dirección de salto
JOV	D		Usa el valor guardado en el registro D como dirección de salto
JOD	D		Usa el valor guardado en el registro D como dirección de salto
JEV	D		Usa el valor guardado en el registro D como dirección de salto
CALL	LIT		Usa el valor LIT como dirección de la subrutina
CALL	D		Usa el valor guardado en el registro d como dirección de la subrutina
RET			
PUSH	A		Guarda el valor del registro A en la posición del stack apuntada por SP
PUSH	B		Guarda el valor del registro B en la posición del stack apuntada por SP
POP	A		Lee el valor del stack en la posición apuntada por SP y la guarda en el registro A
POP	B		Lee el valor del stack en la posición apuntada por SP y la guarda en el registro B
INC	SP		Incrementa el stack pointer
DEC	SP		Decrementa el stack pointer

8. Pregunta Pipelining

8.1. Parte A

Simula la ejecución del siguiente código assembly (a mano) e indica, por cada instrucción, cuando ocurren hazards; incluye la dependencia, la resolución y la respectiva forwaring unit asociada, stalling y flushing. Ante saltos **condicionales**, asume que siempre saltaremos. **Esto es sólo para la sección CODE, no incluyas la sección DATA.**

```
DATA:
    dividendo   0
    divisor     0
    cociente    0
    resto       0

CODE:
JMP main
div:                                                    // división sin signo
    MOV (dividendo),A
    MOV (divisor),B
    while_dividendo_mayor_o_igual_que_divisor:
        MOV A,(dividendo)
        MOV B,(divisor)
        CMP A,B
        JLT end_while
        SUB A,B
        MOV (dividendo),A
        MOV A,(cociente)
        ADD A,1
        MOV (cociente),A
        JMP while_dividendo_mayor_o_igual_que_divisor
    end_while:
        MOV (resto),A
        MOV A,(cociente)
        MOV B,(resto)
    JMP end
main:
    MOV A,10
    MOV B,6
    JMP div    // calcula A / B
end:
```

Importante: en lugar de representar las dependencias y su resolución con flechas (como se hace en las slides), puedes indicarlas de manera clara y ordenada **con texto** a un costado por cada instrucción.

Lo esperado... Esta pregunta apunta a que, siendo capaces de entender el assembly del computador básico, puedan detectar los hazards y resolverlos correctamente.

8.2. Parte B

Tenemos un computador Von Neumann con pipeline que tiene las siguientes etapas: IF, ID, MEM, EX y WB, en ese orden. Recuerda que en un computador Von Neumann tenemos una única memoria, que puede entregar un dato o (*exclusive or*) una instrucción en un mismo acceso a ella.

¿Qué tipos de hazards pueden ocurrir? Muéstralo con el diagrama de etapas del pipeline, indicando las dependencias y cómo se resuelven.

Lo esperado... En lugar de mostrar todas las posibles combinaciones y cómo resolverlas, esperamos que muestres los casos de manera aislada y que sean representativos (tal y como se hace en las clases).

Soluciones

IMPORTANTE: nos reservamos el derecho a no restringirnos del todo a los formatos exigidos por los enunciados al momento de mostrar una solución.

6. Pregunta ISA

1. **No.** Si bien las instrucciones más simples de RISC podrían ser unas pocas dentro de CISC, las propuestas de ambos diseños son contrarias.

   RISC me debería dar muchos más registros, tener instrucciones atómicas que en promedio tarden un solo ciclo en ejecutarse, dejar los accesos a memoria como instrucciones independientes y que las instrucciones que realicen operaciones aritméticas y lógicas sean de registro a registro.

   Por otro lado, CISC tiene muchos menos registros y da soporte a muchas más formas de operar entre direcciones de memoria, teniendo instrucciones más complejas y que tomen varios ciclos en ejecutarse.

2. Tomemos una instrucción dentro de un programa escrito con una ISA CISC, nos encontraríamos las siguientes dificultades:

   - Necesitaríamos varias instrucciones para hacer lo mismo.
   - Tenemos que tener cuidado con que el estado en que dejemos la máquina sea compatible con la instrucción siguiente (por ejemplo, no borrar un dato de un registro que justo iba a ocupar después).

   Esas son las dos dificultades principales.

   Para resolverlas, necesitamos (es una solución posible) cumplir ciertas reglas (no están ordenadas).

   - Para cada instrucción CISC, tener una subrutina RISC que la simule completamente en caso de que no haya una instrucción en RISC que haga esa misma función completamente.
   - Tener una convención para la utilización de los registros. Como RISC nos da más registros que CISC, tendremos algunos de ellos que serán a los que CISC hace alusión mientras que otros serán registros auxiliares y que entenderíamos como "no persistentes" entre dos instrucciones del CISC.
   - Si la instrucción usa literales, cargar antes el literal en un registro auxiliar y pasarlo a la subrutina como argumento.

   Ahora, el procedimiento para hacer la transformación es:

   a) Poner todas las subrutinas para las distintas instrucciones CISC.

   b) Mantenemos las `labels` y variables, adaptando la sintaxis si es necesario.

   c) Reemplazar cada instrucción por un llamado a la subrutina correspondiente o a su instrucción RISC equivalente.

   Si la instrucción CISC usa un literal, anteponemos a esta una instrucción que cargue el literal en un registro auxiliar y luego lo pasamos a la subrutina como argumento al hacer el llamado a esta.

Corrección - 6 puntos

1. **2 puntos.** El puntaje se asigna en base a la correctitud de los argumentos dados en la respuesta.
2. **4 puntos.** El puntaje se asigna en base a las consideraciones que se muestren (2 puntos) y el procedimiento de transformación entregado (2 puntos).

Se asignará menos puntaje en caso de caer en imprecisiones, datos falsos o dejar fuera aspectos importantes.

7. Pregunta Microarquitecturas

Por tiempo, no hay solución propuesta.

Corrección - 6 puntos

1. **1 punto.** Manejo de ciclos de instrucción.
2. **1 punto.** Condition codes.
3. **1 punto.** Soporte instrucciones registros y aritmética.
4. **1 punto.** Soporte para saltos.
5. **1 punto.** Soporte para subrutinas.
6. **1 punto.** Correctitud del circuito lógico.

Dar puntaje parcial según grupos de instrucciones.

Con total libertad nos referimos a total libertad. Si quieren especificar que toman la AU y la LU y las meten en una caja llamada "ALU", perfecto. Pero deben especificar todo eso. También pueden usar buses para no tener infinitos cables. Si su microarquitectura soporta más instrucciones de las pedidas o cosas como usar A y B para direccionamiento, además de C y D, entonces no se penaliza.

8. Pregunta Pipeline

8.1. Parte A

Lo tengo en esta hoja de google.

Corrección - 3 puntos.

1. **+1 punto.** Forwarding units. -0,5 si no las mencionan, se equivocan de FU o les faltaron.
2. **+1 punto.** Saltos y flush.
3. **+1 punto.** Stallings.
4. **-0,5 puntos.** Por errores menores.
5. **1,5 puntos máximo** si no ejecutan el programa y sólo trabajan sobre el código.

Se asignará menos puntaje en caso de caer en imprecisiones, datos falsos o dejar fuera aspectos importantes.

8.2. Parte B

Primero, los *hazards* de datos:

I. `MOV B,1` seguida de `ADD A,B`: en ID necesito el valor que está en MEM. Se resuelve haciendo *forwarding* desde EX/WB hacia MEM/EX.

II. `ADD A,5` seguida de `MOV (B),A`: en ID necesito el valor de A que está en EX. Se resuelve haciendo *stalling* y luego *forwarding* desde EX/WB a ID/MEM.

III. La secuencia `MOV B,1`, `NOP`, `ADD A,B`: en ID necesito el valor de B que está en EX. Se resuelve haciendo *forwarding* desde WB a ID/EX (por ejemplo).

Ahora, *hazards* estructurales:

I. Cualquier par de instrucciones seguida de algo como `MOV (4),A`: la dependencia es estructural entre IF y MEM, ya que ambas instrucciones necesitan acceso a memoria. Se resuelve haciendo *stalling*.

Por último, los *hazards* de control:

I. Ante cualquier salto condicional, necesito esperar a la salida de **EX** para determinar si debo saltar o no. Por lo que debería hacer *stalling* de 3 ciclos.

   Podría realizar una ejecución especulativa para evitar perder esos 3 ciclos y luego hacer *flush* si me equivoqué, sin embargo si en alguna de ellas escribiré en memoria, debo sí o sí hacer *stalling* hasta determinar el salto, ya que no habría cómo restaurar la memoria a su estado original en caso de que el salto sea incorrecto sin tener que gastar más ciclos en restaurarla.

Corrección - 3 puntos

1. 1 punto por cada *tipo* de hazard (datos, control y estructural).

Se asignará menos puntaje en caso de caer en imprecisiones, datos falsos o dejar fuera aspectos importantes.