Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencias de la Computación

IIC2343 – Arquitectura de Computadores

Ayudantía 6

Profesor: Yadran Francisco Eterovic Solano

Ayudante: Germán Leandro Contreras Sagredo (glcontreras@uc.cl)

Temas a tratar

Los temas a tratar dentro de esta ayudantía son:

- Paralelismo a nivel de instrucción.

Preguntas

1. a. **(Examen - I/2013)** Indique qué capacidades/intrucciones gana y pierde el computador básico al agregar soporte para paralelismo a nivel de instrucción.

   b. **(I3 - II/2016)** En caso que la única solución para solucionar un *hazard* de datos sea introducir una burbuja, ¿cuál es el momento más adecuado para hacerlo?

   c. **(Examen - I/2013)** Un computador con microarquitectura avanzada posee un *pipeline* de 30 etapas. ¿Cuál es el elemento de *hardware*, sin contar los registros entre etapas, que tiene el mayor impacto (positivo y negativo) en el rendimiento? Justifique su respuesta.

   d. **(Examen - II/2015)** Un computador x86 monoprocesador posee un *pipeline* de 5 etapas que se ejecutan en el siguiente orden:

      a) *Fetch*: Obtiene desde la memoria el *opcode* de la instrucción a ejecutar.
      b) *Decode*: Decodifica el *opcode*, enviando las señales correspondientes a cada componente.
      c) *Read*: Lee desde registros y memoria los datos requeridos para ejecutar la operación.
      d) *Execute*: Ejecuta la operación aritmética/lógica de la instrucción usando la `ALU`.
      e) *Write*: Almacena en registros o memoria el resultado de la operación aritmética/lógica.

      Además de los mecanismos tradicionales para combatir *hazards* de datos y de control, el computador evita *hazards* estructurales de acceso a memoria entre las etapas *Fetch*, *Read* y *Write*, al utilizar una memoria RAM que permite realizar de manera simultánea tres solicitudes distintas. A pesar de esto, es posible generar un *hazard* estructural de ejecución entre las etapas *Read* y *Execute*, cuando se procesa una instrucción del tipo `MOV Reg1,[Reg2+offset]`. ¿Cómo es posible evitar este *hazard*?

2. a. **(I3 - II/2016)** En promedio, ¿cuántos ciclos por salto pierde un computador con un *pipeline* de 12 etapas, si su unidad predictora acierta el 75 % de las veces? Asuma que los saltos se realizan en la penúltima etapa.

   b. **(I3 - I/2016)** Estime el tiempo de ejecución de un programa que toma N instrucciones en el computador básico con *pipeline*, en base a las siguientes condiciones:

   - El 50 % de las instrucciones realizan lecturas o escrituras en memoria.
   - El 10 % de las instrucciones son de salto y en el 25 % de estas, el salto finalmente se realiza.
   - En el 50 % de las ocasiones, el dato obtenido desde la memoria debe ser utilizado en la instrucción siguiente.

   Cualquier supuesto sobre la solución debe quedar claramente explicado.

3. **(I3 - II/2016)** Considere el siguiente código escrito para el assembly del computador básico con *pipeline*:

```
ADD A,B
MOV (var0),A
MOV (var1),B
```

   a. Indique con un diagrama los *hazards* que se generan al ejecutar el código anterior.

   b. Indique qué sucede al intercambiar el orden de la segunda y tercera línea del código. ¿Cambia el resultado final? En caso de existir *hazards*, indíquelos con un diagrama.

   c. Para los dos casos anteriores, en caso de existir *hazards*, indique como los detectan las *forwarding units* correspondientes, especificando las señales de control participantes.

4. **(I3 - I/2016)** Determine el número de ciclos que se demora el siguiente código, detallando en un diagrama los estados del *pipeline* por instrucción. El *pipeline* tiene *forwarding* entre todas sus etapas, el manejo de *stalling* es por software (instrucción `NOP`) y predicción de salto asumiendo que no ocurre. Indique en el diagrama cuando ocurre *forwarding*, *stalling* y *flushing*.

```
DATA:
    n 1
    index 1
    prev1 0
    prev2 1
    res 0
CODE:
    main:
        MOV A,(n)
        MOV B,(index)
        JEQ end
        ADD B,1
        MOV (index),B
        JMP main
    end:
        MOV A,(prev1)
        MOV B,(prev2)
        ADD A,B
        MOV (res),A
```