ARQUITECTURA DE COMPUTADORES II – Curso 2017

PRÁCTICA 2: ARQUITECTURAS SEGMENTADAS

Bibliografía de referencia:

[1] "*Computer Architecture – A Quantitative Approach*" de J. Hennessy y D. Patterson. Cuarta edición.
1. Apéndice A, secciones A.1 y A.2.
2. Capítulo 2, secciones 2.1 y 2.3.

---

1) Para estudiar la segmentación (*pipeline*) de instrucciones se pueden definir los siguientes conceptos:

- Mejora o *speedup*: La relación entre el tiempo de ejecución de *n* instrucciones sin *pipeline* respecto del tiempo con *pipeline*.
- Productividad, *throughput* o ancho de banda: Cantidad de instrucciones por unidad de tiempo.

Ahora considere un *pipeline* ideal correctamente balanceado de *k* etapas que se utiliza para procesar *n* instrucciones.

a) Deduzca las expresiones matemáticas para cada concepto.
b) ¿Como se relacionan las expresiones anteriores entre sí?
c) ¿Qué sucede si n→∞ o bien si *n* >> *k*?
d) ¿Cuál es el rango de valores que puede tomar la mejora? ¿y la productividad?

2) Suponer un *pipeline* genérico de 5 etapas, con las siguientes latencias de cada etapa: *S1 = 50ns, S2 = 60ns, S3 = 80ns, S4 = 90ns y S5 = 60ns*. Considere que los registros entre etapas agregan una latencia adicional de *10ns*.

a) Responda con sus palabras: ¿Por qué son necesarios los registros intermedios?
b) ¿Cuál es la máxima frecuencia de reloj que se puede aplicar a este *pipeline*?
c) Calcular la mejora y la productividad para un conjunto de 10, 50 y 100 tareas. Interprete los resultados.
d) ¿Qué sucede con estos índices a medida que el número de instrucciones crece?
e) ¿Está correctamente balanceado este *pipeline*? ¿En qué etapa conviene invertir esfuerzo optimizando la configuración si se desea aumentar la frecuencia máxima de operación?

3) Suponga un pipeline genérico de *k* etapas se utiliza para ejecutar n tareas que toman *T* segundos cada una. Los registros intermedios tienen un tiempo de setup T_L. Determine la expresión de la latencia por etapa y de la mejora en función de *T*, T_L, *n* y *k*. Demuestre que la profundidad óptima (que maximiza la mejora) es el valor entero más próximo a:

$$k_{opt} = \sqrt{(n-1)\,T / T_L}$$

Determine la mejora máxima que se logra si n→∞. Compare con los resultados del ejercicio (1): ¿por qué la mejora máxima calculada con este modelo ya no es *k*?

4) Se tienen tres propuestas de organización para un procesador, una de 2 etapas, otra de 3 etapas y otra de 5 etapas. Suponga que la organización de 2 etapas admite un período de reloj de 60ns, la de 3 etapas una 45ns y la de 5 etapas 30ns.

a) ¿Cuántos ciclos de reloj requiere cada implementación para ejecutar una secuencia ininterrumpida (sin saltos ni dependencias) de 10 instrucciones? ¿Cuántos CPI promedio se logran en cada caso?
b) ¿Cuál es la mejora que presenta cada propuesta respecto de una versión no segmentada del mismo procesador, si se sabe que esta última admitiría una frecuencia de reloj máxima de 10MHz (período 100ns)?
c) Repita los incisos anteriores si la secuencia a ejecutar es de 1000 instrucciones.
d) ¿Qué índice es el más relevante para juzgar las alternativas, CPI o mejora?
e) ¿Cuál es la mejora límite que se podría obtener con cada una de estas organizaciones?

5) En la referencia [1] se describen las etapas más importantes en la ejecución de instrucciones de un procesador MIPS y les da los nombres IF, ID, EX, MEM, y WR. Se cuenta con una implementación del procesador MIPS no segmentada y óptima (no ejecuta etapas que no son necesarias para una dada instrucción), y que avanza una etapa por ciclo.

a) ¿Cuántos ciclos toma la ejecución de un almacenamiento en memoria? ¿Cuántos un salto condicional? ¿Cuántos las demás instrucciones?
b) Dibuje el diagrama de estados simplificado que representa la evolución de la ejecución de las instrucciones en función de su tipo (almacenamiento, salto, otras). Cada estado será una de las etapas de ejecución.
c) ¿Cuántos ciclos le tomará a este procesador ejecutar el programa *fibonacci.s* que se descarga junto con esta práctica? ¿Cuántas instrucciones se ejecutan en ese intervalo? ¿Cuál es el valor de CPI alcanzado?
d) ¿Cuántos ciclos tomará la ejecución del mismo programa si se reemplaza la implementación por un procesador MIPS segmentado? ¿Cuál es el valor de CPI alcanzado en este caso? Para este cálculo simplificado desprecie la existencia de detenciones debidas a riesgos del pipeline.
e) ¿Cuál es la latencia de las instrucciones en esta versión segmentada del procesador? ¿Hay diferencias entre unas clases de instrucciones y otras? ¿qué ocurre con las instrucciones que no requieren de todas las etapas de ejecución?

6) Considere las organizaciones de los pipelines de tres arquitecturas diferentes: microcontroladores PIC (2 etapas, IF-EX), ARM v7 (3 etapas, IF-ID-EX) y MIPS clásico (IF-ID-EX-MEM-WR) . Para cada una:

a) Defina las actividades que debe realizar el procesador en cada etapa.
b) Represente en el diagrama de tiempos en qué etapa se encuentra cada instrucción en cada ciclo de reloj.
c) Represente también en otro diagrama la ocupación de etapas en función del tiempo. Eso es, qué instrucción ocupa cada etapa en cada ciclo de reloj.

"You may find it hard to believe that pipelining is as simple as this; it's not."

Hennessy & Patterson

---

7) Dependencias y riesgos de segmentación:

a) Defina, con sus palabras, los diferentes tipos de dependencia que existen entre instrucciones: dependencia de datos y dependencias de control.
b) Defina los diferentes tipos de riesgos de segmentación que existen: riesgos de datos, riesgos de control y riesgos estructurales.
c) Analice la siguiente oración ([1], pag. 70): "*Dependences are a property of programs. Whether a given dependence results in an actual hazard being detected and whether that hazard actually causes a stall are properties of the pipeline organization.*" ¿Qué relación existe entre las dependencias, los riesgos de segmentación y las detenciones de *pipeline*?

8) Riesgos de datos.

a) Defina, con sus palabras los tres tipos de riesgos de datos que pueden aparecer en un pipeline: RAW, WAR, WAW.
b) ¿Cuál de estos últimos corresponde a una dependencia verdadera? ¿a qué se debe esta denominación?
c) ¿Cuáles corresponden a dependencias de nombre? ¿cuál es una antidependencia y cuál una dependencia de salida?
d) ¿Qué medidas pueden llevarse a cabo para minimizar el impacto de este tipo de riesgos?
e) ¿Por qué no ocurren riesgos de tipo WAW ni WAR en un *pipeline* simple con ejecución estrictamente ordenada como el de los primeros RISC?

f) Dado el siguiente programa indicar qué tarea realiza y cuáles son las dependencias/riesgos entre instrucciones existen en el fragmento:

```
MOV   R3, R7
LW    R8, (R3)
ADD   R3, #4
LW    R9, (R3)
```

9) Riesgos estructurales.

a) Defina las causas de los riesgos estructurales. ¿Qué medidas se puede tomar para evitarlos?
b) Una implementación simple de MIPS no cuenta con memorias de datos y programa separadas, por lo que las etapas IF y MEM del pipeline deben turnarse para compartir el acceso. Señale los riesgos estructurales presentes en el fragmento de programa que está al final del ejercicio y que esta organización del procesador hará evidentes.
c) Para el mismo procesador del inciso anterior: dibuje el diagrama de tiempos de la ejecución del fragmento de código. ¿Cuántas detenciones del pipeline serán necesarias para resolver los riesgos estructurales?
d) Para el mismo procesador del inciso anterior: despreciando el impacto de cualquier otro tipo de riesgos, ¿cuál sería el incremento en los CPI promedio de ejecución de un programa ocasionado por la utilización de esta organización, si se sabe que en promedio el 20% de las instrucciones ejecutadas son cargas o almacenamientos?.

```
LW    R2, 0(R4)
LW    R3, 4(R4)
DADD  R3, R2, R3
SW    R3, 8(R4)
DADDI R4, R4, 4
DADDI R5, R5, -1
BNEZ  R5, loop
```

10) Riesgos de control.

a) Explique con sus palabras las medidas que se pueden implementar para minimizar el impacto de los riesgos de control:
    i. Técnicas estáticas:
        1. Pipeline freeze.
        2. *Predict-untaken*.
        3. *Predict-taken*.
        4. *Delayed branch*.
    ii. Predicción dinámica:
        1. Predictor de 1 bit.
        2. Predictor de 2 bits.
b) Los procesadores SPARC y MIPS son algunos de los que utilizan la técnica de *delayed branch*. Esta técnica presenta varias dificultades, pero una crítica particular que se les puede hacer es que hace visible en la ISA una característica propia de la organización del *pipeline* del procesador. Analice brevemente qué restricciones impondría esto sobre una arquitectura que tuviera la misma ISA de estos procesadores pero con una organización de pipeline diferente de la convencional (por ejemplo, que tuviera mayor cantidad de etapas).
c) ¿En términos cualitativos, por qué en general funciona mejor la predicción dinámica de 2 bits que la de 1 bit? Represente y explique las máquinas de estados que regulan el funcionamiento de ambos predictores.

11) Considere una secuencia de instrucciones de longitud n que atraviesa un pipeline de instrucciones de $k = 5$ etapas. El procesador utiliza predicción estática de saltos *predict-untaken*, y el resultado de un salto condicional recién se determina en la última etapa del pipeline. Sea $p = 0.2$ la probabilidad de encontrar una instrucción de salto condicional o incondicional, y sea $q = 0.66$ la

probabilidad de que la instrucción de salto provoque la captación de una instrucción no consecutiva (se toma el salto)[1].

a) Calcule la cantidad de ciclos necesarios y los CPI resultantes si se ejecutan *n* = 10000 instrucciones. Suponga que solamente los riesgos de control producen incrementos en la latencia de ejecución.
b) Determine el nuevo valor de CPI y la mejora respecto del caso (a) si se modifica la arquitectura para que los saltos condicionales sean resueltos en la segunda etapa del pipeline en lugar de en la última.
c) Con un importante esfuerzo adicional se modifica nuevamente la organización de (b) para que utilizar predicción de tipo *predict-taken*. Determine el nuevo valor de CPI y la mejora respecto del caso (a).

12) Para realizar este ejercicio debe instalar el programa WinMIPS64 que se encuentra disponible en la página de la cátedra. Descargue también el programa *fibonacci.s* que se encuentra en el mismo sitio. Como puede anticipar por el nombre el programa calcula los primeros términos de la secuencia numérica de Fibonacci, que es aquella donde:

f[0] = 1
f[1] = 1
f[n+2] = f[n+1] + f[n]

1. Abra el archivo *fibonacci.s* con un editor de texto. Examine el código y trate de familiarizarse con su funcionamiento. Haga una lista de las dependencias existen entre instrucciones en el código.
2. Cargue el programa *fibonacci.s* en el simulador. Configure el simulador para que no utilice ni *Forwarding,* ni *Branch Target Buffer* (predicción de saltos dinámica de 1 bit) ni *Delay Slot* (menú *Configure*).
3. Ejecute el programa paso a paso con la tecla F7, observando las detenciones del pipeline. ¿Entre qué instrucciones ocurren? ¿A qué se debe cada una, riesgos de datos, de control o estructurales?
4. Ejecute el programa hasta su finalización con la tecla F4 y tome nota de la cantidad de ciclos que son necesarios y el CPI promedio logrado. Anote también la cantidad de detenciones de cada tipo ocurridas. Todos estos datos los encontrará en la ventana *Statistics*.
5. En la configuración ahora habilite la opción de *Forwarding*, y ejecute nuevamente el programa paso a paso con la tecla F7. ¿Cuales detenciones se resuelven completamente? ¿Qué detenciones quedan y a qué se deben? Complete la ejecución del programa con la tecla F4 y tome nota de las mismas estadísticas del punto (4).
6. Llegados a este punto, la organización del pipeline ya nos dió todo el rendimiento del que es capaz pero todavía es posible mejorar la *performance* con ayuda del compilador (representado por usted). ¿Qué dependencias son causantes de las detenciones restantes? ¿de qué tipo son? ¿son inevitables?
7. Modifique el programa invirtiendo el orden en que se realizan "*daddi R4,R4,4*" y "*daddi R5,R5,-1*". Convénzase de que este cambio no modifica el resultado del programa porque no hay ninguna dependencia entre estas instrucciones.
8. Cargue el nuevo programa, ejecute, y tome nota de las estadísticas de ejecución. Note que el tiempo de ejecución mejora porque eliminamos una detención que era necesaria para resolver un riesgo, y eso gracias a que separamos las instrucciones causantes de la dependencia que le da origen, DADDI y BNEZ (¿recuerda la relación entre dependencia, riesgo y detención de la que hablamos en algún ejercicio anterior?).
9. Aún queda una detención por datos más. Encuentre otra forma de ordenar las instrucciones del bucle para que elimine también esta detención y obtenga las nuevas estadísticas de ejecución.
10. Finalmente, una combinación de habilidad del compilador y modificación en la organización nos puede dar medio pancito más de rendimiento. En la configuración del *pipeline* habilite *"Enable Delay Slot"*, dejando habilitado *"Enable Forwarding"*. ¿Qué ocurre si en estas condiciones se ejecuta el programa sin modificaciones? Modifique el programa de forma tal

1 Dichos valores de probabilidad corresponden a la estadística de los programas pertenecientes a una versión de SPEC.

que se ejecute en 78 ciclos (o si lo logra, en menos). ¿Todas las instrucciones ejecutadas de esta forma son "útiles"?

11. Fundamente: ¿podríamos ahora lograr rendimiento adicional desenrollando el bucle?
12. Confeccione una tabla que contenga las estadísticas recopiladas para cada variante de la organización y del programa. Agregue en la misma tabla la mejora de cada versión respecto de la primera de todas.

NOTA: El WinMIPS64 es un programa un tanto temperamental, y cada tanto luego de cerrarse intempestivamente debido a un error luego ocurre que se niega a arrancar. Cuando eso suceda elimine el archivo "*winmips64.las*" que se encuentra en la misma carpeta que el ejecutable y todo debería funcionar nuevamente.

13) Ejecute el programa *hailstone.s* que se puede encontrar junto al WinMIPS64. El programa calcula el máximo valor de las secuencias *hailstone* correspondientes a los primeros 100 números naturales. La secuencia *hailstone* ("granizo") es una secuencia definida como:

si h[n] = par,	h[n+1] = h[n] / 2
si h[n] = impar,	h[n+1] = 3 * h[n] + 1
si h[n+1] = 1,	Termina la secuencia.

que es una secuencia que se sabe siempre es de largo finito (eventualmente siempre alcanza el valor 1). Por ejemplo, la secuencia correspondiente al número 5 es "5, 16, 8, 4, 2, 1" y el máximo es 16.

1. Abra el archivo *hailstone.s* con un editor de texto. Examine el código y trate de familiarizarse con su funcionamiento. Observe qué dependencias existen entre las instrucciones, y la estructura de bucles que compone el algoritmo.
2. Ejecute el programa sin *forwarding*. ¿Cuántos ciclos toma? ¿Cuántos de esos fueron detenciones? ¿Cuál es el mínimo de ciclos necesarios para ejecutar este programa, suponiendo que lograra eliminar todas las detenciones? Este número es la cota de lo que podemos lograr optimizando la ejecución sin modificar el algoritmo.
3. Habilite la opción "*Enable Forwarding*". ¿Cuál es la mejora que se obtiene gracias a esta mejora en la organización del pipeline?
4. Finalmente intente mejorar aún más la *performance* reordenando instrucciones; ¿tienen impacto significativo los cambios posibles? ¿Qué limitaciones encuentra? Explíquelo en términos de los tipos de dependencia que observa en el programa.
5. Opcional. Intente ahora optimizar el programa modificando el algoritmo si es necesario (todo vale, siempre que el resultado sea correcto). ¿En cuántos ciclos puede calcular los primeros 100 máximos? Opine: ¿le parece que el compilador sería capaz de lograr este tipo de mejora?