Contents

1 Cache

1.1 Interleaved memory

Se van intercalando direcciones de memoria en distintos bloques, no de manera consecutiva.

De esta manera no hay que esperar tiempos de ciclo

1.2 En los sistemas reales

- Se aumenta ancho de banda
- Se usa Int. Mem

2 Practico 5

2.1 Ej 1

- rapido, + caro

2.1.1 Freq acceso

El sistema de memoria que mayor numero de accesos tiene más frecuentemente se accede

2.2 Ej 2

2.2.1 Howto

1. Fer frec. clock
2. Ver tiempo acceso memoria
3. Pasar frec. a tiempo

$$T_{clk} = \frac{1}{frec}$$

4. Dividir tiempo de acceso por T_{clk}

2.3 Organizacion cache

2.3.1 Areas

- Seccion de datos
  - Guarda los datos en bloques (M words)
- Bit de validacion
- Seccion de tag/etiqueta
  - Se usa como key (Como en un diccionario)

2.3.2 Diferencia con la ram

A diferencia de la ram, puede que ninguna entrada coincida con el tag

2.4 Ej 3

2.4.1 Howto

2.4.1.1 Como saber tamaño area de datos?

"Cache de mapeo directo de **16KiB**"

El texto en negrita es el tamaño del area de datos

2.4.1.2 Correspondencia directa (CHEQUEAR)

- Se guarda el numero de bloque en el tag, y de ahí hay correspondencia directa entre la memoria de ese bloque y la cache.
- No es economicamente viable, ya que si se acceden a distintos bloques hay muchos miss

2.4.1.3 Correspondencia asociativa (???)

2.4.1.4 Bits area tag

$$bitsTag = \frac{numBloques}{numLineas}$$

2.4.1.5 Cantidad bloques

$$\frac{2^N}{M}$$

2.4.1.6 Calcular tamaño total

capTotal = lineas * anchoBloque

CapTotal = tamAreaDatos

AnchoBloque = cantBloques * tamBloque

- Despejar lineas

2.5 Ej 4

2.5.1 Notas

- Anotar datos enunciado
  - bloques
  - palabras por bloque
  - tamaño address
- Palabras del cache SIEMPRE hace referencia a palabras del procesador
- Ver cuantas palabras caben en area de datos
- Si hay más de una sola palabra en el area de datos hacen falta más bits para direccionar
  - Si esto sucede pueden ignorarse algunos de los bits menos significativos. A esto se le llama "direccionamiento de palabra"
  - En este ej no hay direccionamiento de palabra porque solo es una palabra la que se usa
- Hacerse la pregunta "Cuantas veces entra la cache en la memoria principal organizada de a bloques"
  - $tamTag = \frac{2^{addrSize}}{tamaño_{bloque}}$??
- Separar el address en tag, index y offset byte
- El index se usa para ver que entrada de cache usar
- Numeracion de las word: w3, w2, w1, w0.
  - Va de más grande a más chico

2.5.2 Howto

- $CantBloques = \frac{tamMP}{\frac{len(w_p)}{len(w_m)}}$

1. Separar la address en los siguientes campos:
   - tag: $tamTag = \frac{cantBloques}{cantLineasCache}$??
   - index: cantidad lineas cache
   - wordSelector: Solo si se guarda más de una word en la cache
   - offset byte: $cantBytes = \frac{tamPalabra}{2^3}$
2. Cargar/leer datos de la linea denotada por el index
3. Setear tag y word

2.6 Ej 5

2.6.1 Howto

- Para ver tamaño address ver cuantos bits hace falta para direccionar la memoria principal
  - Lo mismo para determinar tamaño de index

2.7 Ej 6 (VER TEORICO)

2.7.1 Howto

- Si es en cuatro vias dibujamos 4 tablas
- El campo set se calcula segun cuantos conjuntos hay que direccionar
  - Si son 16 conjuntos hacen falta 4 bits

2.8 Ej 8

- Escribir datos principales de la memoria y cache: cantPalabras, cantBloques, cantLineasCache
- El tag indica la n-esima vez que entra el cache en la memoria principal
- Para chequear si el address está bien ver si direcciona la MP
- Para calcular palabras por bloque: $\frac{cantPalabras}{cantBloques}$

2.8.1 a) Correspondencia directa

- tag: $\frac{cantBloques}{cantLineasCache}$
- Index: cantLineasCache
- W: CantPalabrasPorBloque = $\frac{cantPalabras}{cantBloques}$

2.8.2 b) Full asociativa

- w: cantPalabrasPorBloque
- tag: $\frac{cantPalabrasMP}{cantPalabrasPorLineaCache}$

2.8.3 c) Asociativa por conjuntos

- set: $\frac{cantLineasCache}{cantVias}$
- w: cantPalabrasPorBloque
- tag: $\frac{cantBloquesMP}{cantLineasPorVia}$

2.9 Ej 9

$$AMAT = hitTime + missRate * missPenalty$$

$$ClockCycles = CPI + \frac{memAccesses}{instructionCount} * missRate * missPenalty$$

2.9.1 9d

Hay que calcular para stall-data y stall-instructions y luego sumar

$$CPI_prom = CPI + memStallData + memStallInst$$

$$memStallClockCycles = \frac{memAccesses}{instructionCount} * missRate * missPenalty$$

2.10 Ej 10

- FrecClock = 1/HitTime
- MissPenalty = tiempo acceso mem principal

3 Tecnicas de prediccion de saltos

3.1 "Operador ternario" para evitar saltos

En vez de hacer un salto, se pueden usar instrucciones que funcionan de manera similar a un operador ternario.

Este cambio lo realiza el compilador.

3.2 BTFNT

Backward taken forward not taken.

Si se salta para atras se asume taken y para adelante not taken

3.3 Predictores locales

Predicen en base a predicciones anteriores del mismo salto

3.3.1 Predictor de 2 bits

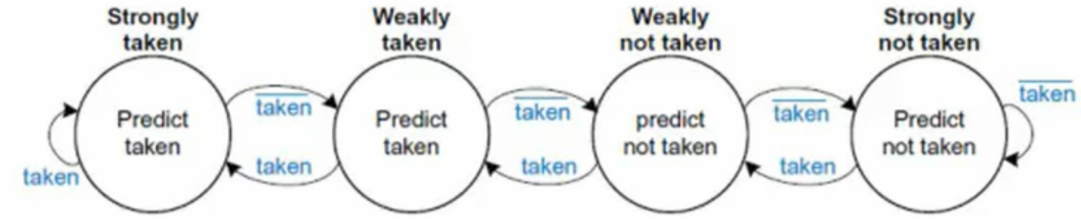


Figure 7.62 Two-bit branch predictor state transition diagram

3.4 Predictores globales

3.4.1 Predictor de dos niveles

- Realizan la prediccion usando los ultimos n saltos ejecutados
- Usan un shift register (GR) que guarda si los ultimos n saltos fueron Taken (1) o Not Taken (0)

3.4.1.1 PHT

- PHT: pattern history table
- Se indexa con GR + PC
- Guarda dos bits que siguen la misma logica que el predictor de 2 bits

3.4.2 Predictor gshared

- Usa GR y tabla PHT como en el predictor de dos niveles
- Hace un XOR entre el GR y los ultimos bits del PC

3.4.3 Predictor por torneo

- Usa predictores globales o locales segun el tipo de instruccion en base a un selector (el cual tambien "se entrena")
- Tiene la ventaja de que usa el mejor predictor posible para cada caso

3.4.4 Tagged hybrid predictors

- Usa muchos predictores de dos niveles pero con distintos tamaños de GR

4 Practico 6

4.1 Ej 1

- $LatEtapa = tiempoCiclo(Tc) = \frac{tiempoTotal}{N} + penalidadRegistroPipeline$
- $CPI = 1.23 + 0.1 * (N - 5)$
- $tiempoPorInstruccion(Ti) = Tc * CPI$

4.2 Ej 2

- La penalidad si se hace un fetch incorrecto es de N

$$cantCaminoCorrecto + fetchPenalty * missedFetches$$

$cantCaminoCorrecto = \frac{cantInst}{W}$

4.3 Ej 3

$$precision = \frac{aciertos}{total}$$

4.3.1 b)

- Hacer tabla con patron, prediccion y fallo o acierto

4.4 Ej 4

4.4.1 a)

- Pensar solo en el contexto de la instruccion
- Se puede predecir el salto que realiza una instruccion si el numero que usa es aleatorio?

4.4.2 b)

- Hay algun if que si o si se ejecute si los otros dos se ejecutan?

4.5 Ej 6

- Solo hay forwarding entre issue packets, no hay forwarding interno entre dos instrucciones

4.5.1 Tips:

4.5.2 a)

- Hacer tablita pipeline
- Hacer mismo procedimiento que con pipeline
- No se tiene en cuenta WAW porque es 1-issue

4.5.3 b)

- Hacer tablita

ALU or Branch | Data | Clk |

4.5.4 c)

4.5.4.1 Loop unroling

- Duplicar el codigo para loopear menos veces y poder emparejar mejor las instrucciones

4.5.4.2 Tener en cuenta

- Dependencias de nombres no pueden ir juntas
- Dependencias de datos no pueden ir juntas
- Si hay dependencias de datos hay que hacer stall
- Si se lee y despues se escribe un registro pueden ir juntos. (Pensarlo usando la tabla)

5 Static multiple issue

- Se complejiza más el micro para ejecutar dos instrucciones a la vez
- Se arma un camino para las instrucciones que acceden a memoria y otro para las que no

5.1 Como se agrupan las instrucciones paralelas?

- Se agrupan en *issue packets*
- Se suelen agrupar instrucciones tipo R o Branch y instrucciones de acceso a memoria
- Las instrucciones se leen de a dos (64 bits)
- Si uno de los issue no se puede usar, se lo acompaña con un nop
- Compilador revisa dependecias de datos en un issue packet
- No se puede hacer forwarding dentro del mismo issue packet
- Asumimos que no hay dependencias de memoria (se encarga el hardware), solo de registros

5.2 Tipos de dependencias de datos

5.2.1 Dependencia real de datos

Es lo que veniamos haciendo en pipeline

5.2.2 Dependencia de nombre

Cuando dos instrucciones del mismo Issue Packet escriben el resultado en el mismo registro

5.2.2.1 Como se soluciona?

Usando registry renaming

5.3 Hazards

5.3.1 RAW (Read After Write)

Es causada por la dependencia real de datos

5.3.2 WAW (Write After Write)

Es causada por la dependencia de nombre

5.4 Dependencias de control

Cuando tenemos un branch y luego una instruccion normal, la instruccion se ejecuta por más que el branch tenga que hacer un salto

5.4.0.1 Casos

- Branch + instruccion
- Instruccion que depende de un salto + instrucccion que no depende

5.4.0.2 Solucion

Emparejar con nop

6 Dynamic scheduling

6.1 Definicion

Hardware reordena las instrucciones para evitar stalls y manteniendo flujo de datos y exception behaviour

6.2 Unidades funcionales (FU)

Hardware que realiza una operacion

6.2.1 Ejemplos

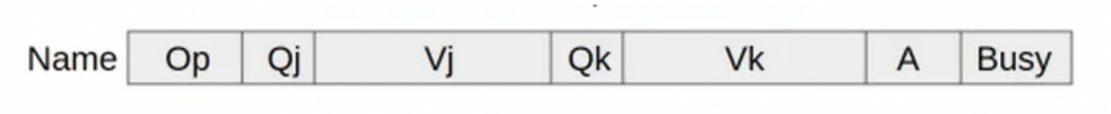
- ALU
- Memoria
- Multiplicador
- Etc

6.3 Reservation Stations

Registros asociados a una Unidad Funcional que guardan:

- Operacion
- Operandos necesarios

6.3.1 Campos



- Op: operacion a realizar
- Qj, Qk: Tag de la RS que va a producir el operando necesario
  - Un valor de 0 indica que ya se encuentra disponible
- Vj, Vk: Valores de los operandos
  - Para loads, Vk guarda el offset
- A: contiene el resultado del calculo de direccion
  - Aplica solo a load o store
- Busy:
  - 1: cuando se está ejecutando
  - 0: cuando está en espera

6.4 Algoritmo de Tomasulo

6.4.1 Idea

- Captuar operandos apenas estan disponibles para evitar recurrir a registros

6.4.2 Que pasa si hay dependencias de datos?

Se designa la RS que producira el operando necesario

6.4.3 Load y store

Para evitar complicaciones, asumimos que siempre se ejecutan en orden y que se frena la ejecucion si hay dependencia de datos

6.5 Etapas Tomasulo

6.5.1 Fetch (IF)

- Se levantan instrucciones de memoria
- Se colocan en la FIFO

6.5.2 Issue y Decode (IS/ID)

Se decodifican las instrucciones.

6.5.2.1 Si hay RS

- Se le pasa la instruccion y operandos

6.5.2.2 Si no hay RS

- Hace stall hasta que haya una libre

6.5.2.3 Si los operandos no estan en registros

- Se linkea con la RS que lo producira
- La instruccion queda en espera hasta poder ejecutarse

6.5.3 Execute (Ex)

6.5.3.1 Cuando todos los operandos estan listos

- La instruccion se ejecuta en la UF correspondinte

6.5.3.2 Loads y stores

- Requieren que se calcule la direccion de memoria antes de poder acceder a ella
- La direccion calculada se almacena en orden en el buffer correspondiente usando orden de ejecucion del programa
  - Esto previene hazard de memoria

6.5.4 Write result (Wb)

6.5.4.1 Que es el CDB?

Es el Common Data Bus y se encarga de escribir el resultado en los lugares necesarios

6.5.4.2 Cuando el resultado está disponible

Se usa la CDB para escribirlo en:

- Los registros
- Las RS que lo necesitan

6.6 Dependencias de datos

6.6.1 Real dependencia de datos (RAW)

Instr1 produce dato usado por instr2

6.6.2 Dependencias de nombre

- Dos instrucciones usan mismo registro o posicion de memoria
  - Sin flujo real de datos

6.6.2.1 Dependencia de salida (WAW)

- Instr1 e Instr2 escriben misma posicion de memoria o registro
- El orden original se debe preservar para que el valor final sea correcto

6.6.2.2 Antidepenencia (WAR)

- Instr1 escribe registro o posicion de memoria que Instr2 necesita leer
- El orden debe preservarse para asegurar que se lea el dato correcto

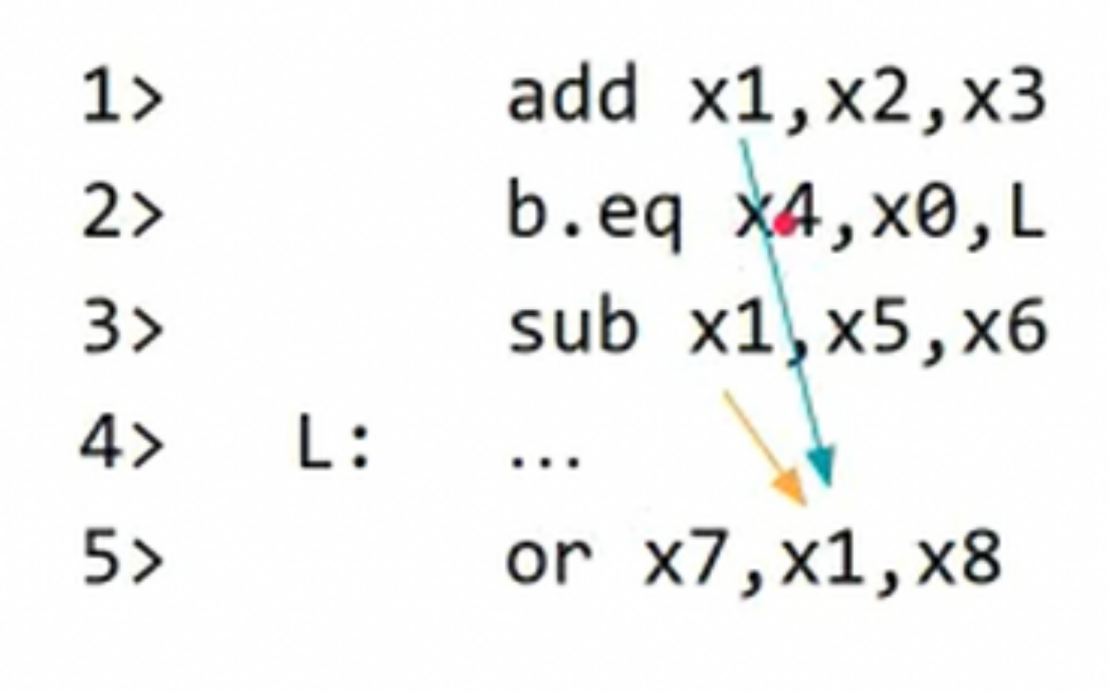
6.6.2.3 Register renaming

- Soluciona dependencias de nombre cambiando el nombre de los registros usados

6.6.3 Dependencia de datos condicional

- El orden del programa establece que instr1 genera el dato necesario para instr2
- Por más que haya dos dependencias solo se puede dar 1 hazard
- Cual hazard se genera depende del salto condicional

6.6.3.1 Ejemplo



6.7 Depenencias de control

Todas instrucciones cuya ejecucion depende de un salto tienen una dependencia de control con dicho salto

7 Practico 7

7.1 Ejercicio dependencias

7.1.1 Dependencia datos

Clasificar en:

- WAW
- RAW
- WAR

Si depende de un salto agregar que es condicional

7.1.2 Dependencia de control

Todas las instr que dependan del salto

7.2 Ej 1

7.2.1 Denotar valor dentro de un registro

Se usa [x2]

7.2.2 Notas

- Calcular el address para stur y ldur toma un clock
- El Wb toma 1 clk
- Solo un load/store puede estar busy a la vez

7.2.2.1 Ejemplo

Si hay un salto al final del programa que va hacia el inicio todas las instrucciones del medio tienen dependencia de control con esta ultima instruccion

8 Howto Tomasulo

Cargar datos en tabla Stage:

8.1 IF (1 clk)

- Toma 1 clk

8.2 IS (1 clk):

- cargar datos instruccion en RS
  - Siempre se elige la primer libre
  - Si dato no está, cargar tag en Q
- cargar datos Register status para el registro target
- No se puede alterar el orden al pasar de IF a IS
  - No puede pasar la instr8 antes que la instr5

- Si justo se esta escribiendo un registro que necesita una nueva instruccion de issue se pone en el V, pero se marca como que no se puede usar aun
- Si dos instrucciones modifican el mismo registro ponemos la ultima instruccion en el tag del Register Status

8.3 Ex:

8.3.1 Load/store

- Load/Store son secuenciales
  - Solo se ejecuta 1 a la vez, no en paralelo (no store al mismo tiempo que load)
- Calcular Addr (1 clk)
  - Se pone solo el offset primero en A y luego se pone [Xi + offset]
  - No se marca como Busy

8.3.2 Caso normal

- Ejecutar instruccion (Clk depende del tipo de RS)
  - Marcar como busy
- Instrucciones con FU distintas pueden correr en paralelo, pero si son del mismo tipo no
- Si es un Branch, no se hace más issue hasta que termine

8.4 Wb (1 clk):

- Borramos datos RS, pero marcamos como que aun no se puede usar
  - Esto lo hacemos porque la escritura del dato está en proceso
- Borramos tag Register Status, pero marcamos como que aun no se puede usar
  - Reemplazamos en las RS que lo usan, pero marcamos como que aun no se puede usar
  - Reemplazamos los Q por 0

9 Preguntas

- Cuando hay varias words en una linea de cache, cuando se guarda data en el cache, guarda solo la word que marca la address o guarda todo?
  - Respuesta: se guarda todo el bloque siempre
- El wordSelector es big endian o little endian?
  - Respuesta: Mas significativo a la izquierda
  - Osea, se usaria el siguiente orden: w3, w2, w1, w0

9.1 Como funciona/se calcula el resultado usando la PHT?

Supongamos que se inicializa toda la PHT en 0b10 (Weakly taken) A medida que se va ejecutando la instruccion se compara la prediccion con el resultado real (T o NT) y en base a eso se va actualizando el contenido de la PHT (Usando la maquina de estados)

9.1.1 Ejemplo

```
for (i=0; i < 100; i++) {
    for (j=0; j< 3; j++) {
        ...
    }
}
```

```
0x00: add x0, xzr, xzr
0x04: L2: add x1, xzr, xzr
0x08: L1: ...
0x0C: addi x1, x1, 1
0x10: cmpi x1, 3
0x14: b.lt L1
0x18: addi x0, x0, 1
0x1C: cmpi x0, 99
0x20: b.lt L2
```

(Tomamos ultimos 4 bits del PC)

GR	PC	PHT	Taken?
0000	0x4	10	True
0001	0x4	10	True
0011	0x4	10	False
0110	0x0	10	True
———	———	———	———
1101	0x4	10	True
1011	0x4	10	True
0111	0x4	10	False
1110	0x0	10	True
———	———	———	———
1101	0x4	11	True
1011	0x4	11	True
0111	0x4	01	False
1110	0x0	11	True
———	———	———	———
1101	0x4	11	True
1011	0x4	11	True
0111	0x4	00	False
1110	0x0	11	True

10 P (A) 2023

10.1 1)

10.1.1 Calcular tag

- $\frac{tamMemoriaB}{tamCacheB}$
  - Hay que agregar bytes, no poner por ejemplo 2^{64} si no $2^{64}B$
- $\frac{cantBloques}{tamIndex}$
  - $cantBloques = \frac{cantWmMemPrincipal}{cantWmPorLineaCache}$

10.1.2 Llenar tabla

- Desglosar addr en los campos y rellenar
- Si hay dos index iguales se pisan
- Solo hay hit si coinciden tag e index con un elemento ya existente

10.1.3 Calcular AMAT

$$AMAT = hitTime + missRate * missPenalty$$

10.1.4 Calcular tiempo hit (REPASAR)

- $\frac{1}{1Ghz}$
- $1/hz = s$

10.1.5 Consideraciones Asoc Conjuntos

- Lineas por via = $\frac{lineasCache}{cantVias}$
- Para determinar la cantidad de vias ver cuantas veces se sobreescribe una linea en la cache
  - En AC no se sobreescriben lineas
- Se duplican cantidad words por bloque
  - Por ende, se duplica tamaño bloque
  - Esto depende de la cantidad de vias, en este caso 2, por eso se duplica

10.2 2)

10.2.1 a)

- No importa distancia entre instrucciones, pueden ser dependencias pero no hazard

10.2.2 b)

- Si una instruccion es destino de un salto, no puede ir junto a otra en el mismo issue packet
- Los branch suelen ir solos acompañados con un nop, salvo algunas excepciones

10.2.3 c)

- Tener en cuenta que una vez se llega a al ultimo issue, se demoran 5 ciclos más en terminarse de ejecutar todas las instrucciones pendientes

11 P2022 B

11.1 Ej 2

Tag asoc conjuntos:

- tag $= \frac{\frac{MPSize}{WpSize}}{cantLineas}$
- tag $= \frac{MPSize}{CacheSize}$