Ejercicios T4

Ejercicio 2.

Agregue instrucciones NOP al código siguiente para que se ejecute correctamente en una canalización (pipeline) que no maneje los peligros de los datos.

addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
add x15, x13, x12

La razón por la que se necesitan las instrucciones NOP es porque estas instrucciones no realizan ninguna operación, pero ocupan un ciclo de reloj en la CPU. Estas instrucciones se utilizan para rellenar el espacio entre las instrucciones que realmente hacen algo, lo que permite que la CPU tenga suficiente tiempo para obtener los datos necesarios y prepararse para la siguiente instrucción que sí hace algo. En este código, se necesitan instrucciones NOP para garantizar que la instrucción addi x14, x11, 15 no toma los valores no actualizados de los registros x11 y x12, los cuales fueron actualizados por la instrucción add x13, x11, x12.

```
addi x11, x12, 5
nop
nop
add x13, x11, x12
addi x14, x11, 15
nop
add x15, x13, x12
```

Ejercicio 1.

En este ejercicio, examinamos cómo el pipeline afecta el tiempo de ciclo de reloj del procesador. Los apartados de este ejercicio suponen que las etapas individuales de la ruta de datos tienen las siguientes latencias:

Instrucción	if	id	ex	mem	wb
Latencia (ps)	250	350	150	300	200

Procesador	ALU	Jump	Load	Store
Porcentaje	45%	20%	20%	15%

¿Cuál es el tiempo de ciclo de reloj en un procesador canalizado y no canalizado?

En un procesador no canalizado, el tiempo de ciclo de reloj es la suma de todas las latencias:

```
Tc = 250 + 350 + 150 + 300 + 200 = 1250 ps
```

En un procesador canalizado, el tiempo de ciclo de reloj es el tiempo de la etapa más rápida:

```
Tc = max(250, 350, 150, 300, 200) = 350 ps
```

Por lo tanto, el tiempo de ciclo de reloj en un procesador canalizado es mayor que en uno no canalizado.

¿Cuál es la latencia total de una instrucción lw en un procesador canalizado y no canalizado?

La latencia de una instrucción lw (load word) en un procesador canalizado y no canalizado depende de la cantidad de etapas que la instrucción necesita para completarse.

En el caso de un procesador canalizado, la instrucción lw puede atravesar todas las etapas del pipeline: IF, ID, EX, MEM y WB. Según la tabla de latencias dada en la pregunta, las latencias de estas etapas son 250, 350, 150, 300 y 200 ciclos de reloj, respectivamente. Por lo tanto, la latencia total de una instrucción lw en un procesador canalizado sería la suma de estas latencias:

```
250 + 350 + 150 + 300 + 200 = 1250 ciclos de reloj.
```

En contraste, en un procesador no canalizado, la instrucción lw solo necesita pasar por una única etapa para completarse. Según la tabla de latencias, la latencia de la etapa MEM (Memory Access) es de 300 ciclos de reloj, por lo que la latencia total de una instrucción lw en un procesador no canalizado sería de 300 ciclos de reloj.

En resumen, la latencia total de una instrucción lw en un procesador canalizado sería de 1250 ciclos de reloj, mientras que en un procesador no canalizado sería de 300 ciclos de reloj.

La instrucción lw (load word) es una operación de lectura de memoria que carga un valor de una palabra de memoria en un registro del procesador. En la arquitectura MIPS, la instrucción lw tiene dos operandos: el primer operando es el registro destino donde se almacenará el valor de la palabra leída de memoria, y el segundo operando es la dirección de memoria donde se encuentra la palabra.

Para leer el valor de una palabra de memoria, primero debemos acceder a la memoria para obtener la palabra, lo que implica la etapa MEM (Memory Access) en el pipeline. En la etapa MEM, se realiza la lectura de la palabra de memoria y se coloca en el registro intermedio llamado Memory Data Register (MDR). Luego, en la etapa WB (Write Back), el valor del MDR se escribe en el registro destino.

Por lo tanto, la instrucción lw pasa por la etapa MEM en el pipeline porque necesita acceder a la memoria para leer la palabra, lo que se realiza en esta etapa.

Si podemos dividir una etapa de la ruta de datos canalizada en dos nuevas etapas, cada una con la mitad de la latencia de la etapa original, ¿qué etapa dividiría y cuál es el nuevo

tiempo de ciclo de reloj del procesador?

Se divide la instrucción ID al ser la mas ser la mas larga (350 ps) pasando a ser 175 ps.

Por lo que cuando el nuevo tiempo al ser canalizado seria de 300 ps la nueva instrucción mas larga MEM.

Suponiendo que no haya paradas o peligros, ¿cuál es la utilización de la memoria de datos?

Para calcular la utilización de la memoria de datos, necesitamos saber cuántas instrucciones de carga y almacenamiento hay en el flujo de instrucciones. Dado que el 20% de las instrucciones son de carga y el 15% son de almacenamiento, el total es del 35%.

Luego, necesitamos determinar la cantidad de ciclos de reloj que la etapa MEM necesita para completar cada instrucción de carga o almacenamiento. Dado que la latencia de MEM es de 150 ciclos de reloj, se necesita un ciclo para completar una instrucción de carga o almacenamiento.

En consecuencia, la memoria de datos se utiliza el 35% del tiempo en cada ciclo de reloj.

Por lo tanto, la utilización de la memoria de datos es del 35%.

Suponiendo que no haya paradas ni peligros, ¿cuál es la utilización del puerto de registro de escritura de la unidad "Registros"?

Para calcular la utilización del puerto de registro de escritura, necesitamos saber cuántas instrucciones utilizan la etapa WB (escritura en registro) en el flujo de instrucciones. Dado que el 45% de las instrucciones son de ALU, que utilizan la etapa WB, y que no hay ninguna instrucción de salto o de carga/almacenamiento en esta etapa, el 45% de las instrucciones utilizan el puerto de registro de escritura.

Luego, necesitamos determinar la cantidad de ciclos de reloj que la etapa WB necesita para completar cada instrucción que la utiliza. Dado que la latencia de WB es de 200 ciclos de reloj, se necesita un ciclo para completar una instrucción que utiliza esta etapa.

En consecuencia, el puerto de registro de escritura se utiliza el 45% del tiempo en cada ciclo de reloj.

Por lo tanto, la utilización del puerto de registro de escritura es del 45%.

Ejercicio 9.

Este ejercicio explora cómo el manejo de excepciones afecta el diseño de pipelines. Los tres primeros apartados de este ejercicio se refieren a las dos instrucciones siguientes:

beqz x11, LABEL
lw x11, 0(x12)

¿Qué excepciones puede desencadenar cada una de estas instrucciones? Para cada una de estas excepciones, especifique la etapa de canalización en la que se detecta.

La instrucción "beqz x11, LABEL" puede desencadenar una excepción de "excepción de salto" si la condición en x11 no se cumple y el procesador intenta saltar a una dirección de etiqueta no válida. Esta excepción se detecta en la etapa de ejecución.

La instrucción "lw x11, 0(x12)" puede desencadenar varias excepciones, como:

- Excepción de alineación de memoria: si la dirección de memoria no es un múltiplo del tamaño de la palabra, lo que hace que la operación de carga sea imposible. Esta excepción se detecta en la etapa de decodificación.
- Excepción de acceso a memoria: si la dirección de memoria a la que se intenta acceder no está permitida. Esta excepción se detecta en la etapa de acceso a memoria.
- Excepción de fallo de página: si la dirección de memoria a la que se intenta acceder no está mapeada en la tabla de páginas. Esta excepción se detecta en la etapa de traducción de direcciones.

Si hay una dirección de controlador independiente para cada excepción, muestre cómo se debe cambiar la organización del pipeline para poder controlar esta excepción. Puede suponer que las direcciones de estos controladores se conocen cuando se diseña el procesador.

Si se tiene una dirección de controlador independiente para cada excepción, se puede diseñar el pipeline para que tenga una unidad de control de excepciones separada para cada tipo de excepción que pueda ocurrir. Cada unidad de control de excepciones se encargará de manejar una excepción en particular y tendrá su propia dirección de controlador.

En general, el diseño del pipeline se puede organizar de la siguiente manera:

1. Etapa de búsqueda de instrucciones: esta etapa se encarga de buscar la siguiente instrucción a ejecutar y determinar la dirección de la siguiente instrucción.
2. Etapa de decodificación: esta etapa se encarga de decodificar la instrucción, determinar la operación que se va a realizar y los operandos que se necesitan para realizarla.
3. Etapa de ejecución: esta etapa se encarga de ejecutar la operación y producir el resultado.
4. Etapa de acceso a memoria: esta etapa se encarga de acceder a la memoria para obtener o almacenar datos.
5. Etapa de escritura de resultados: esta etapa se encarga de escribir el resultado de la operación en el registro adecuado.
6. Unidad de control de excepciones: esta unidad de control de excepciones se encarga de manejar las excepciones que puedan ocurrir en el pipeline. Cada tipo de excepción tiene su propia unidad de control de excepciones, con su propia dirección de controlador.

Si la segunda instrucción se obtiene inmediatamente después de la primera, describa lo que sucede en el pipeline cuando la primera instrucción causa la primera excepción que enumeró en el Ejercicio 4.9.1.
Muestre el diagrama de ejecución de canalización desde el momento en que se obtiene la

primera instrucción hasta el momento en que se completa la primera instrucción del controlador de excepciones.

Si la primera instrucción ("beqz x11, LABEL") causa una excepción de salto, el pipeline deberá ser interrumpido y se deberá manejar la excepción antes de que se pueda continuar con la ejecución de la segunda instrucción ("lw x11, 0(x12)"). En este caso, el pipeline deberá ser redirigido a la dirección de controlador adecuada para manejar la excepción.

El siguiente diagrama muestra el flujo de ejecución del pipeline desde el momento en que se obtiene la primera instrucción hasta el momento en que se completa la primera instrucción del controlador de excepciones:

```
Etapa de búsqueda de instrucciones:      beqz x11, LABEL
Etapa de decodificación:                 beqz x11, LABEL
Etapa de ejecución:                      (excepción de salto detectada)
Unidad de control de excepciones:        Manejar excepción de salto
                                          - Guardar estado actual del procesador
                                          - Saltar a la dirección de controlador de excepciones
                                          - Realizar la acción necesaria para manejar la excepción
                                          - Restaurar el estado del procesador
                                          - Volver al controlador de flujo de instrucciones
Etapa de búsqueda de instrucciones:      (nada)
Etapa de decodificación:                 (nada)
Etapa de ejecución:                      (nada)
Etapa de acceso a memoria:               (nada)
Etapa de escritura de resultados:        (nada)
```

Ejercicio 8.

La importancia de tener un buen predictor de saltos depende de la frecuencia con la que se ejecutan los saltos condicionales. Junto con la precisión del predictor de bifurcaciones, esto determinará cuánto tiempo se dedica al estancamiento debido a saltos mal predichas. En este ejercicio, supongamos que el desglose de las instrucciones dinámicas en varias categorías de instrucciones es el siguiente:

Categoría de Instrucción	Porcentaje
R-type	40%
beqz/bnez	25%
jal	5%
lw	25%
sw	5%

Además, suponiendo las siguientes precisiones de predictor de bifurcación:

Predictor de bifurcación	Precisión	Estancamiento adicional (IPC)
Siempre tomado	45%	0.55
Siempre no tomado	55%	0.45
2-bit	85%	0.15

Apartado 1) Los ciclos de estancamiento debido a saltos mal pronosticadas aumentan el IPC. ¿Cuál es el IPC adicional debido a saltos mal predichas con el predictor siempre tomado? Supongamos que los resultados de la bifurcación se determinan en la etapa ID y se aplican en la etapa EX que no hay peligros para los datos y que no se utilizan ranuras de retardo.

El cálculo del IPC adicional debido a los saltos mal predichos con el predictor "Always-taken" se puede calcular teniendo en cuenta la frecuencia de los saltos condicionales y la precisión del predictor de bifurcaciones.

La fórmula para calcular el IPC es 1 + (1 - precisión del predictor) * (porcentaje de instrucciones de salto) * penalización.

En este caso, el porcentaje de instrucciones de salto (beqz/bnez) es 25% o 0.25. La precisión del predictor "Always-taken" es 45% o 0.45. La penalización por salto mal predicho en una arquitectura de tubería simple es a menudo de 3 ciclos.

Entonces el cálculo sería:

IPC = 1 + (1 - 0.45) * 0.25 * 3 = 1.4125

Esto significa que con un predictor "Always-taken", el IPC se incrementa a 1.4125 debido a los saltos mal predichos. Es importante notar que este valor representa un incremento en el número de ciclos requeridos para ejecutar las instrucciones, lo que en realidad significa una disminución en el rendimiento.

Apartado 2) Repita el apartado anterior para el predictor "siempre no tomado".

Para el predictor "Always-not-taken", la precisión es del 55% o 0.55. Usando la misma fórmula que en el cálculo anterior, podemos calcular el IPC adicional debido a los saltos mal predichos.

IPC = 1 + (1 - precisión del predictor) * (porcentaje de instrucciones de salto) * penalización

Aquí, el porcentaje de instrucciones de salto (beqz/bnez) es del 25% o 0.25, y la penalización por salto mal predicho en una arquitectura de tubería simple es a menudo de 3 ciclos.

Por lo tanto, el cálculo sería:

IPC = 1 + (1 - 0.55) * 0.25 * 3 = 1.3375

Esto significa que con un predictor "Always-not-taken", el IPC se incrementa a 1.3375 debido a los saltos mal predichos. Como mencioné antes, este valor representa un incremento en el número de ciclos requeridos para ejecutar las instrucciones, lo que en realidad significa una disminución en el rendimiento.

Apartado 3) Repita el apartado 1 para el predictor de 2 bits.

Para el predictor de 2 bits, la precisión es del 85% o 0.85. Usamos la misma fórmula que en los cálculos anteriores para calcular el IPC adicional debido a los saltos mal predichos.

IPC = 1 + (1 - precisión del predictor) * (porcentaje de instrucciones de salto) * penalización

Aquí, el porcentaje de instrucciones de salto (beqz/bnez) es del 25% o 0.25, y la penalización por salto mal predicho en una arquitectura de tubería simple es a menudo de 3 ciclos.

Por lo tanto, el cálculo sería:

IPC = 1 + (1 - 0.85) * 0.25 * 3 = 1.1125

Esto significa que con un predictor de 2 bits, el IPC se incrementa a 1.1125 debido a los saltos mal predichos. Como se mencionó anteriormente, este valor representa un incremento en el número de ciclos requeridos para ejecutar las instrucciones, lo que en realidad significa una disminución en el rendimiento. Sin embargo, este es el rendimiento más alto (o el menor impacto en el rendimiento) de los tres predictores dados, gracias a su mayor precisión del 85%.

Apartado 4) Con el predictor de 2 bits, ¿qué aceleración se lograría si pudiéramos convertir la mitad de las instrucciones de rama en alguna instrucción ALU? Suponga que las instrucciones predichas correcta e incorrectamente tienen la misma posibilidad de ser reemplazadas.

Primero, la proporción de instrucciones de salto se reduce a la mitad, por lo que en lugar del 25%, ahora es del 12.5% (0.125).

El IPC después de esta modificación, utilizando un predictor de 2 bits (85% de precisión), se calcula como:

IPC_posterior = 1 + (1 - 0.85) * 0.125 * 3 = 1.05625

Aceleración = IPC_anterior / IPC_posterior = 1.1125 / 1.05625 ≈ 1.053

Lo siento por el error en el cálculo anterior. El resultado correcto de la aceleración sería aproximadamente 1.053, lo que significa que el rendimiento mejora en un 5.3% aproximadamente.

Apartado 5) Con el predictor de 2 bits, ¿qué aceleración se lograría si pudiéramos convertir la mitad de las instrucciones de rama de una manera que reemplazara cada instrucción de rama con dos instrucciones ALU?
Suponga que las instrucciones predichas correcta e incorrectamente tienen la misma posibilidad de ser reemplazadas.

Tratando las instrucciones ALU como instrucciones separadas:

Aquí, se trata a las dos instrucciones ALU resultantes como instrucciones separadas. Como resultado, el programa modificado ahora tiene 1.125n instrucciones (donde n es el número original de instrucciones), ya que la mitad del 25% de las instrucciones de rama produce una instrucción adicional. Del total de 1.125n instrucciones, .125n de estas (o 11.1%) son instrucciones de rama. Así, se calcula un IPC posterior de 1.01665 teniendo en cuenta el incremento del 12.5% en las instrucciones.

Por lo tanto, la aceleración sería:

Aceleración = IPC_anterior / (1.125 * IPC_posterior) = 1.0375 / (1.125 * 1.01665) = .91

Apartado 6) Algunas instrucciones de bifurcación son mucho más predecibles que otras. Si sabemos que el 80% de todas las instrucciones de bifurcación ejecutadas son saltos de bucle invertido fáciles de predecir que siempre se predicen correctamente, ¿cuál es la precisión del

predictor de 2 bits en el 20% restante de las instrucciones de bifurcación?

Primero, calculemos la tasa de predicción general considerando que el 80% de las instrucciones de bifurcación son predecibles y siempre se predicen correctamente, y que el predictor de 2 bits tiene una precisión del 85% en general.

Entonces, la precisión total de la predicción de bifurcaciones sería:

Precisión_total = 0.80 * 1.00 + 0.20 * Precisión_2_bits

Dado que sabemos que la Precisión_total es 0.85 (85%), podemos resolver la ecuación para la Precisión_2_bits, que representa la precisión del predictor de 2 bits en el 20% restante de las instrucciones de bifurcación.

0.85 = 0.80 * 1.00 + 0.20 * Precisión_2_bits
0.85 = 0.80 + 0.20 * Precisión_2_bits
0.05 = 0.20 * Precisión_2_bits
Precisión_2_bits = 0.05 / 0.20
Precisión_2_bits = 0.25 o 25%

Por lo tanto, para que la precisión total sea del 85% con las condiciones dadas, la precisión del predictor de 2 bits en el 20% restante de las instrucciones de bifurcación tendría que ser del 25%.