Practica 4 Arquitectura de Computadoras

Oscar Albornoz v24584769

## a. ¿En el ámbito de la informática que es para usted la memoria?

En informática, la **memoria** es el dispositivo que retiene, memoriza o almacena datos informáticos durante algún período de tiempo. La memoria proporciona una de las principales funciones de la computación moderna: el almacenamiento de información y conocimiento. Es uno de los componentes fundamentales de la computadora, que interconectada a la unidad central de procesamiento y los dispositivos de entrada/salida , implementan lo fundamental del modelo de computadora de la arquitectura de von Neumann, que permitió desarrollar más que simple calculadoras.

## b. ¿Qué importancia tiene la aparición de la memoria en la arquitectura de computadoras?

Es muy importante, puesto que permitió implementar el concepto de programa almacenado, facilitando la lectura de un conjunto de instrucciones, así como el almacenamiento de los resultados. La memoria es el área de trabajo para la mayor parte del software de un computador. Existe una memoria intermedia entre el procesador y la RAM, llamada caché, pero ésta sólo es una copia (de acceso rápido) de la memoria principal almacenada en los módulos de RAM.

## c. ¿Qué es para usted un pipelining? Nombres 3 ejemplos prácticos (no necesariamente relativos a la informática).

En la informática, una tubería es un conjunto de elementos de procesamiento de datos conectados en serie, donde la salida de un elemento es la entrada de la siguiente. Los elementos de un pipeline se ejecutan a menudo en paralelo o en tiempo cortado; en ese caso, una cierta cantidad de almacenaje del almacenador intermediario se inserta a menudo entre los elementos.

Las tuberías relacionadas con la informática incluyen:

Tuberías de instrucciones, como la tubería RISC clásica, que se utilizan en unidades centrales de procesamiento (CPUs) para permitir la ejecución superpuesta de varias instrucciones con el mismo circuito. El circuito se divide generalmente en etapas y cada etapa procesa una

instrucción a la vez. Ejemplos de etapas son decodificación de instrucciones, aritmética / lógica y búsqueda de registros.

Las pipelines de gráficos, que se encuentran en la mayoría de las unidades de procesamiento gráfico (GPU), que consisten en múltiples unidades aritméticas, o CPU completas, que implementan las diversas etapas de las operaciones de rendering comunes (proyección en perspectiva, recorte de ventana, cálculo de color y luz, renderizado, etc.).

Canales de software, donde se pueden escribir comandos donde la salida de una operación se alimenta automáticamente a la siguiente operación siguiente. El sistema Unix es un ejemplo clásico de este concepto, aunque otros sistemas operativos también soportan tuberías.

HTTP pipelining, donde se envían varias solicitudes sin esperar el resultado de la primera solicitud.

La producción industrial en masa, es también otro ejemplo de pipeling, en donde los productos son construidos por partes, y dichas partes son trabajadas de forma simultánea, como en la industria automotriz, el ensamble de equipos electrónicos y demás.

**d. ¿Cuáles son las etapas de un pipelining?**

Fetch. Hasta cuatro instrucciones son leídas desde la caché de instrucciones.

Decode. Después de la etapa fetch, las instrucciones son predecodificadas y entonces insertadas en el buffer de instrucciones.

Group. La tarea principal aquí es ensamblar un grupo de hasta cuatro instrucciones y todas ellas son despachadas en un solo ciclo. De este grupo de cuatro, dos pueden ser de enteros y dos pueden ser de punto flotante o gráficas. El archivo de registro de enteros es accesado en esta etapa.

Execute. Las dos ALU's de enteros procesan datos desde el archivo de registro de enteros. Los resultados son computados y, vía bypassing, se ponen a disposición de otras instrucciones dependientes en el siguiente ciclo. En esta etapa también se calcula las direcciones virtuales para las operaciones de memoria, en paralelo con la computación de la ALU. En cuanto a punto flotante y gráficas, el archivo de registro de punto flotante es accesado durante esta etapa.

Cache. La dirección virtual de las operaciones de memoria calculadas en la etapa de execute es enviada al tag de la RAM para determinar si el acceso (load o store) es un acierto o desacierto en el caché de datos. En forma paralela, la dirección virtual es también enviada a la unidad de administración de memoria de datos (DMMU) para ser traducidas en un

dirección física. Las operaciones ALU de la etapa execute generan códigos de condiciones en esta etapa. Estos códigos son enviados a la unidad de prefecth y dispatch (PDU), el cual chequea si un salto condicional en el grupo fue correctamente predecido. En caso de una mala predicción, las instrucciones en el pipeline son limpiadas, y las instrucciones correctas son traídas. Las instrucciones de punto flotante y gráficas comienzan su ejecución durante la etapa X1.

N1. Un caché de dato acertado o errado es determinado en esta etapa. Si un load erra un caché de dato, entra al load buffer. La dirección física de un store es enviada al store buffer en esta etapa.

N2. El pipe de enteros espera el pipe de punto flotante. La mayoría de las instrucciones de punto flotante y gráficas finalizan su ejecución durante esta etapa. Después de N2, los datos pueden ser desviados a otras etapas,

N3. Los bloqueos son resueltos en esta etapa.

Write. En esta etapa todos los resultados son escritos en los archivos de registros de la arquitectura (enteros y punto flotante). Una vez completada esta etapa, las instrucciones son consideradas terminadas.

**e. ¿Qué son los "data hazards" en un pipelining?**

Los riesgos de datos ocurren cuando éstos son modificados. El ignorar riesgos de datos potenciales puede resultar en condiciones de carrera (a veces llamadas riesgos de carrera). Hay tres situaciones en las que puede aparecer un riesgo de datos:

1. **Read after Write** (RAW) o **dependencia verdadera**: Un operando es modificado para ser leído posteriormente. Si la primera instrucción no ha terminado de escribir el operando, la segunda estará utilizando datos incorrectos.
2. **Write after Read** (WAR) o **anti-dependencia**: Leer un operando y escribir en él en poco tiempo. Si la escritura finaliza antes que la lectura, la instrucción de lectura utilizará el nuevo valor y no el antiguo.
3. **Write after Write** (WAW) o **dependencia de salida**: Dos instrucciones que escriben en un mismo operando. La primera en ser emitida puede que finalice en segundo lugar, de modo que el operando final no tenga el valor adecuado.

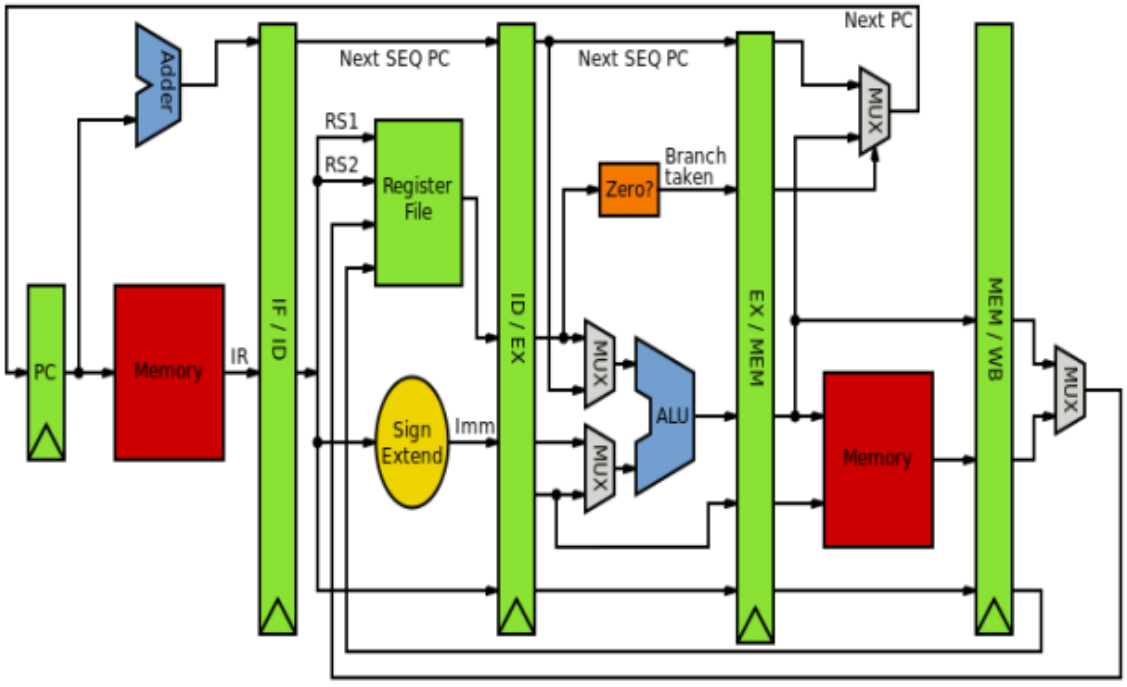
Los operandos envueltos en riesgos de datos pueden residir en memoria o en registros.

**f. ¿Qué métodos se utilizan para tratar los "data hazards"?**

Normalmente se resuelve por transmisión de datos o registro. Esto se basa en el hecho de que los datos seleccionados no se utilizan realmente en ID sino en la etapa siguiente: ALU.

El reenvío funciona de la siguiente manera: el resultado ALU del buffer EX/MEM siempre se retroalimenta a las teclas de entrada ALU. Si el hardware de reenvío detecta que su operando fuente tiene un nuevo valor, la lógica selecciona el resultado más reciente en lugar del valor leído del archivo de registro.

**g. Explique con sus palabras el funcionamiento del pipelining correspondiente al diseño lógico del MIPS que se muestra en la siguiente figura:**




El pipelining del diseño consta de 5 etapas, las cuales por obviedad trabajan simultáneamente sobre el producto que generan sus fases previas. En la primera, llamada Fetch se busca en memoria y trae la instrucción apuntada por el contador de programa. Posteriormente, se decodifica dicha instrucción, desmembrándola en las direcciones de registro que subyacen y la acción que debe realizarse sobre los operados que allí residen. En la siguiente etapa, como su nombre lo indica se

ejecuta la instrucción, evaluando previamente el valor de los operandos se utiliza la ALU para ejecutar operaciones de carácter aritmético lógicas. El penúltimo paso es el acceso a memoria, alojando allí el resultado de la operación si así explícitamente lo indicase la instrucción, y a su vez, se envía la señal de alerta para el contador de programa apunte a la siguiente instrucción. Finalmente, en la ultima fase del ciclo, que recibe el nombre de Write Back, se guarda el resultado en la dirección de registro que haya sido señalada.

## h. ¿Qué es un Flip-flop?

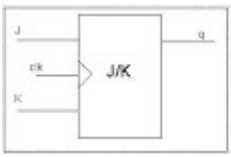
Un flip flop es un circuito electrónico, llamdo también simplemente biestable, que tiene dos estados estables. El flip flop es un elemento básico de memoria que es capaz de almacenar un número binario (bit), es decir, que permanece indefinidamente en uno de sus dos estados posibles aunque haya desaparecido la señal de excitación que provocó su transición al estado actual.

Debido a su amplia utilización, los flip flops se han convertido en un elemento fundamental dentro de los circuitos secuenciales

## i. Tipos de flip-flops y diferencia entre ellos.

### FLIP FLOP JK

El flip flop JK se emplea para eliminar la incertidumbre cuando las señales de entrada J=K=1, en el caso del flip flop RS, esta asignación de los valores de las entradas estaban prohibidas en el caso del RS,sin embargo aquí en el JK, se dará la función basculamiento,




| clk | J | K | q |
|---|---|---|---|
| ↑ | L | L | q |
| ↑ | L | H | L |
| ↑ | H | L | H |
| ↑ | H | H | notq |

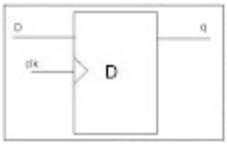
El flip flop JK es el más completo de ,los flip flops que se emplean. Tiene dos entradas J y K, similares a las entradas S y R de un flip flop RS. La estrada J realiza la función set y la entrada K la función reset . La principal diferencia entre ambos es que J y K pueden valer uno simultáneamente, a diferencia del flip flop RS, en este caso la salida

cambia de estado, pasando a valer lo contrario de lo que valía antes. Es el basculamiento

En este ejemplo, se ha implementado un flip flop JK, basándonos en la operatividad de un flip flop d, previamente descrito en la librería rtlpkg.Esto se hace através de la utilización de las señales qx y dx,

**FLIP FLOP D**

Este circuito tiene como entradas las señales de reloj (clk) y de información (d), siendo activa por flanco de subida. Como salida se tiene al puerto q.



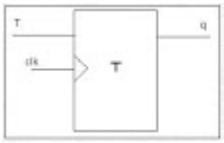

| clk | D | q |
|---|---|---|
| ↑ | L | L |
| ↑ | H | H |

Esta entidad es muy sencilla ya que las entradas y la salida son de un único bit de ancho.

El circuito opera de la siguiente forma: cuando hay un flanco de subida en el puerto de entrada clk, y la entrada d vale '1', entonces la salida q pasa a tomar el valor de d. Cuando clk está a nivel bajo, la entrada d se encuentra desabilitada, manteniendo q el valor anterior. Esta es la base de su operatividad como memoria.

**FLIP FLOP T**

El flip flop T, dispone de una única entrada de control t. Activado por flancos de subida, es empleado como un contador de flancos.La salida q, mantendra su valor anterior hasta la llegada del siguente flanco.




| clk | T | q |
|---|---|---|
| ↑ | L | q |
| ↑ | H | not q |

El flip flop T basa toda su operatividad en el valor lógico de su única entrada de control. Si en t tenemos el nivel lógico uno, la salida q basculara, cambiando constantemente su valor según le bayan llegando

los flancos de subida del reloj. Si la entrada t está a nivel lógico cero, el flip flop mantiene su valor anterior a modo de una memoria.

## j. ¿Qué son direcciones de memoria?

En informática, una dirección de memoria es un dispositivo receptor para una localización de memoria con la cual un programa informático o un dispositivo de hardware deben almacenar un dato para su posterior reutilización.

Una forma común de describir la memoria principal de un ordenador es como una colección de celdas que almacenan datos e instrucciones. Cada celda está identificada unívocamente por un número o dirección de memoria.

Para poder acceder a una ubicación específica de la memoria, la CPU genera señales en el bus de dirección, que habitualmente tiene un tamaño de 32 bits en la mayoría de máquinas actuales. Un bus de dirección de 32 bits permite especificar a la CPU {\displaystyle 2^{32}} 2^{{32}} = 4.294.967.296 direcciones de memoria distintas.

Debido a la estructura de 32 bits de un microprocesador común como los de Intel, las direcciones de memoria se expresan a menudo en hexadecimal. Por ejemplo, para no tener que escribir 11111101010000000000010101100 podemos escribir 3F5000AC en hexadecimal.

## k. ¿Qué es un registro y cómo se compone?

En arquitectura de ordenadores, un **registro** es una memoria de alta velocidad y poca capacidad, integrada en el microprocesador, que permite guardar transitoriamente y acceder a valores muy usados, generalmente en operaciones matemáticas.

Los registros están en la cumbre de la jerarquía de memoria, y son la manera más rápida que tiene el sistema de almacenar datos. Los registros se miden generalmente por el número de bits que almacenan; por ejemplo, un "registro de 8 bits" o un "registro de 32 bits". Los registros generalmente se implementan en un banco de registros, pero antiguamente se usaban biestables individuales, memoria SRAM o formas aún más primitivas.

El término es usado generalmente para referirse al grupo de registros que pueden ser directamente indexados como operandos de una

instrucción, como está definido en el conjunto de instrucciones. Sin embargo, los microprocesadores tienen además muchos otros registros que se usan con un propósito específico, como el contador de programa . Por ejemplo, en la arquitectura IA32, el conjunto de instrucciones define 8 registros de 32 bits. Los **registros de datos** se usan para guardar números enteros. En algunas computadoras antiguas, existía un único registro donde se guardaba toda la información, llamado *acumulador*.

- Los **registros de memoria** se usan para guardar exclusivamente direcciones de memoria. Eran muy usados en la arquitectura Harvard, ya que muchas veces las direcciones tenían un tamaño de palabra distinto que los datos.
- Los **registros de propósito general** (en inglés GPRs o *General Purpose Registers*) pueden guardar tanto datos como direcciones. Son fundamentales en la arquitectura de von Neumann. La mayor parte de las computadoras modernas usa GPR.
- Los **registros de coma flotante** se usan para guardar datos en formato de coma flotante.
- Los **registros constantes** tienen valores creados por hardware de sólo lectura. Por ejemplo, en MIPS el registro cero siempre vale 0.
- Los **registros de propósito específico** guardan información específica del estado del sistema, como el puntero de pila o el registro de estado.

También existen los registros bandera y de base.

**l. ¿Qué es un archivo de registro y cómo se realiza la lectura/ escritura sobre él?**

El archivo de registro es una forma conveniente de describir un conjunto de registros. Tratamos el archivo de registro como una caja negra. En particular, el archivo de registro tiene las siguientes entradas:

**SRC 1 Addr** Este es el índice del primer registro fuente. También se llama **$rs** en MIPS, y a veces se llama el registro Read 1. Contiene 5 bits, ya que se necesitan 5 bits para especificar uno de los 32 registros.

**SRC 2 Addr** Este es el índice del segundo registro fuente. También se llama **$rt** en MIPS, y a veces llamado el registro Read 2. Contiene 5 bits.

**DST Addr** Este es el índice del registro de destino. También se llama **$rd** en MIPS (para instrucciones de tipo R---para I-type, $rt, es el registro de destino), y a veces se llama el registro Write. Contiene 5 bits.

**DST Data** Se trata de los datos que se escribirán en el registro de destino, especificados por DST Addr. Son 32 bits.

**WE** Los datos en DST Data se escriben en el registro en el índice **DST Addr** sólo cuando **WE = 1**. Si **WE = 0**, no se actualizan los registros en el archivo de registro.

También tiene los siguientes resultados

**SRC 1 Data** Este es el contenido del primer registro fuente, como se especifica en **SRC 1 Addr**.

**SRC 2 Data** Este es el contenido del primer registro fuente, según lo especificado por **SRC 2 Addr**.

| Opcode | Register s | Registrar t | Registrar d | Shif | Function |
|---|---|---|---|---|---|
| 000 000 | 000 00010 | 00011 | 00001 | 00000 | 100 000 |

Lectura de un archivo de registro

Imagina que queremos ejecutar la siguiente instrucción:

```
sumar $r1, $r2, $r3 # R[1] <- R[2] + R[3] + R[3
```

¿Cuáles son los pasos individuales necesarios para que esto suceda? Para realizar la suma, se necesitan los valores de los registros 2 y 3.

Necesitamos recuperar los operandos, es decir, los valores del registro 2 y 3, del fichero de registro. Entonces, ¿qué necesitamos decirle al archivo de registro? Tenemos que decirle qué registros queremos.

El registro s es el primer registro fuente. El registro t es el segundo registro fuente. El registro d es el registro de destino. Recuerde que en la instrucción agregue $r1, $r2, $r3, $r1 es el registro de destino, mientras que $r2 y $r3 son los dos registros fuente.

Esta es una de las razones por las que un buen diseño ISA ayuda. Como conocemos la disposición de una instrucción tipo R, podemos obtener los bits deseados para los registros fuente sin decodificar mucho.

Una vez que los diez bits se envían desde el IR como entradas al archivo de registro, el archivo de registro produce 64 bits de salida. Especificamos 5 bits para el índice del primer registro fuente, y éste produce los 32 bits que son el contenido de ese registro. También

especificamos 5 bits para el índice del primer registro fuente, y éste produce los 32 bits que son el contenido de ese registro.

Hay un pequeño retardo desde que las direcciones de entrada (indexes) son leídas por el archivo de registro, hasta que las salidas (contenido del registro) son producidas por el archivo de registro

**Escribir en el Archivo de registros**

Leer de un archivo de registro es sólo la mitad de la historia. También necesitamos modificar los registros en el archivo de registro. Aquí es donde podemos tomar nuestra señal de la memoria.

Con la memoria, necesitamos la capacidad de leer de memoria y escribir en memoria. Para distinguir entre qué operación queremos hacer, tenemos el pin de control, R/\W. Cuando R/\W = 1, decimos a la memoria que queremos leer de memoria. Cuando R/\W = 0, decimos a la memoria que queremos escribir en memoria.

Resulta que sólo leeremos desde el archivo de registro como se muestra en la sección anterior. Tendremos entradas adicionales al archivo para escribir en el archivo de registro.

Por lo tanto, usaremos una entrada de control llamada WE para activar escritura. Cuando WE = 1, entonces queremos escribir en el archivo de registro. De lo contrario, no escribimos al archivo de registro. Por supuesto, necesitaremos especificar a qué registro escribir (no queremos escribir en todos los registros).

Para ver cómo debemos configurar el archivo de registro para la escritura, considere la instrucción otra vez.

```
sumar $r1, $r2, $r3 # R[1] <- R[2] + R[3] + R[3
```

Una vez que añadimos R[2] a R[3], necesitamos guardar el resultado en un registro, es decir, R[1]. Este es el registro de destino.

Cada instrucción puede, como máximo, actualizar un registro --el registro de destino.

Necesitamos la capacidad de especificar el índice del registro de destino en el archivo de registro.

Al igual que los registros de origen, requerimos 5 bits para indicar el registro de destino. Además, al igual que el registro de destino, estos 5 bits pueden venir de IR. En particular, los bits B15-11 son los bits utilizados para el registro de destino en una instrucción tipo R. Estos bits son entradas para DSTaddr.

También necesitamos 32 bits de datos para escribir en el registro. Así que esto es DSTdata. Los datos proceden de la salida de la ALU.

De laboratorio

**a. ¿Qué representan los tipos de datos sc_uint, sc_int y sc_vector de la biblioteca systemC?**

**sc_uint** es un entero sin signo, y es un número entero fijo de precisión de tamaño 64 bits. Las operaciones subyacentes utilizan 64 bits, pero el tamaño del resultado se determina en la declaración de objeto. Los operadores del tipo sc_uint se pueden convertir a tipo sc_int y viceversa utilizando sentencias de asignación. Cuando se asigna un entero a un operando sin signo, el valor entero en la forma de complemento de 2 se interpreta como un número sin signo. Cuando se asigna un unsigned a un operando firmado, el unsigned se expande a un número de 64 bits sin firmar y luego se truncan para obtener el valor firmado.

**sc_int** es entero firmado, y es un número entero de precisión fijo de 64 bits. Las operaciones subyacentes utilizan 64 bits, pero el tamaño del resultado se determina en la declaración de objeto. En sc_int los valores se almacenan en el halago de 2. Puesto que es entero firmado, el bit de signo para un sc_int de N bit ancho se almacena en la posición de N-1 bit.

**sc_vector** permite crear vectores de objetos SystemC, que normalmente requieren un parámetro de nombre en su constructor.

Proporciona un acceso de tipo array a los miembros del vector y gestiona los recursos asignados automáticamente. Una vez que el tamaño es asignao y se inicializan los elementos, no se admiten operaciones de cambio de tamaño adicionales. Las firmas de constructores personalizados son soportado por una función de plantilla init, que soporta asignaciones de elementos definidas por el usuario.

**b. ¿Qué tipo de datos manejan los read() y write() de las variables de systemC?**

Primitivos e implementados por la librería.

**c. ¿Qué tipo de representación permite .read().to_string (SC_BIN)?**

Devuelve una cadena de caracteres del valor leído en binario.