Proyecto de Arquitectura de Computadoras

Curso 2021

Microporcesador S-MIPS

Darío Fragas González, C213

Introducción:

En este proyecto se implementó en Logisim un procesador que implementa la arquitectura de juegos de instrucciones S-MIPS. Arquitectura de 32 bits donde el procesador tiene 32 registros de 32 bits, de propósito general, en el que $r0$ tendrá siempre el valor constante 0 y r31 será el *stack pointer* (**SP**).

Visión General

El procesador se implementó siguiendo el patrón de un procesador multiciclos, donde se aprovecha una única Unidad Aritmética Lógica (**ALU**) en vez de tener un sumador extra para incrementar el puntero de instrucciones (**PC**). Se implementa mediante la conexión de 3 componentes principales: Control Unit (**CU**), Datapath y Memoria Caché, los cuales se componen a su vez de varios subcomponents.

A modo general tendremós una memoria Caché que facilitará la interacción con la RAM, pues aprovechará que en cada lectura de la RAM puede leer 4 valores de esta. Esta memoria recibe entradas de datos y de dirección del Datapath y a su vez envía a este datos leidos de la memoria. Además recibe entradas de control del Control Unit que establece cuando se debe leer o escribir.

El Datapath manejará los registros -agrupados en el Register File-, la **ALU** y multiples registros que propician la implementación multiciclo del procesador. En general el Datapath recibe una única entrada de datos -que pueden ser datos o instrucciones- junto a multiples banderas y señales de control que vienen del Control Unit, y tiene una salida de datos, una de direcciones y los respectivos códigos de operación o función utilizados por el Control Unit para determinar a qué estados transicionar.

El Control Unit será el componente que maneje toda la lógica del microprocesador, se encuentra en un primer estado IDLE de donde pasa a un estado FETCH donde se lee la proxima instrucción de la RAM (o la Caché). Luego pasa un estado en que se decodifica la instrucción para luego pasar a un estado donde se ejecuta dicha instrucción, de aquí en adelante surgen otros estados en dependencia del tipo de instrucción a ejecutar y al final todos regresan al estado IDLE (con exepcion de la instrucción halt que detiene la ejecución).

Datapath

Instruction Register:

Cuando se lee una instrucción esta se almacena en un registro llamado Instruction Register (**IR**), dicho registro se implemento en un componente cuya función es dividir la instrucción de 32 bits, así se toman los 6 primeros bits como el **Op-Code**, los próximos 5 como **RS**, los siguientes 5 como **RT** y los proximos 5

como **RD**, los últimos 6 como **F-Code**, los bit de la posición 0-15 como offset, además de sacar la instrucción entera puesto que los bits 0-25 son usados en la operación jump (**j**).

Register File:

Estará formado por 32 registros de 32 bits, el primer registro siempre contiene el valor cero y cuaquier intento de escritura en él va a tener efecto nulo, a su vez el ultimo registro será el puntero a la pila (**SP**), el cual puede ser modificado aunque se recomienda que no se haga y que su valor sea manejado exclusivamente por las operaciones pop y push. Dicho registro se inicializa apuntado a la última posición de la memoria. Las entradas al Register File serán los selecotres A y B que determinan el par de registros a leer, un selector que indica el registro que va a ser modificado en caso de escritura, una entrada de datos a escribir y una bandera que determine si se realiza proceso de escritura o de lectura.

En el Datapath las entradas de los selectores A y B siempre van a ser **RS** y **RT** respectivamente, la bandera de lectura/escritura viene del **CU**, el selector del registro a escribir puede tomar 3 valores:

- **RD** -operaciones de tipo R mayormente-
- **RT** -operaciones de tipo I mayormente-
- El valor 31 usado para escribir en el **SP** en las operaciones de push y pop.

Los datos a escribir pueden tomar 4 valores:

- La salida de datos de la **ALU**
- La entrada de datos de la memoria
- Valor proveniente de los registros **HI** y **LO**
- Valor aleatorio proveniente del generador de números aleatorios.

Las salidas del RegFile serán los registros A y B seleccionados para leer y una salida especial para el **SP**. Las salidas A y B están conectadas a dos registros en el que se almacenan los valores leidos en la fase de decodificación de la instrucción.

ALU:

Este componente recibe dos operandos y un codigo que identifica el tipo de operación a realizar, teniendo como salidas el resultado de dicha operación y banderas de control.

La entrada del primer operando podrá tomar 3 posibles valores:

- El valor almacenado en **PC**
- El registro A sacado del RegFile
- El valor almacenado en **SP**

En la entrada del segundo operando podrán entrar 4 valores:

- El valor B sacado del RegFile
- La constante 4 usada para incrementar o substraer dirección
- El valor del offset de la instrucción extendido a 32 bits
- El valor del offset de la instrucción extendido a 32 bits y con un shift izquierdo de 2 (Equivalente a multiplicar por 4)

Así la salida principal de la ALU se conecta a un registro (**ALUOut**), en caso de operaciones de multiplicación o división el resultado se depositará en los registros **HI** y **LO**. Varias banderas que determinan si el resultado de la operación fue cero y otras que indican si el primer operando es mayor, menor o igual que cero. Estas banderas se conectan al Branch Control para determinar si una instrucción de salto debe ejecutarse.

PC:

El program counter será un registro con una bandera que contrale su escritura y un selector que determina que valor va a escribirse en el. Así los posibles valores a escribir en **PC** son:

- La salida de la **ALU**
- El registro **ALUOut** (no confundir con el mencionado anterior)
- El valor tomado de una instrucción j

- El registro A en caso de una instrucción jr

La escritura en este registro se controla mediante la bandera **IRWrite** proveniente del **CU** o en caso de instrucciónes de Branch si se cumple el criterio.

La salida de datos puede ser el registro A o el registro B, el A solo sale en las operaciones de tty y de push. La de dirección puede ser **PC** en caso de necesitar leer una instrucción o **ALUOut** cuando se requieren datos, además puede tomar el valor de **SP** para las instrucciones push y pop.

Control Unit

El Control Unit maneja la lógica principal del procesador, en un primer momento está en un estado de IDLE del cual pasa inmediatamenta el estado de FETCH en el cual tomará una instrucción de la memoria. Aquí setea la bandera **IorD** a 0 para indicar que se va a tomar la dirección de memoria a la que apunta **PC**, además de emitir un control **MemRead** que le indica a la memoria que se quiere realizar una lectura. Permanece en este estado hasta que se haya leido la instrucción. Una vez leida la instrucción antes de pasar al siguiente estado aprovecha que la **ALU** no está siendo usada para calcular el valor $PC + 4$ de la próxima instrucción, en caso de ser una operación de salto este valor será sobreescrito, además se acierta la bandera **IRWrite** para escribir la instrucción leida.

En el estado Decode se leeran los respectivos registros dados por la instrucción, dado que en este momento no se encuentra en uso la **ALU** se aprovecha para calcular el valor $PC + 4 + offset * 4$ utilizado en las operaciones de tipo Branch, y dado que ya tenemos el valor de $PC + 4$ solo tenemos que setar la entrada A de la **ALU** a **PC** y la entrada B al offset con 2 shift izquierdo.

Aquí se analiza la instrucción a ejecutar para determinar que acciones realizar.

En el caso de instrucciones lw y sw se usa la **ALU** para sumar los valores leidos de A con el offset luego en caso de ser sw se manda a escribir el registro B en la dirección calculada, en caso de ser lw se lee la dirección calculado y luego se

escribe en el registro **RT**. Luego se regresa a IDLE, (nótese en este punto que ya el valor de **PC** apunta a la siguiente instrucción).

Para las instrucciónes de tipo R lógica-aritemeticas que consisten en operar dos valores y guardar los resultados se lleva a cabo en dos ciclos de reloj, el primero para operarlos en la **ALU** y otro para escribirlos en el RegFile, de aquí se pasa al estado **IDLE**.

En caso de instrucciones de tipo Branch, se ponen los registros A y B como entradas de la **ALU** poniendo la operación a realizar como resta, esto hará que el Branch Control reciba las respectivas banderas con la operación de Branch y tenga como salida 1 si ha de hacerse un branch, en tal caso se habilita la escritura en **PC** tomándose la nueva dirección aquella calculada en la etapa de decode ($PC + 4 + offset * 4$) que se encuentra actualemente en **ALUOut**

Para las operaciones aritméticas y lógicas de tipo I se seleccionan las entradas de la **ALU** de forma que en A tendremos el registro que será el primer operando y en el segundo el valor inmediato tomado del offset extendido. Aquí se cambiará el método de extensión, tomandose extensión con cero para el caso de operaciones lógicas. Luego se escribe al registro correspondiente.

Las instrucciones de tipo Jump solo habilitan la escritura en PC y tomara el valor resultante de combinar los 4 primeros bit de PC con los 26 últimos bits de la instrucción con un shift izquierdo de dos, o el valor del registro A en dependencia si la instrucción es j o jr respectivamente.

Las instrucciónes mfhi y mflo primero selecciona el registro a leer y luego escribe al registro deseado aplicando las banderas y selectores pertinentes.

Para la operación push se selecciona como entadas de la ALU el registro **SP** y la constante 4 con el control para efectuar una resta. El resultado se pone en la salida de dirección del Datapath y en en la salida de datos de este se ponen los selectores para que salga el valor almacenado en el registro A. Se acierta la bandera para escribir en memoria y una vez escrito se actualiza **SP**. Para la operación pop primero se saca la dirección de **SP** y se aciertan banderas de lectura de memoria. Una vez leido se escribe al registro pertinente el nuevo valor, por ultimo se actualiza el valor de **SP**.

La instrucción tty activa la bandera TTY ENABLE además de activar el selector de salida al valor del registro A.

La instrucción rand activa como entrada de el RegFile la salida del generador de números aleatorios y la entrada **RD** como dirección a escribir.

La instrucción kbd activa la señal KBD ENABLED además de la señal KBD INPUT que cambia la entrada al Datapath a la entrada KBD DATA.

Por último la instrucción halt setea la salida STOP para detener la ejecución y se queda en un estado con un loop infinito hacia sí mismo.

Caché

La memoria Caché está formada por 8 líneas de 32 bytes. Lo que le da una capacidad de 256 bytes. Se toman 3 bits de la dirección para el indíce, determinando en qué linea se escribe la palabra. Se toman 2 bits de offset que indican en qué banco se escribe la palabra y 13 bits para el tag. El tipo de mapeo es Direct Mapping. La política de escritura es wirte back. El proceso más lento que realiza la Caché es cuando se pide escribir datos en una línea que contiene datos válidos pero que no tienen el mismo tag y estos datos están marcados como dirty, en este caso se escriben los datos en Caché a la memoria principal, luego se trae a la caché el bloque de la memoria principal a donde corresponde la escritura y se escribe en la caché, marcando los datos como dirty. Si los datos en una línea de caché son válidos y el tag de dicha línea coincide con el tag del dato a escribir la escritura se hace en un ciclo de reloj. Igual la lectura se hace en un solo ciclo del reloj si el dato buscado hace hit. En caso de que los datos en caché sean válidos y no coincida el tag con el del dato buscado, si los datos en caché están marcados dirty se escriben primero a la memoria principal, luego en cualquiera de los casos se traen los datos del bloque correspondiente a la caché y se lee o escribe (marcando como dirty) el dato requerido.

Observaciones

1- Todas las componentes que presentan memoria tienen entradas de reloj y de reset que son las mismas para todo el microprocesador.

2- Para cualquier instrucción se llevan a cabo al menos 4 ciclos del reloj.

3- La operación que más demora (toamando el tiempo de lectura y escritura en memoria igual a un ciclo de reloj) es la operación pop que demora 8 ciclos.

4- En general la mayoría de las operaciones se completan en 5 ciclos del reloj.