Informe de la Práctica Profesional Facultad de Matemática y Computación Procesador S-MIPS

Javier A. Oramas López C-212
Profesor Responsable: Carlos Bermudes Porto

June 28, 2021

1 Introducción

El Objetivo de esta práctica fue diseñar en el logisim un procesador que implemente la arquitectura de juegos de instrucciones S-MIPS. S-MIPS es una arquitectura de 32 bits. Con este trabajo tenia como principal objetivo poner en práctica todo lo aprendido en el curso de Arquitectura de Computadoras (AC) y lograr el buen funcionamiento del S-MIPS, es decir que fuer capaz de ejecutar y realizar todas las operaciones con éxito.

2 Flujo del Procesador

El microprocesador S-MIPS está formado por los siguientes componetes **Program Counter(PC)**, **Branch Control Unit (BCU)**, **Instruction Fetcher (IF)**, **Instrction Decoder (ID)**,**Control Unit (CU)**, **Register File(RF)**, **Arithmetic Logic Unit (ALU)** y **Cache**

Este flujo intenta explicar de la mejor manera posible el funcionamiento del micro, no obstante está explicado de manera secuencial, cuando muchas de estas operaciónes pueden estar ocurriendo en paralelo o pueden no cumplir explicitamente el orden de escritura, todo está en dependencia del programa que se esté ejecutando

1. Al inicio de la ejecución de un programa, la CU envía la señal nextInstr al PC y BCU
2. Como no hay ninguna instrucción anterior, el PC envía la direción inicial al BCU, a la Cache de instrucciónes y al IF
3. El Cache de instrucciónes recibe la dirección de la instrucción que deberá enviar a IF, carga el dato y lo envía a IF

4. El IF recibe la dirección donde se encuentra la instrucción a ejecutar, además recibe 4 outputs de la cache, de donde obtendrá la instrucción a decodificar

5. El ID recibe la instrucción del IF, la procesa y, en dependencia del tipo de instrucción, puede tener distintos comportamientos:

   (a) Instrucción R:
   por las salidas Rs, Rd y Rt, envía las direcciónes en el RF de los registros que serán utilizados por la instrucción a ejecutar

   (b) Instrucción I
   por las salidas Rs y Rt envía los registros a ser utilizados, además por la salida Offset envía el offset

   (c) Instrucción J
   por la salida Dir envía la direccion de memoria a donde debe saltar la instrucción

   además de estas salidas, el ID tiene una serie de flags que representarán cada una de las posibles operaciónes a realizar, solo una de ellas puede estar encendida a la vez

6. La CU recibe todas las flags del ID y en dependencia de la operación a realizar envía el código de operación (solo para instrucciónes de la ALU), flags para garantizar el correcto funcionamiento de los componentes (como puede ser control de lectura o escritura en el RF, la operación a realizar por el BCU, los jumps , activación del KBD o TTY entre otros)

7. EL RF recibe los selectores de registro enviados por el ID, si se va a escribir en él, recibe el dato a escribir de la fuente que sea pertinente (Resultado de la ALU, entrada de KBD, datos en memoria enviados por la cache de datos, etc...), además de las flags correspondientes de la CU, en dependencia de estos realizará una de las siguientes operaciones:

   (a) obtiene los valores solicitados por los selectores recibidos y los devuelve

   (b) escribe el dato recibido en la dirección indicada

   (c) mueve el SP en dependencia de la operación (pop/push)

8. La ALU recibe los datos enviados por el RF como operandos, alternativamente puede operarse con el offset o con un valor constante 0 sustituyendo a uno de estos elementos, recibe tambien, proveniente de la CU una señal que indica si se desea realizar la operación con signo o sin signo, y el código de operación, para seleccionar el resultado que se desea obtener la ALU emite como salida el resultado de la operación seleccionada, así como dos salidas extra llamadas Hi y Lo, para datos extras provenientes de operaciones con multiplicación y división

además cuenta con los flags slt, igualdad ente los operandos, menor que cero e igual a 0, que brindan información extra de la operación, el primero es extendido a 32 bits y puede ser almacenado en el RF, los tres últimos son enviados a la BCU

9. La BCU puede realizar dos tipos de operaciónes, el movimiento a la siguiente instrucción o saltar a una instrucción específica, recibe además los flags de las condicionales para seleccionar la siguiente instrucción el resultado de estas operaciones es enviado al PC para iniciar el próximo ciclo de operaciones

10. Una vez terminada la ejecución de una instrucción, la CU da nuevamente la señal de obtener una instrucción nueva, hasta que la instrucción a ejecutar se halt, la cual termina el proceso de ejecución activando el flag de STOP en la CU

3 Componentes

Muchos de estos componentes reciben además entradas de reloj y/o reset, se omiten en la explicación, dado que es evidente e irrelevante la explicación de las mismas

3.1 Program Counter(PC)

El Program Counter está formado por un registro de 32 bits encargado de almacenar la dirección en memoria de la instrucción actual

3.1.1 Entradas

1. En: señal que activa el registro para almacenar un nuevo dato

2. PC-In: Nueva dirección a almacenar

3.1.2 Salidas

1. PC-Out: Dirección en memoria de la instrucción actual

3.2 Branch Control Unit (BCU)

La Branch Control Unit es la encargada de mover el valor de PC, en dependencia de la operación esto puede ser:

1. Instrucción Siguiente (PC+4)

2. Saltar a una posición específica (PC = Dest)

3. Mover según el offset si se cumple una condición (PC + Offset), la condición que se debe cumplir está dada por las flags beq, bne, blez, bgtz, bltz, y se calcula su cumplimiento o no a partir de las entradas ZF, ¡0 e = recibidas de la ALU

3.2.1 Entradas

1. beq: añade el Offset a PC si Rs = Rt
2. bne: añade el Offset a PC si Rs $\neq$ Rt
3. blez: añade el Offset a PC si Rs ≤ 0
4. bgez: añade el Offset a PC si Rs ≥ 0
5. bltz: añade el Offset a PC si Rs < 0
6. j: PC = Dest
7. jr: Pc = Rs
8. next: Selecciona si se ha de actualizar el valor de salida o no
9. ZF: Rs == 0
10. =: Rs = Rt
11. < 0: Rs < 0
12. PC-In: dirección de la instrucción actual

3.2.2 Salidas

1. PC-Out: Dirección de la nueva instrucción

3.3 Instruction Fetcher (IF)

EL Instruction Fetcher recibe la dirección en memoria de la próxima instrucción a ser ejecutada, dicha dirección previamente ha sido precargada por la cache de instrucciones y es enviada a través de una de las 4 entradas de memoria, el IF selecciona el banco correspondiente y envía la instrucción almacenada en este al registro donde almacena la instrucción actual, esta misma es devuelta al usuario

3.3.1 Entradas

1. Ready: Señal que indica que ya el Cache de instrucciones tiene cargada la dirección que se desea acceder
2. Bank0-3: 4 entradas correspondientes a las salidas de los 4 bancos de memoria (proveniente del Cache de instrucciones)

3.3.2 Salidas

1. Inst: Instrucción a ser decodificada y posteriormente ejecutada
2. Done: flag que señala q terminó el proceso de obtener la instrucción

3.4 Instrction Decoder (ID)

El Instruction Decoder recibe la instrucción y extrae los bits 26-31 para determinar el tipo de instrucción:

1. Instrucción R: de los bits 0-5 obtinene la operación y activa el flag correspondiente divide el resto de la instrucción para obtener las direcciones de los registros Rt, Rd y Rs
2. Instrucción I: los bits 26-31 definen la operación y activa el flag correspondiente, del resto de la instrucción obtiene las direcciones de Rt y Rs, así como el valor de Offset
3. Instrucción J: los bits 26-31 definen la operación y activa el flag correspondiente, el resto de la instrucción será tomado como la dirección de la operación jump

3.4.1 Entradas

1. Instruction: Instrucción a decodificar

3.4.2 Salidas

1. 36 flags: correspondientes a cada operación realizable por el micro
2. Rs,Rt,Rd: salidas de las direcciónes en el RF de los registros a utilizar
3. Offset: salida del valor de Offset de las instrucciones tipo I
4. Dir: Salida de la dirección para las instrucciónes de tipo J

3.5 Control Unit

La Control Unit recibe la operación a realizar, si esta es una operacion aritmético/lógica enviá el codigo de esa operacion a la ALU, además actia otras flags en dependencia de la operación que se vaya a realizar

3.5.1 Entradas

1. 36 flags provenientes del ID: de todos los flags provenientes del ID obtiene la información de l aoperación que se va a realizar

3.5.2 Salidas

1. Next: solicitar próxima instrucción
2. OpCode: código de operación de la ALU
3. Sign: Define si la operación en la ALU se ha de realizar con signo o sin él
4. beq, bne, blez, bgtz bltz: señales de operación en el BCU
5. j, jr: saltos en el BCU
6. enRND: Activa el random generator
7. instEn: Señal de activación para la Cache de instrucciones
8. STOP: señal de halt, temina la ejecución
9. enRead,enWrite: señal de lectura y escritura para la cache de datos
10. pushRf, popRF: operaciones de pila en el RF
11. enSP: activa o desactiva el registro SP del RF (para modificar su contenido)
12. R/W: intercambia entre lectura y escritura en el RF
13. TTYEn: activa el TTY (para imprimir los datos)
14. KBDEn: activa el KBD (para recibir datos del teclado)

3.6 Register File(RF)

EL Register File consta de 32 registros de 32 bits, enumerados de R_0 a R_{31} donde R_0 siempre va a contener el valor 0 y R_{31}, también llamado SP actuaraá como contenedor del puntero de pila (Stack Pointer) El Register File puede realizar 2 operaciones:

1. Read: Recibe dos direccioes de 5 bits, las cuales se harán corrseponder con un registro de los que conforma el RF, los valores de ambor registros se enviarán por las salidas del RF
2. Write: Recibe una dirección de 5 bits y un valor de 32 bits a almacenar el en registro correspondiente a la dirección recibida

3.6.1 Entradas

1. dirA, dirB: entradas de 5 bits para indicar los registros que se desean
2. dirData: dirección del registro donde se desea guardar la informaci'on recibida
3. Data: información que se desea guardar en el RF
4. pop, push: operaciónes para manipular el SP
5. EnSP: permite guardar o no las modificaciones en el SP

3.6.2 Salidas

1. S1, S2: valor de los registros seleccionados
2. SP: Stack Pointer

3.7 Arithmetic Logic Unit(ALU)

La Arithmetic Logic Unit es la encargada de realizar las operaciones aritmético lógicas, recibe la operación a realizar de la CU, junto con los operandos ya sea del RF, Offset o el valor constante 0 si se activa la flag Sign, se ha de remover el bit de signo de los operandos, de lo contrario directamente estos se pueden usar, se selecciona la operación deseada y se manda a la salida el resultado, alternativamente pudiera enviarse tambien información por las salidas Hi y Lo si se tratara de una multiplicación o división

3.7.1 Entradas

1. A,B: Operandos
2. Sign: indicador de si se debe tomar operaciones con signo o sin signo
3. OpCode: selector para determinar que operación se ha de realizar

3.7.2 Salidas

1. Res: resutado de la operación
2. Hi,Lo: puede devolver, dependiendo de si la operación realizada es multiplicación o division el resultado de la multiplicacion por ambas salidas o el resultado de la division por Hi y el resto de la división por Lo
3. flags ZF, < 0, $=$, Sign: flags que describen las propiedades correspondientes de Res

3.8 Cache

El cache implementado fue diseñado con una función de mapeo Direct Mapped, politica de reemplazo Random, y politica de escritura write trough y fue dividido en dos componentes:

3.8.1 Cache Instr

La Cache de instrucciones recibe la dirección de memoria que se desea acceder, se divide en offset tag, e idx se verifica si la cache contiene el dato que se desea obtener, de ser así, directamente se devuelve el dato, de lo contrario se envía a la RAM la solicitud de la dirección deseada, se activa un contador hasta el Read time recibido por la Cache y una vez termine se guarda el valor en el banco de la cache correspondiente y se enviía el dato como parte de la salida, además se activa el flag de terminado

3.8.2 Cache Data

La cache de Datos Funciona de manera parecida a la cache de instrucciones, recibe el address a buscar, si está escribiendo envía los datos a la ram para que se escriban y una vez termine este proceso, almacena los datos en el bloque que le corresponde, si existe un bloque disponible lo almacena allí, de lo contrario, selecciona una dirección random y alli lo almacena, para la lectura el procedimiento es análogo, al procedimiento de la cache de instrucciones, solo que tomando el bloque a modificar en la cache de manera random

3.8.3 Entradas

1. Banks0-3: bancos de ram
2. Addr: dirección a leer o escribir
3. Data (solo en Cache de datos): Valor a guardar en la Cache
4. RT, WT: tiempos de lectura y escritura de la Ram

3.8.4 Salidas

1. CacheBank0-3: 4 bancos del cache
2. memAddr: dirección que se desea obtener de la ram
3. DoneRead/Write: flags que señalan el fin de la lectura o escritura respectivamente
4. CS: Chip Select

4 Conclusiones

El desarrollo de este trabajo permitió alcanzar un mayor dominio en el funcionamiento de los microprocesadores S-MIPS, así como e desarrollo del Cache, llevando lo aprendido en la teoría a un nivel práctico, lidiando además con todas las complicaciones que conlleva este proceso

5 biliografía

1. Digital Design and Computer Architecture
2. Clases 2-4, 6-8, 10 del Curso Aquitectura de Computadoras 2021-2022, Matcom, UH