26 de Octubre de 2020

Tecnológico de Monterrey

Unidad de Control y ALU control

Laboratorio de arquitectura de Computadoras

ITESM Campus Monterrey

Profesor
Diego Fernando Valencia Martínez

Miguel Morales de la Vega
Javier Mondragón Martín del Campo

A00821541
A01365137

Introducción

Durante esta práctica se implementara el “datapad” de nuestro MIPS-32, por lo consiguiente se calcularán las entradas de los multiplexores y componentes distintos para ejecutar cada instrucción dependiendo de la entrada de el “operation instruction” o instrucción de operación, esto también definirá cómo se interpreta la cadena de bits restante que se encuentra después de los que definen la instrucción de operación, siendo de tipo R, I o J definidas a continuación.

- Tipo R: utilizado por las instrucciones aritmética, este formato contiene diferentes secciones, la primera de 6 bits correspondiente al OpCode este siempre esta en 0, después siguen 3 secciones de 5 bits cada una correspondiente al registro 1, 2 y el de destino, la última sección corresponde a Funct el cual nos indica qué operación aritmética se ejecutará.
- Tipo I: usado en las instrucciones de transferencia, saltos por condicional y operaciones con inmediatos. Se conforma por 4 secciones, la primera corresponde a 6 bits de Opcode la cual nos indica la instrucción a ejecutar, las siguientes dos secciones de 6 bits corresponden al registro base y destino, La última sección de 16 bits corresponde al inmediato o desplazamiento según la instrucción especificada.
- Tipo J: utilizado en instrucciones de bifurcación como lo es el “jump”. Se conforman de 2 secciones, la primera es el OpCode que nos indica la instrucción a realizar y la segunda sección de 26 bits hace referencia a la dirección destino.

Para desarrollar el comportamiento de cada instrucción y que de distintas maneras se tendrá que realizar dos módulos, el primero siendo el “Control Unit” o unidad de control, que definirá el comportamiento de los multiplexores exceptuando el de la entrada Jr y entradas de los componentes diferentes a la unidad de control aritmetica. Siendo el segundo un “ALU Control” que le indicara que tipo de operación realizará la unidad de control aritmética dependiendo de lo que reciba de la unidad de control, haciendo uso de el multiplexor que tendrá la entrada Jr si es que la instrucción lo solicita.

Procedimiento

Control Unit

La unidad de control configura las entradas de multiplexores dependiendo del tipo de instrucción como comentado anteriormente, para definir las operaciones de instrucciones, este proyecto se basó en el manual de operaciones de un procesador oficial MIPS, donde describen las mismas en su manual de su página oficial, sin embargo, no se implementaron todas las instrucciones, solo se agregaron ADD, SUB, AND, OR, SLT, JR, LW, SW, BEQ, J, ADDI, ORI y LUI. Esta unidad le comunica por una entrada de bits llamados ALUop con el ALU Control, esto con el motivo de comunicarle el tipo de instrucción que se está ejecutando y la operación que requiere que realice la unidad de operación aritmética en caso que no sea tipo R la instrucción. Estas definiciones se pueden visualizar en la Tabla 2.1.1.

ALU Operations				
Instrucción	Tipo	OpCode	Operación	ALUop
SPECIAL	R	000000	SPECIAL	000
LW	I	100011	ADD	100
SW	I	101011	ADD	100
BEQ	I	000100	SUB	101
J	J	000010	NOP	010
ADDI	I	001000	ADD	100
ORI	I	001101	OR	110
LUI	I	001111	BUPPER	111

Tabla 2.1.1
Definiciones de las instrucciones de operación

Como se nota en la tabla anterior, hacen falta 5 instrucciones. Esto es porque en un procesador oficial de MIPS tiene entre sus definiciones un tipo de operación llamado “SPECIAL” el cual requiere más bits para definir la instrucción a utilizar, a este tipo de instrucción nos referimos a la tipo R (Tabla 2.1.2).

SPECIAL
Add
Sub
And
Or
Slt
Jr

Tabla 2.1.2
Operaciones SPECIAL

Para este desarrollo se realizó una tabla de verdad la cual nos orientará en el planteamiento de las entradas y salidas en los distintos casos para las operaciones de cada una de las instrucciones de la unidad de control. Considerando una instrucción no definida como OTHERS y tomándola como un NOP (No operación).

Control Unit										
Instrucción	Tipo	OpCode	RegDst	Jump	Branch	MemRead	MemToReg	MemWrite	ALUSrc	RegWrite
SPECIAL	R	000000	1	0	0	0	0	0	0	1
LW	I	100011	0	0	0	1	1	0	1	1
SW	I	101011	X	0	0	0	0	1	1	0
BEQ	I	000100	X	0	1	0	X	0	0	0
J	J	000010	X	1	0	0	X	0	X	0
ADDI	I	001000	0	0	0	0	0	0	1	1
ORI	I	001101	0	0	0	0	0	0	1	1
LUI	I	001111	0	0	0	0	0	0	1	1
OTHERS	X	XXXXXX	X	0	0	0	X	0	X	0

Tabla 2.1.3
Tabla de verdad de las entradas de los componentes sobre distintas instrucciones

Una vez realizadas las tablas se llevó a cabo la implementación en VHDL de este módulo replicando la tabla 2.1.3 de todas las salidas del Control Unit exceptuando la del ALUop. Para utilizar el case select se utilizó una señal que contuviera los 8 bits y cada bit de la señal estará conectada a cada señal saliendo del componente. El ALUop llevará lo indicado definido en la tabla 2.1.1 con case select. (Código 2.1.4)

```
architecture Behavioral of ControlUnit is

signal outBus : STD_LOGIC_VECTOR (7 downto 0);

begin

      with opCode select
            outBus <= "10000001" when "000000", -- SPECIAL
                              "00011011" when "100011", -- LW
                              "00000110" when "101011", -- SW
                              "00100000" when "000100", -- BEQ
                              "01000000" when "000010", -- J
                              "00000011" when "001000", -- ADDI
                              "00000011" when "001101", -- ORI
                              "00000011" when "001111", -- LUI
                              "00000000" when others;
```

```
      RegDst      <= outBus(0);
      Jump        <= outBus(1);
      Branch      <= outBus(2);
      MemRead     <= outBus(3);
      MemToReg <= outBus(4);
      MemWrite <= outBus(5);
      ALUSrc      <= outBus(6);
      RegWrite <= outBus(7);


      with opCode select
            ALUOp <= "000" when "000000", -- SPECIAL
                           "100" when "100011", -- LW
                           "100" when "101011", -- SW
                           "101" when "000100", -- BEQ
                           "010" when "000010", -- J
                           "100" when "001000", -- ADDI
                           "110" when "001101", -- ORI
                           "111" when "001111", -- LUI
                           "010" when others;

end Behavioral;
```

Code 2.1.4
Codigo del Control Unit en VHDL

ALU Control

Como se comentó anteriormente, este módulo da uso de la salida a la cadena de bits que definen el comportamiento de la unidad aritmética, a esta se le llama ALU Ctrl y de igual manera a la salida Jr que va conectado al multiplexor con esa entrada.
El módulo se realizó tomando en cuenta las tablas definidas para acordar las mismas definiciones de las entradas que este iba a recibir de el control de unidad por medio de la entrada ALUop, se realizó una tabla de verdad para definir qué operación se iba a realizar en la unidad de control operando con estas operaciones. Se considera otra instrucción no definida como others y resultaría en NOP. De igual manera se incluyó la salida Jr en esta tabla y solo será habilitada en el caso de la instrucción Jr. El resultado es mostrado en la tabla 2.2.1.

ALU Control						
Instrucción	Tipo	OpCode	ALUCtrl	ALU Operación	Instruction Code	Jr
Add	R	000	010	ADD	100000	0
Sub	R	000	110	SUB	100010	0
And	R	000	000	AND	100100	0
Or	R	000	001	OR	100101	0
Slt	R	000	111	SLT	101010	0
Jr	R	000	101	NOP	001000	1
J	J	010	101	NOP	XXXXX	0
Lw	I	100	010	ADD	XXXXX	0
Sw	I	100	010	ADD	XXXXX	0
Beq	I	101	110	SUB	XXXXX	0
Addi	I	100	010	ADD	XXXXX	0
Ori	I	110	001	OR	XXXXX	0
Lui	I	111	100	B UPPER	XXXXX	0
Others	X	XXX	XXX	X	XXXXX	0

Tabla 2.2.1
Tabla de verdad de el ALU Control

Una vez terminada la tabla se utilizó el mismo razonamiento que la unidad de control para el desarrollo del código de VHDL y se realizó con la misma metodología (Codigo 2.2.2).

```
architecture Behavioral of ALUControl is
signal Special : STD_LOGIC_VECTOR (2 downto 0);
begin

      with Instruc select
            Special <=    "000" when "100100", --AND
                                  "001" when "100101", --OR
                                  "010" when "100000", --ADD
                                  "110" when "100010", --SUB
                                  "111" when "101010", -- SLT
                                  "101" when others;

      with ALUOp select
            ALUCtrl <=  "010" when "100", --ADD / LW,SW,ADDI
                                 "110" when "101", --SUB / BEQ
```

```
                                    "001" when "110", --OR /ORI
                                    "100" when "111", --B UPPER / LUI
                                    "101" when "010", --NOP
                                    Special when "000", --TIPO R
                                    "000" when others; --NOTHING

      Jr <= '1' when ALUOp = "001" and Instruc = "001000" else '0';

end Behavioral;
```

Código 2.2.2
Código del ALU Control en VHDL

Resultados

Control Unit

Para visualizar que las salidas estén operando correctamente se generó un testbench que mostrará el comportamiento de las salidas de este módulo dependiendo de la instrucción dada (Código 3.1.1).

```
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
            opCode <= "000000"; -- SPECIAL
            wait for 100 ns;
            opCode <= "100011"; -- LW
            wait for 100 ns;
            opCode <= "101011"; -- SW
            wait for 100 ns;
            opCode <= "000100"; -- BEQ
            wait for 100 ns;
            opCode <= "000010"; -- J
            wait for 100 ns;
            opCode <= "001000"; -- ADDI
            wait for 100 ns;
            opCode <= "001101"; -- ORI
            wait for 100 ns;
            opCode <= "001111"; -- LUI
            wait for 100 ns;
            opCode <= "111111"; -- X
            wait for 100 ns;
      -- insert stimulus here
```

```
      wait;
   end process;
```

Código 3.1.1
Codigo del Control Unit Testbench en VHDL

Los resultados de este testbench son mostrados en la imagen 3.1.2.

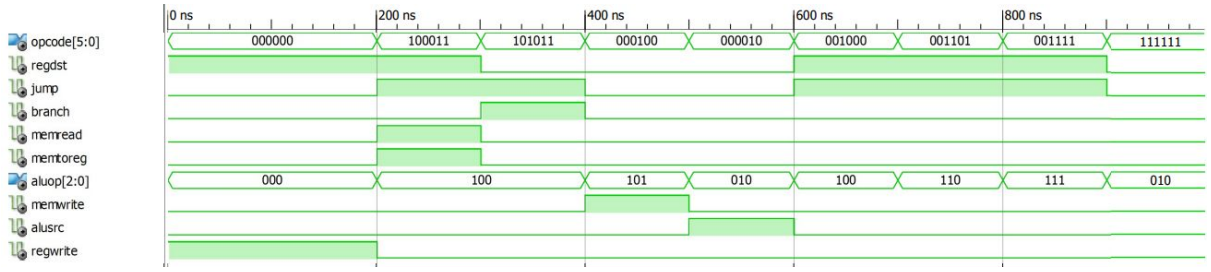

Imagen 3.2.2
Resultados del Control Unit Testbench en VHDL

ALU Control

De igual manera, para visualizar que las salidas estén operando correctamente se generó otro testbench que mostrará el comportamiento de las salidas de este módulo dependiendo de la entrada dada por la unidad de control y si lo requiere, la porción de instrucción extra (Código 3.2.1).

```
ENTITY ALUControl_tb IS
END ALUControl_tb;

ARCHITECTURE behavior OF ALUControl_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT ALUControl
    PORT(
         Instruc : IN  std_logic_vector(5 downto 0);
         ALUOp : IN  std_logic_vector(2 downto 0);
         ALUCtrl : OUT  std_logic_vector(2 downto 0);
         Jr : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal Instruc : std_logic_vector(5 downto 0) := (others => '0');
   signal ALUOp : std_logic_vector(2 downto 0) := (others => '0');
```

```
      --Outputs
   signal ALUCtrl : std_logic_vector(2 downto 0);
   signal Jr : std_logic;


BEGIN

      -- Instantiate the Unit Under Test (UUT)
   uut: ALUControl PORT MAP (
          Instruc => Instruc,
          ALUOp => ALUOp,
          ALUCtrl => ALUCtrl,
          Jr => Jr
        );


   -- Stimulus process
   stim_proc: process
   begin

      -- insert stimulus here
            Instruc <= "100100"; --AND
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;
            Instruc <= "100101"; --OR
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;
            Instruc <= "100000"; --ADD
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;
            Instruc <= "100010"; --SUB
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;
            Instruc <= "101010"; -- SLT
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;
            Instruc <= "000000"; --JR
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;

            ALUOp <= "100"; --ADD / LW,SW,ADDI
            wait for 100 ns;
            ALUOp <=  "101"; --SUB / BEQ
            wait for 100 ns;
```

```
            ALUOp <= "110"; --OR /ORI
            wait for 100 ns;
            ALUOp <= "111"; --B UPPER / LUI
            wait for 100 ns;
            ALUOp <= "010"; --NOP
            wait for 100 ns;
            ALUOp <= "000"; --TIPO R
            wait for 100 ns;
            ALUOp  <= "001"; --NOTHING
        wait for 100 ns;
    end process;

END;
```

Imagen 3.2.1
Código del ALU Control Testbench en VHDL

Los resultados de este test bench son mostrados en la imagen 3.2.2.

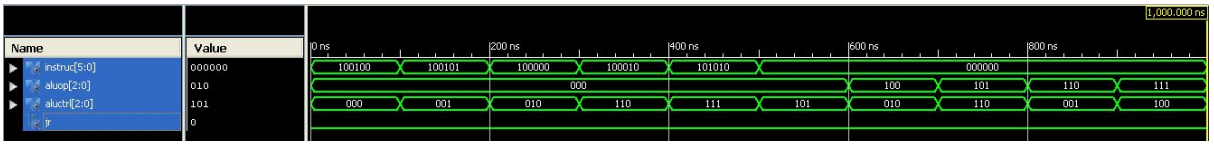

Imagen 3.2.2
Resultados del ALU Control Testbench en VHDL

Conclusión

Durante la práctica se desarrollaron tablas de verdad que ayudaron a facilitar el desarrollo del código, estas tablas tuvieron que ser modificadas varias veces para mostrar el comportamiento oportuno de cada operación, recordando que cuando no se utiliza un componente simplemente se puede dejar una X en la tabla de verdad. Durante el desarrollo de las mismas se aprendieron conceptos y se visualizaron los aprendidos en el curso de una manera más aterrizada a la vida real, viendo como un procesador transforma una instrucción en un resultado dependiendo del tipo de operaciones y componentes que requiera. Podemos observar que los resultados de los testbench no son muy relevantes ya que los módulos aún no están integrados pero de todas formas sirven para saber si VHDL interpretó bien el código. Adicionalmente vimos cómo estas unidades operan los módulos que se realizaron a lo largo de este laboratorio y que están listos para ser juntados para terminar nuestro procesador MIPS32

Reflexion Individual

Miguel Morales:

Durante el desarrollo del laboratorio logre poner en práctica los diferentes conceptos que se ven durante la clase de Arquitectura Computacional, con un mayor énfasis en el tipo de instrucciones de la arquitectura MIPS así como la codificación de estas. En el desarrollo de la práctica pude ver de manera interna como las instrucciones maquinales son ejecutadas por los diferentes componentes de hardware así como analizar la función de cada unos de los componentes, módulos y entidades que hemos estado realizando a lo largo del semestre. Esta práctica fue enriquecedora con alguna dificultad al hacer la relación de la sintaxis de la instrucción con los componentes necesarios para su funcionamiento.

Javier Mondragon:

Pienso que el desarrollo de esta práctica es la que le da forma a los módulos que desarrollamos durante todo el semestre ya que vemos cómo se implementa y cuál será su funcionalidad gracias a estos módulos. Es interesante como todo se interconecta y cómo podemos aplicar estos conceptos no solo a la arquitectura MIPS, si no, que nos podemos dar una idea que así están diseñadas la mayoría de las arquitecturas. También es importante recordar en el diseño de tablas de verdad que lo que no se utilice simplemente se asigna como cualquier valor “X” ya que para el usuario final (hasta para nosotros), nos mostrara en un futuro para próximas expansiones o para la operación normal del MIPS y búsqueda de errores de código insertado en el MIPS saber si todas las entradas entran un efecto en la salida y si son importantes.

Referencias

- https://www.mips.com/products/architectures/mips32-2/
- https://s3-eu-west-1.amazonaws.com/downloads-mips/documents/MD00086-2B-MIPS32BIS-AFP-6.06.pdf
- http://max.cs.kzoo.edu/cs230/Resources/MIPS/MachineXL/InstructionFormats.html