Fecha: 28 de septiembre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

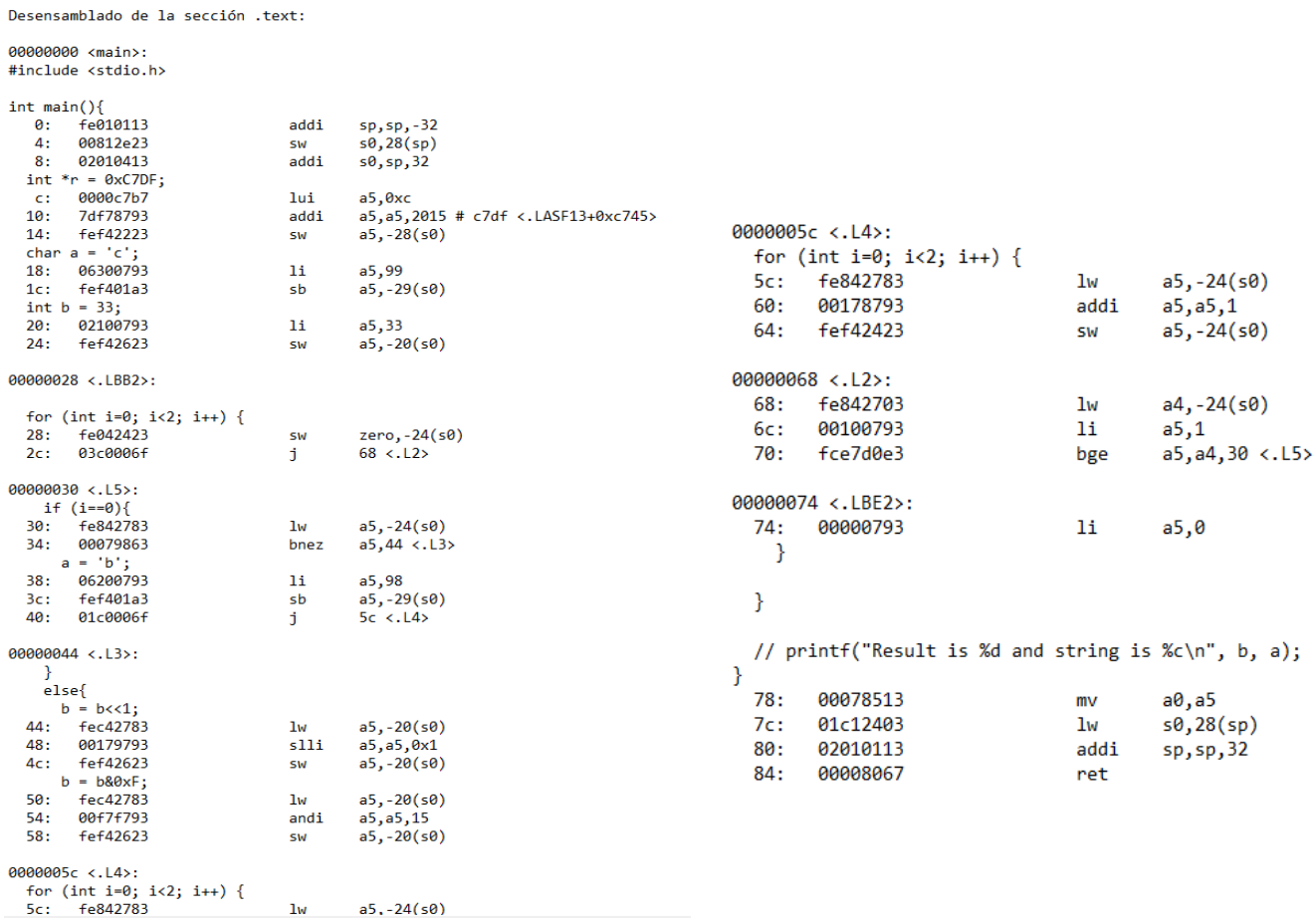
Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Reunión para coordinar primer avance del proyecto.

Acuerdos tomados: Se realizó una reunión por medio de Discord para hacer el primer avance del proyecto.

Se hizo la compilación del código C y se obtuvieron las instrucciones en binario y ensamblador (Ver Figura 1), además se realizó una tabla con la explicación de cada una de las instrucciones obtenidas.



```
Desensamblado de la sección .text:

00000000 <main>:
#include <stdio.h>

int main(){
   0:   fe010113                addi    sp,sp,-32
   4:   00812e23                sw      s0,28(sp)
   8:   02010413                addi    s0,sp,32
  int *r = 0xC7DF;
   c:   0000c7b7                lui     a5,0xc
  10:   7df78793                addi    a5,a5,2015 # c7df <.LASF13+0xc745>
  14:   fef42223                sw      a5,-28(s0)
  char a = 'c';
  18:   06300793                li      a5,99
  1c:   fef401a3                sb      a5,-29(s0)
  int b = 33;
  20:   02100793                li      a5,33
  24:   fef42623                sw      a5,-20(s0)

00000028 <.LBB2>:

  for (int i=0; i<2; i++) {
  28:   fe042423                sw      zero,-24(s0)
  2c:   03c0006f                j       68 <.L2>

00000030 <.L5>:
    if (i==0){
  30:   fe842783                lw      a5,-24(s0)
  34:   00079863                bnez    a5,44 <.L3>
      a = 'b';
  38:   06200793                li      a5,98
  3c:   fef401a3                sb      a5,-29(s0)
  40:   01c0006f                j       5c <.L4>

00000044 <.L3>:
    }
    else{
      b = b<<1;
  44:   fec42783                lw      a5,-20(s0)
  48:   00179793                slli    a5,a5,0x1
  4c:   fef42623                sw      a5,-20(s0)
      b = b&0xF;
  50:   fec42783                lw      a5,-20(s0)
  54:   00f7f793                andi    a5,a5,15
  58:   fef42623                sw      a5,-20(s0)

0000005c <.L4>:
  for (int i=0; i<2; i++) {
  5c:   fe842783                lw      a5,-24(s0)

0000005c <.L4>:
  for (int i=0; i<2; i++) {
  5c:   fe842783                lw      a5,-24(s0)
  60:   00178793                addi    a5,a5,1
  64:   fef42423                sw      a5,-24(s0)

00000068 <.L2>:
  68:   fe842703                lw      a4,-24(s0)
  6c:   00100793                li      a5,1
  70:   fce7d0e3                bge     a5,a4,30 <.L5>

00000074 <.LBE2>:
  74:   00000793                li      a5,0
    }

  }

  // printf("Result is %d and string is %c\n", b, a);
}
  78:   00078513                mv      a0,a5
  7c:   01c12403                lw      s0,28(sp)
  80:   02010113                addi    sp,sp,32
  84:   00008067                ret
```

Figura 1. Objdump generado del código C. Fuente: Obtenido por los estudiantes.

Fecha: 30 de septiembre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Revisión de diagrama para las instrucciones Tipo S y Tipo I.

Acuerdos tomados: Se verifica entre los miembros el diagrama de propuesta para el avance 1 para instrucciones tipo S:

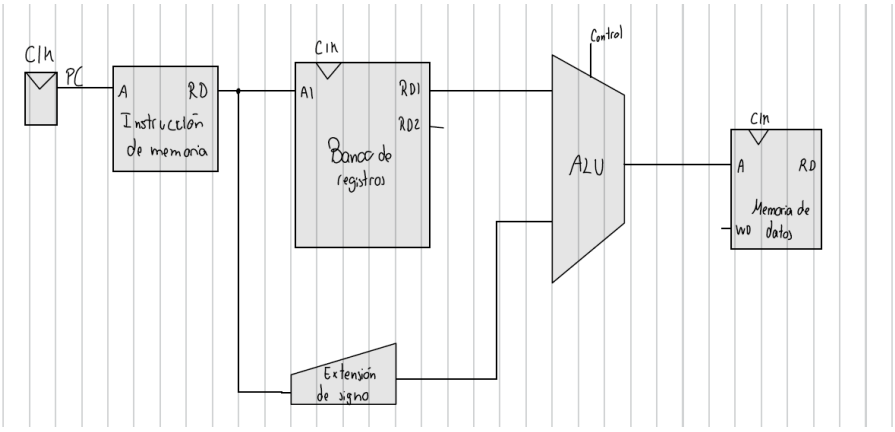


Figura 2. Diagrama para las instrucciones Tipo I. Fuente: Elaborado por las estudiantes.

Se finalize la reunion con este diseño a la espera de revision para el avance 1.

Fecha: 2 de octubre, 2024

Participantes: Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Temas discutidos: Restructuración de diagramas para instrucciones tipo S e I

Acuerdos tomados: Se realizó la corrección de los diagramas basándose en la arquitectura múlticiclo.

Los diagramas realizados se adjuntan a continuación:

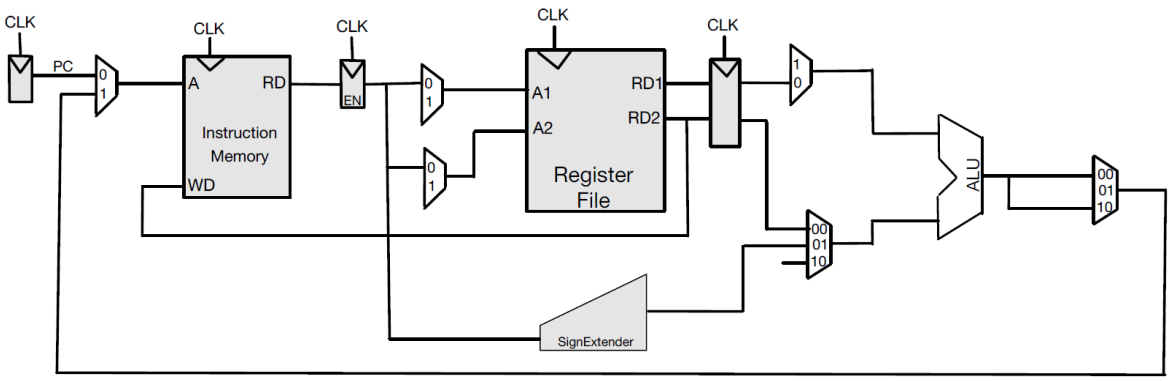


Figura 3. Diagrama para las instrucciones Tipo I. Fuente: Elaborado por las estudiantes.

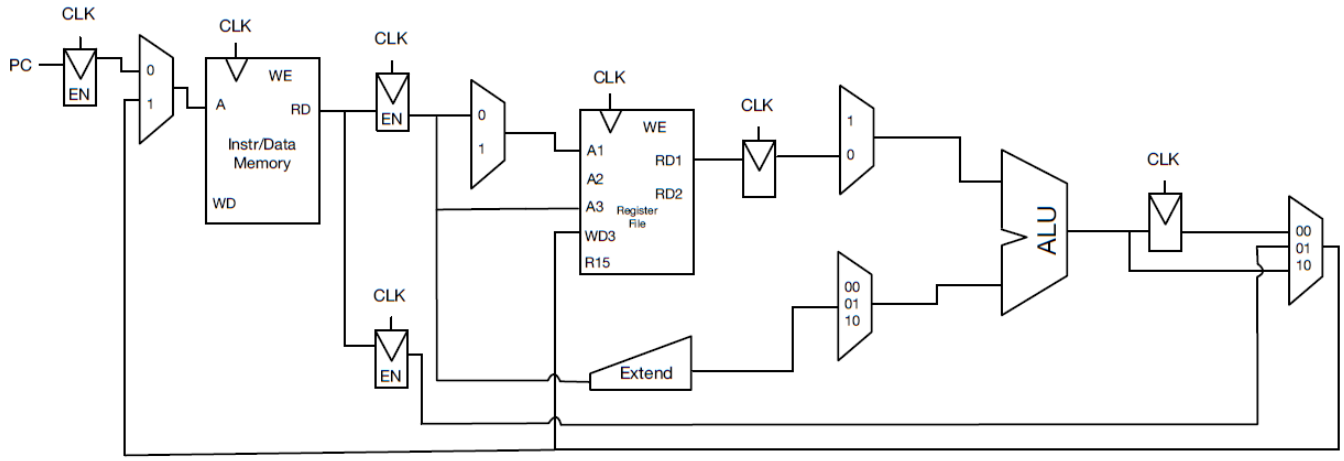


Figura 4. Diagrama para las instrucciones Tipo S. Fuente: Elaborado por las estudiantes.

Fecha: 5 de octubre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Elaboración de los diagramas tipo B y J.

Acuerdos tomados: Se realizó una revisión de los diagramas realizados previamente para corroborar que todo este de la manera correcta, y se inició realizo la elaboración de los diagramas de tipo B y tipo J para multiciclo:

Los diagramas realizados se adjuntan a continuación:

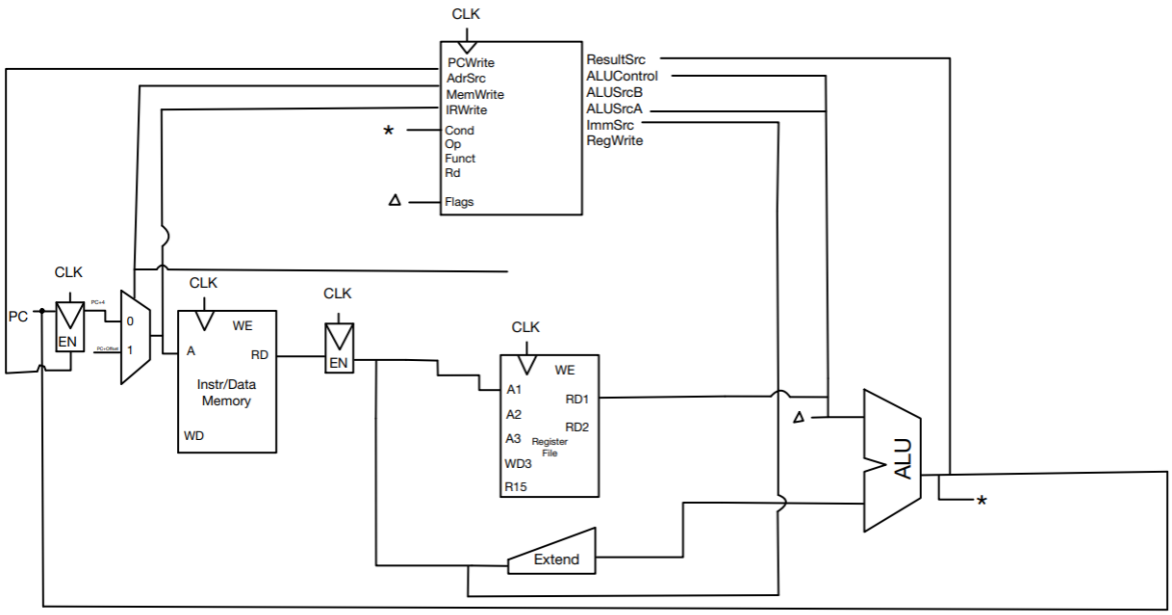


Figura 5. Diagrama para las instrucciones Tipo B. Fuente: Elaborado por las estudiantes.

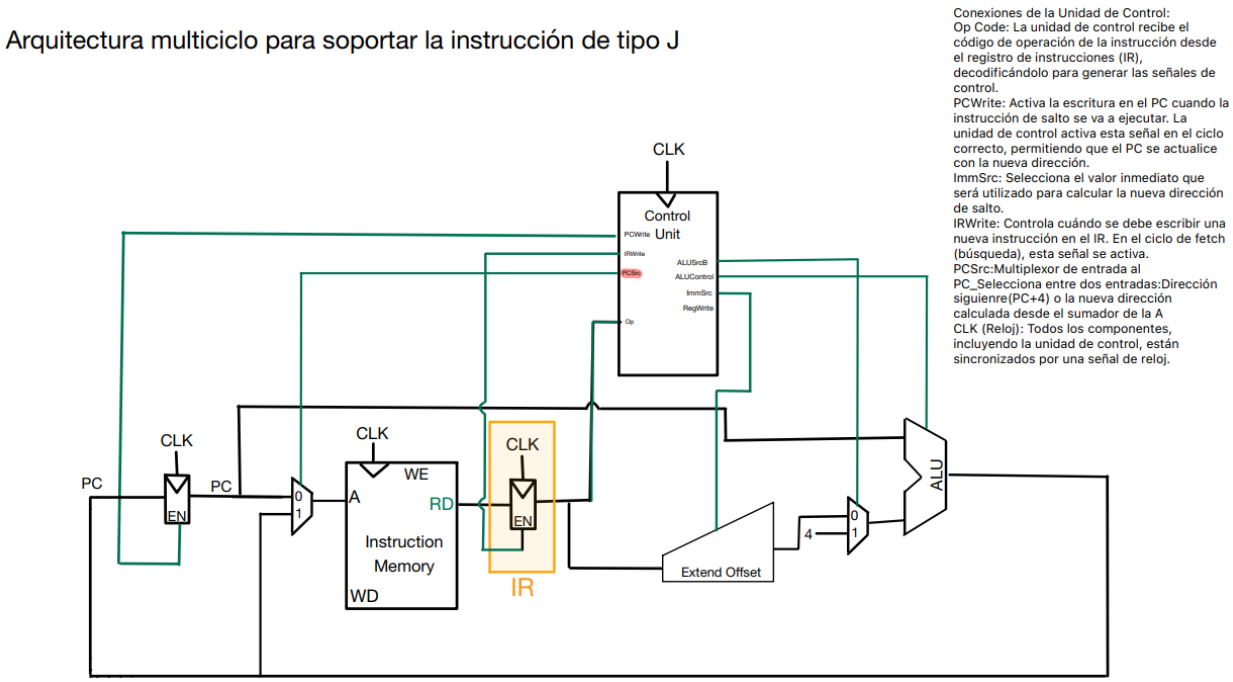


Figura 6. Diagrama para las instrucciones Tipo J. Fuente: Elaborado por las estudiantes.

Fecha: 10 de octubre, 2024

Participantes: Andres David Apuy Garro

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Ana Victoria Rojas Lazo.

Temas discutidos: Revisión de diagramas elaborados y organización de elaboración de máquina de estados para arquitectura multiciclo.

Acuerdos tomados: Se realizó una revisión de los diagramas para cada tipo de instrucciones obtenidas y seguidamente se elaboró el ultimo diagrama que es el de tipo U. Además, se tomó la decisión de iniciar con la máquina de estados para la arquitectura multiciclo.

Los diagramas realizados se adjuntan a continuación:

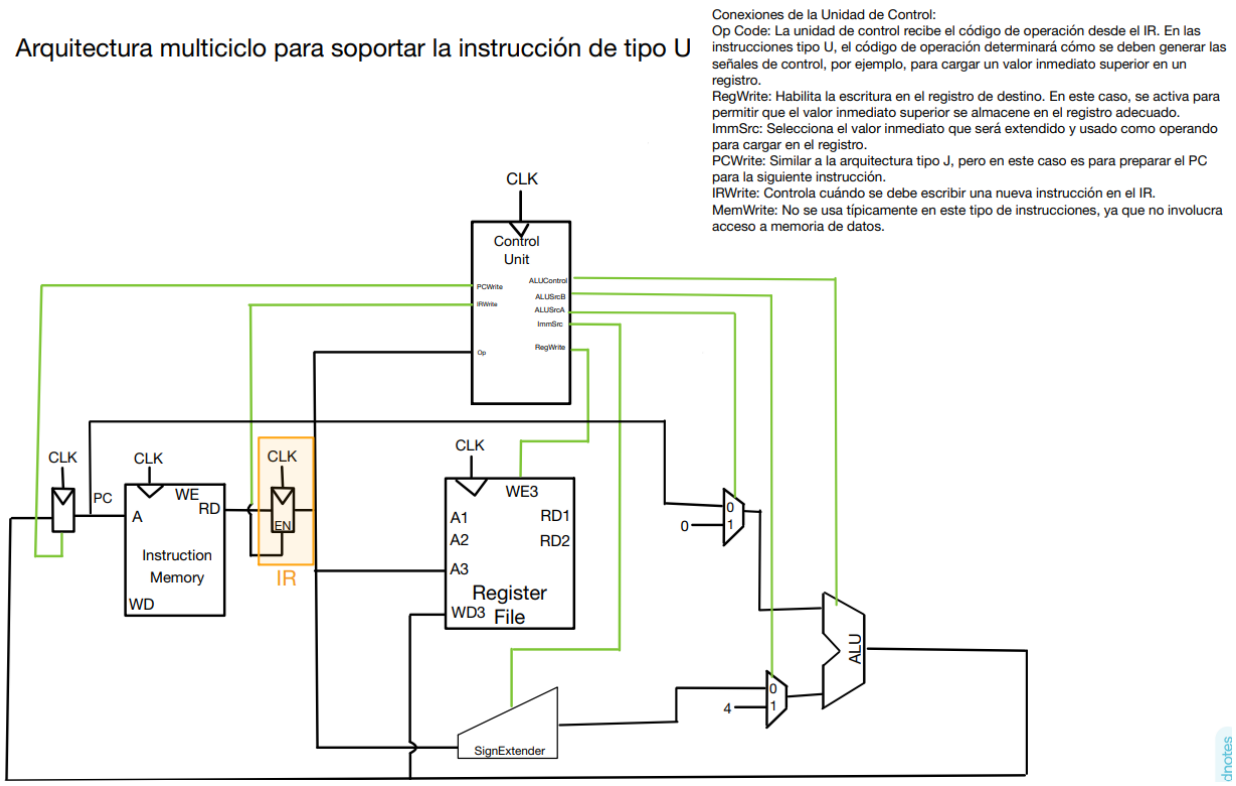


Figura 7. Diagrama para las instrucciones Tipo J. Fuente: Elaborado por las estudiantes.

Fecha: 13 de octubre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Maquina de estados para la unidad de control

Acuerdos tomados: Se presenta un diseño de la UC como maquina de estados para poder implementar en Verilog.

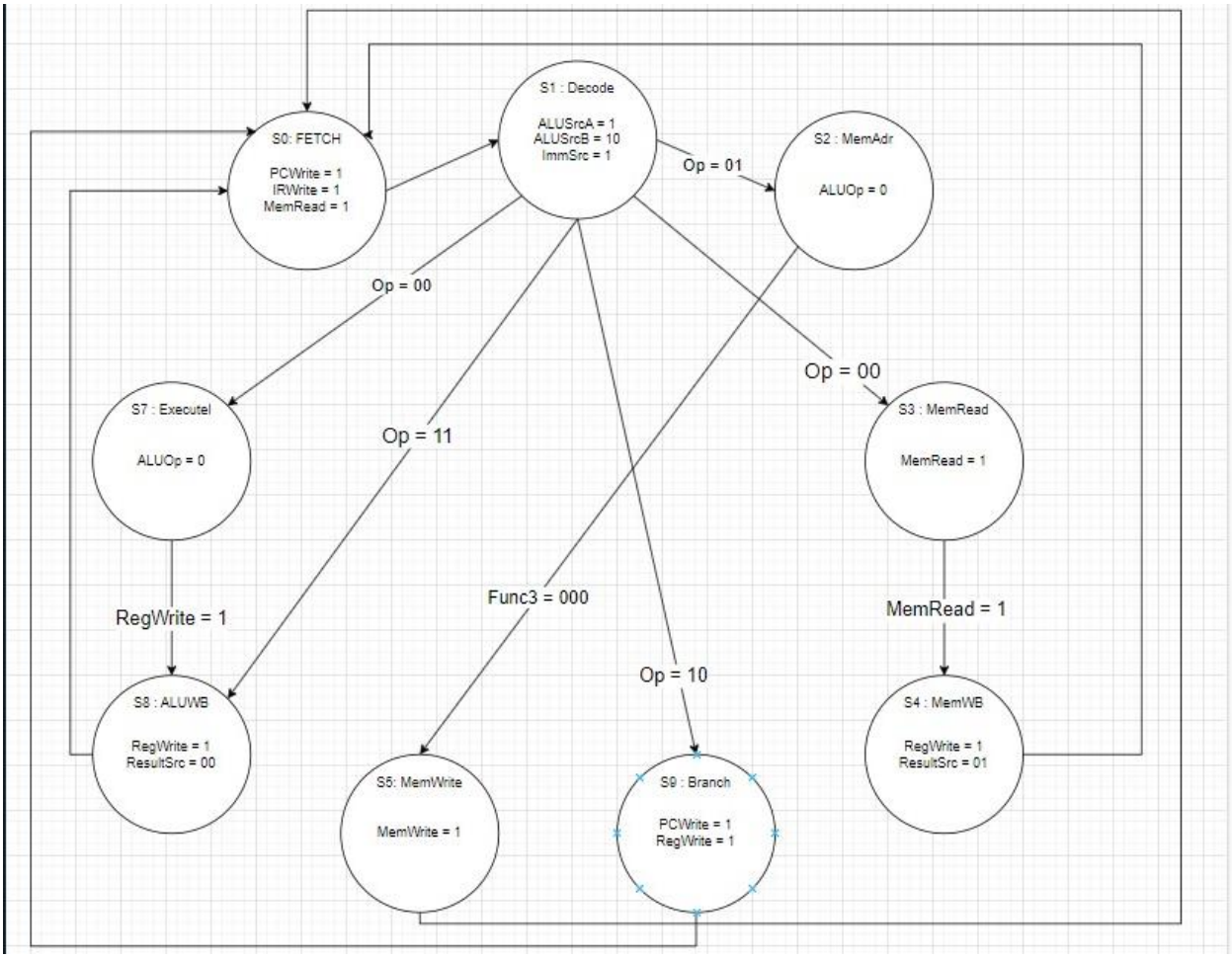


Figura 8. Diagrama de máquina de estados. Fuente: Elaborada por las estudiantes.

Fecha: 18 de octubre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Implementación en verilog de los módulos, ALU, Memoria de instrucciones, Mux y Contador PC.

Acuerdos tomados: Se realizó una reunión para desarrollar los módulos en la implementación en Verilog.

```
module ALU(
    input [31:0] A, B,
    input [3:0] alu_op,
    output reg [31:0] C
);

    always @(*) begin
        case (alu_op)
            4'b0000: C = A + B;              // Suma
            4'b0001: C = A << B[4:0];        // Shift a la izquierda
            4'b0010: C = ($signed(A) < $signed(B)) ? 32'b1 : 32'b0; // Comparación signed
            4'b0011: C = (A < B) ? 32'b1 : 32'b0; // Comparación unsigned
            4'b0100: C = A ^ B;              // XOR
            4'b0101: C = A >> B[4:0];        // Shift a la derecha lógico
            4'b0110: C = $signed(A) >>> B[4:0]; // Shift a la derecha aritmético
            4'b0111: C = A | B;              // OR
            4'b1000: C = A & B;              // AND
            4'b1001: C = A - B;              // Resta
            default: C = 32'b0;
        endcase
    end
endmodule
```

Figura 9. Código de ALU en Vivado. Fuente: Elaborada por las estudiantes.

```
module Instruction_Memory(
    input [31:0] addr,                // Dirección de la instrucción
    output reg [31:0] instruction     // Instrucción leída de la memoria
);
    // Definición de la memoria de instrucciones de 32 bits, con 30 posiciones (índices 0 a 29)
    reg [31:0] instruction_memory [0:29]; // Cambiado a 29 para incluir dos instrucciones adicionales

    // Inicialización de la memoria de instrucciones
    initial begin
        // Instrucciones iniciales
        instruction_memory[0]  = 32'h00800e23; // SW x8, 0(x1) - Guarda el contenido de x8 en la dirección 0 + x1
        instruction_memory[1]  = 32'h02010413; // ADDI x8, x2, 32 - Suma 32 a x2 y almacena el resultado en x8
        instruction_memory[2]  = 32'h0000c7b7; // LUI x15, 0xc7 - Carga 0xc7 en la parte superior de x15
        instruction_memory[3]  = 32'h7df78793; // ADDI x15, x15, 2015 - Suma 2015 a x15
        instruction_memory[4]  = 32'hfef42223; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[5]  = 32'h06300793; // ADDI x15, x0, 99 - Carga el valor 99 en x15
        instruction_memory[6]  = 32'hfef401a3; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[7]  = 32'h02100793; // ADDI x15, x0, 33 - Carga el valor 33 en x15
        instruction_memory[8]  = 32'hfef42623; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[9]  = 32'hfe042423; // SW x15, -32(x8) - Guarda x15 en la dirección x8 - 32
        instruction_memory[10] = 32'h03c0006f; // JAL x0, 60 - Salta a la dirección actual + 60
        instruction_memory[11] = 32'hfe842783; // LW x15, -24(x8) - Carga en x15 el valor en la dirección x8 - 24
        instruction_memory[12] = 32'h00079863; // BEQ x15, x0, offset - Si x15 es igual a 0, salta a la dirección de offset
        instruction_memory[13] = 32'h06200793; // ADDI x15, x0, 98 - Carga el valor 98 en x15
        instruction_memory[14] = 32'hfef401a3; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[15] = 32'h01c0006f; // JAL x0, 28 - Salta a la dirección actual + 28
        instruction_memory[16] = 32'hfec42783; // LW x15, -20(x8) - Carga en x15 el valor en la dirección x8 - 20
        instruction_memory[17] = 32'h00179793; // ADDI x15, x15, 1 - Incrementa x15 en 1
        instruction_memory[18] = 32'hfef42623; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[19] = 32'hfec42783; // LW x15, -20(x8) - Carga en x15 el valor en la dirección x8 - 20
        instruction_memory[20] = 32'h00f7f793; // ANDI x15, x15, 15 - Realiza una AND con 15 y almacena el resultado en x15
        instruction_memory[21] = 32'hfef42623; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[22] = 32'h00178793; // ADDI x15, x15, 1 - Incrementa x15 en 1
        instruction_memory[23] = 32'hfef42423; // SW x15, -16(x8) - Guarda x15 en la dirección x8 - 16
        instruction_memory[24] = 32'hfe842703; // LW x15, -24(x8) - Carga en x15 el valor en la dirección x8 - 24
        instruction_memory[25] = 32'h00100793; // ADDI x15, x0, 1 - Carga el valor 1 en x15
        instruction_memory[26] = 32'hfce7d0e3; // BNE x15, x15, offset - Si x15 es diferente de x15, salta a offset
        instruction_memory[27] = 32'h00000793; // ADDI x15, x0, 0 - Carga el valor 0 en x15

        // Instrucciones adicionales de branch
        instruction_memory[28] = 32'h00050463; // BEQ x10, x0, offset - Si x10 es igual a x0, salta a offset
```

Figura 10. Código Memoria de instrucciones en Vivado. Fuente: Elaborada por las estudiantes.

Fecha: 20 de octubre, 2024

Participantes: Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Implementación en verilog de los módulos, Archivo de registros, Maquina de estados, PCplus4.

Acuerdos tomados: Se realizó una reunión para desarrollar los módulos en la implementación en Verilog.

```
module RegisterFile(
    input clk, reset, reg_wr,
    input [4:0] raddr1, raddr2, waddr,
    input [31:0] wdata,
    output reg [31:0] rdata1, rdata2
);
    reg [31:0] registerfile [31:0];

    // Reinicia el banco de registros cuando se activa el reset
    integer i;
    always @(posedge clk) begin
        if (reset) begin
            for (i = 0; i < 32; i = i + 1)
                registerfile[i] <= 32'b0;
        end else if (reg_wr && (waddr != 0)) begin
            // Solo permite escritura si reg_wr está activo y la dirección no es 0
            registerfile[waddr] <= wdata;
        end
    end

    always @(*) begin
        // Realiza la lectura asincrónica
        rdata1 = registerfile[raddr1];
        rdata2 = registerfile[raddr2];
    end
endmodule
```

Figura 11. Código de Archivo de registros en Vivado. Fuente: Elaborada por las estudiantes.

```
module PC_Plus4 (
    input [31:0] A,         // Entrada (valor al que se le sumará 4)
    output [31:0] B         // Salida (resultado de A + 4)
);
    assign B = A + 4;
endmodule
```

Figura 12. Código de PCplus4 en Vivado. Fuente: Elaborada por las estudiantes.

Se tuvo problemas con el módulo de la máquina de estados por lo cual no se adjunta resultado.

Fecha: 22 de octubre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Toma de decisión de cambio de arquitectura y realización y corrección de diagramas de nueva arquitectura.

Acuerdos tomados: Se tomó la decisión de utilizar la arquitectura uniciclo ya que era más sencilla de implementar, y además se empezó nuevamente con los diagramas conociendo la base de estos pero enfocándose en los diagramas de arquitectura uniciclo, y corrigiendo los que ya se tenian.

Los diagramas realizados se adjuntan a continuación:

B-Type

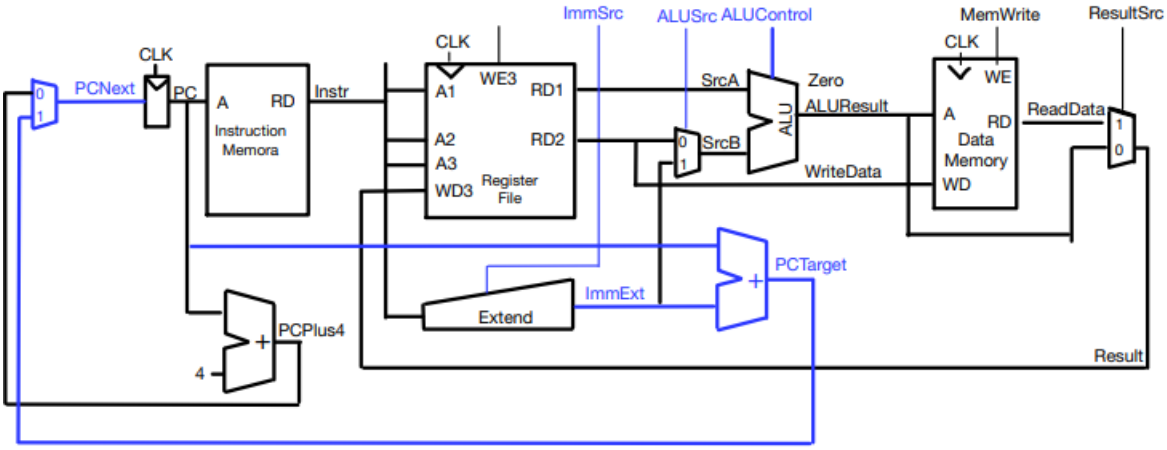


Figura 13. Diagrama para las instrucciones Tipo B Uniciclo. Fuente: Elaborado por las estudiantes.

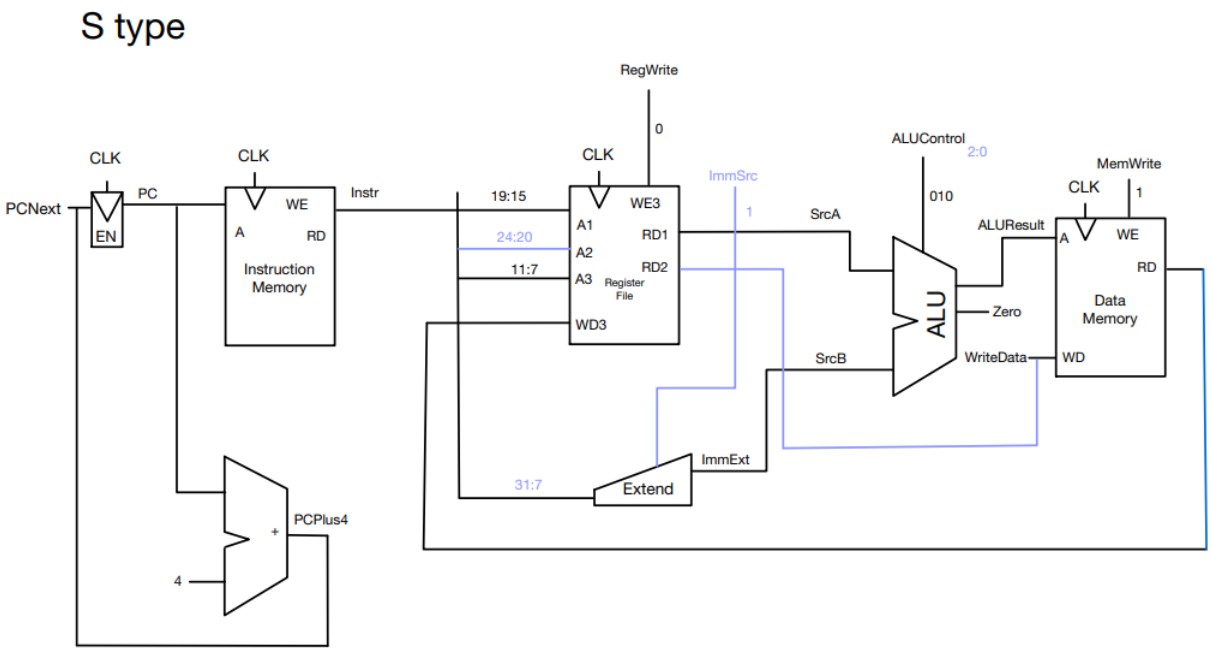


Figura 14. Diagrama para las instrucciones Tipo S Uniciclo. Fuente: Elaborado por las estudiantes.

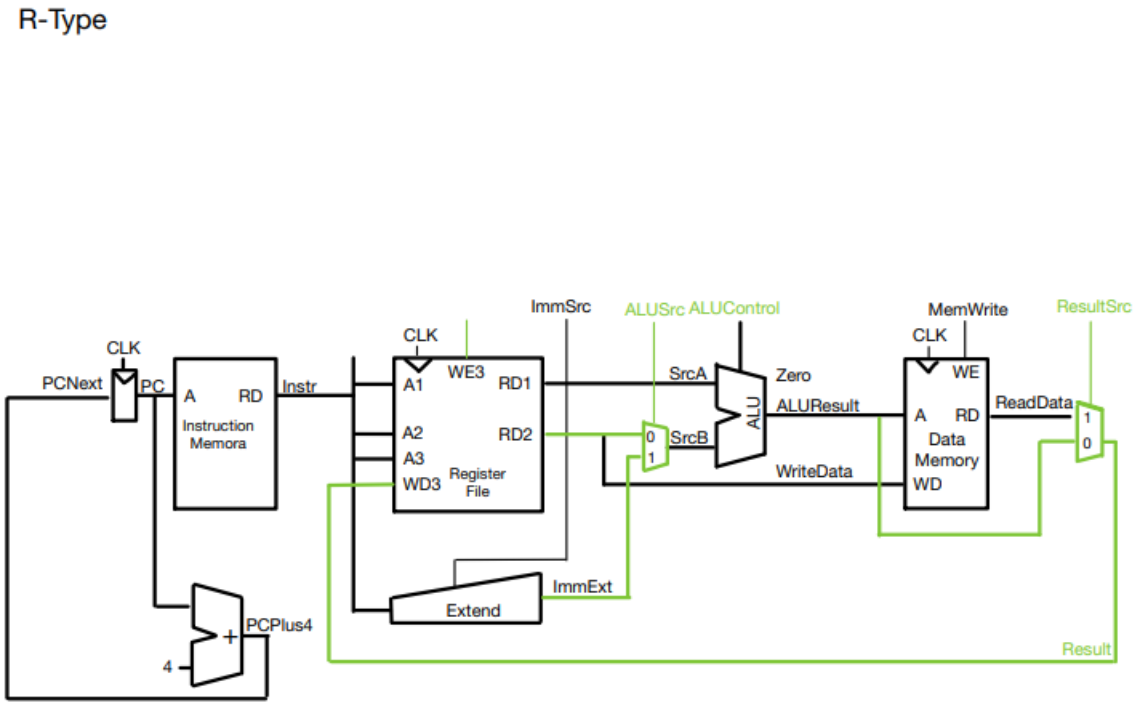


Figura 15. Diagrama para las instrucciones Tipo S Uniciclo. Fuente: Elaborado por las estudiantes.

Fecha: 27 de octubre, 2024

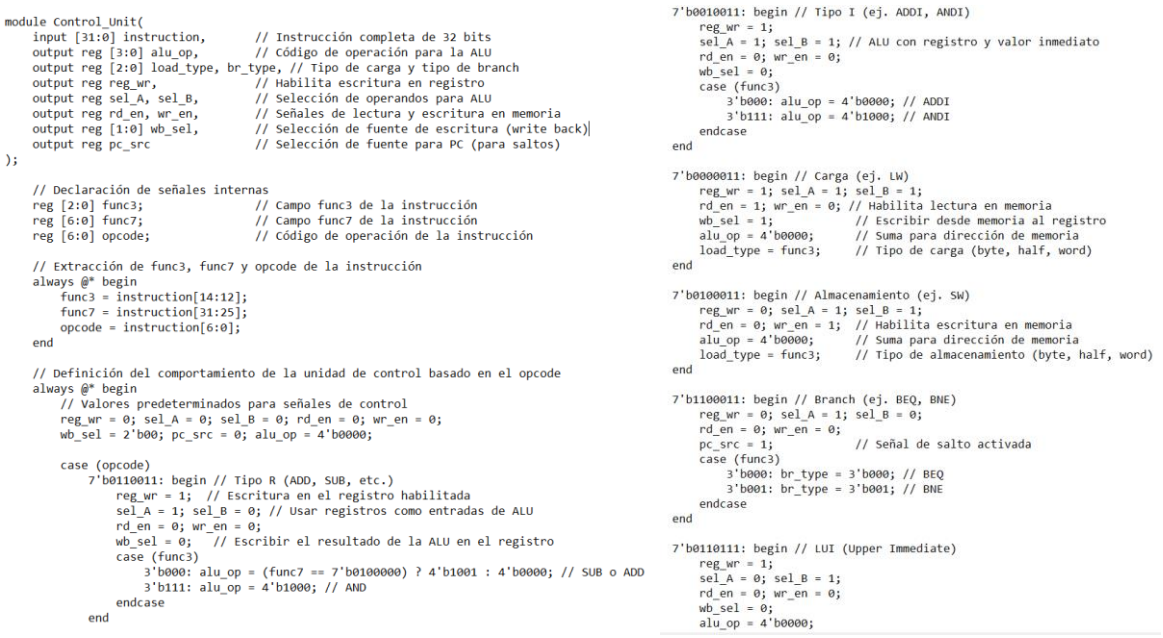
Participantes: Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Implementación en verilog de correcciones para arquitectura uniciclo.

Acuerdos tomados: Se añadió el módulo de la unidad de control necesaria para la arquitectura uniciclo y se corrige todo lo que no ajusta correctamente.



```
module Control_Unit(
    input [31:0] instruction,       // Instrucción completa de 32 bits
    output reg [3:0] alu_op,        // Código de operación para la ALU
    output reg [2:0] load_type, br_type, // Tipo de carga y tipo de branch
    output reg reg_wr,              // Habilita escritura en registro
    output reg sel_A, sel_B,        // Selección de operandos para ALU
    output reg rd_en, wr_en,        // Señales de lectura y escritura en memoria
    output reg [1:0] wb_sel,        // Selección de fuente de escritura (write back)
    output reg pc_src               // Selección de fuente para PC (para saltos)
);

    // Declaración de señales internas
    reg [2:0] func3;                // Campo func3 de la instrucción
    reg [6:0] func7;                // Campo func7 de la instrucción
    reg [6:0] opcode;               // Código de operación de la instrucción

    // Extracción de func3, func7 y opcode de la instrucción
    always @* begin
        func3 = instruction[14:12];
        func7 = instruction[31:25];
        opcode = instruction[6:0];
    end

    // Definición del comportamiento de la unidad de control basado en el opcode
    always @* begin
        // Valores predeterminados para señales de control
        reg_wr = 0; sel_A = 0; sel_B = 0; rd_en = 0; wr_en = 0;
        wb_sel = 2'b00; pc_src = 0; alu_op = 4'b0000;

        case (opcode)
            7'b0110011: begin // Tipo R (ADD, SUB, etc.)
                reg_wr = 1;  // Escritura en el registro habilitada
                sel_A = 1; sel_B = 0; // Usar registros como entradas de ALU
                rd_en = 0; wr_en = 0;
                wb_sel = 0;   // Escribir el resultado de la ALU en el registro
                case (func3)
                    3'b000: alu_op = (func7 == 7'b0100000) ? 4'b1001 : 4'b0000; // SUB o ADD
                    3'b111: alu_op = 4'b1000; // AND
                endcase
            end

    7'b0010011: begin // Tipo I (ej. ADDI, ANDI)
        reg_wr = 1;
        sel_A = 1; sel_B = 1; // ALU con registro y valor inmediato
        rd_en = 0; wr_en = 0;
        wb_sel = 0;
        case (func3)
            3'b000: alu_op = 4'b0000; // ADDI
            3'b111: alu_op = 4'b1000; // ANDI
        endcase
    end

    7'b0000011: begin // Carga (ej. LW)
        reg_wr = 1; sel_A = 1; sel_B = 1;
        rd_en = 1; wr_en = 0; // Habilita lectura en memoria
        wb_sel = 1;           // Escribir desde memoria al registro
        alu_op = 4'b0000;     // Suma para dirección de memoria
        load_type = func3;    // Tipo de carga (byte, half, word)
    end

    7'b0100011: begin // Almacenamiento (ej. SW)
        reg_wr = 0; sel_A = 1; sel_B = 1;
        rd_en = 0; wr_en = 1;  // Habilita escritura en memoria
        alu_op = 4'b0000;      // Suma para dirección de memoria
        load_type = func3;     // Tipo de almacenamiento (byte, half, word)
    end

    7'b1100011: begin // Branch (ej. BEQ, BNE)
        reg_wr = 0; sel_A = 1; sel_B = 0;
        rd_en = 0; wr_en = 0;
        pc_src = 1;           // Señal de salto activada
        case (func3)
            3'b000: br_type = 3'b000; // BEQ
            3'b001: br_type = 3'b001; // BNE
        endcase
    end

    7'b0110111: begin // LUI (Upper Immediate)
        reg_wr = 1;
        sel_A = 0; sel_B = 1;
        rd_en = 0; wr_en = 0;
        wb_sel = 0;
        alu_op = 4'b0000;
```

Figura 16. Código de unidad de control en vivado. Fuente: Elaborada por las estudiantes.

Fecha: 2 de noviembre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Revisión de diagramas finales y elaboración de últimos diagramas para arquitectura uniciclo.

Acuerdos tomados: Se decidió realiza los diagramas el mismo día ya para tener una visión más clara de lo que se debe realizar en los testbench de cada uno de los módulos. Además se inició con la elaboración del informe de proyecto.

Los diagramas realizados se adjuntan a continuación:

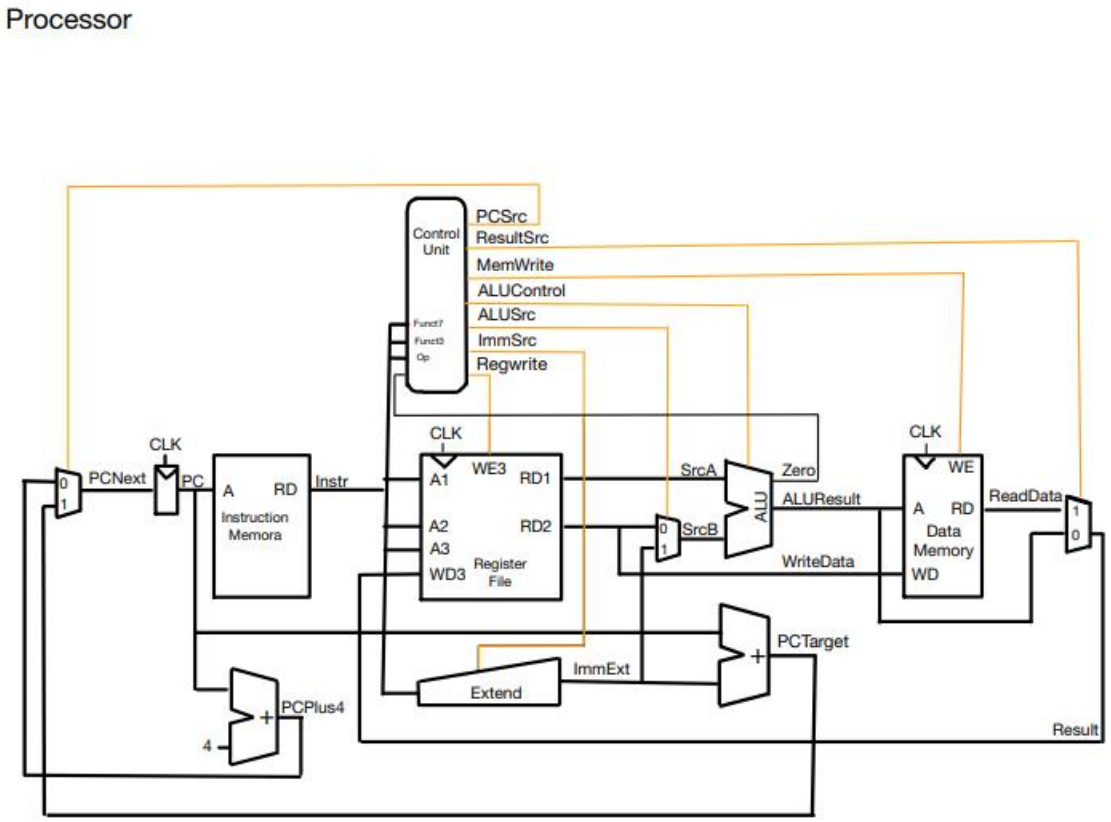


Figura 17. Diagrama para arquitectura Uniciclo. Fuente: Elaborado por las estudiantes.

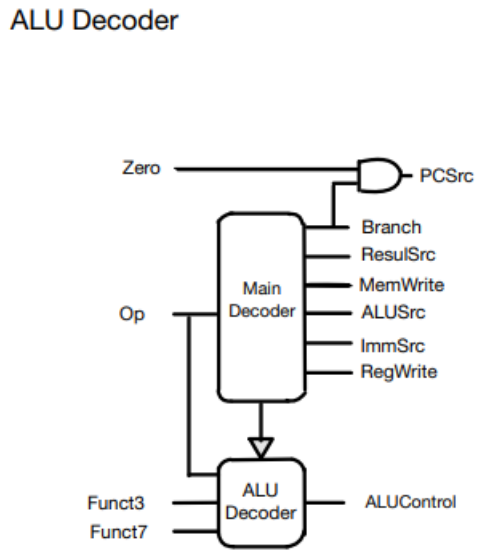


Figura 18. Diagrama para ALU de arquitectura Uniciclo. Fuente: Elaborado por las estudiantes.

Fecha: 4 de noviembre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Corrimiento de testbench para los diferentes módulos.

Acuerdos tomados: Se generaron los testbench por módulo para verificar que funcionaran adecuadamente. Se muestran algunos a continuación:

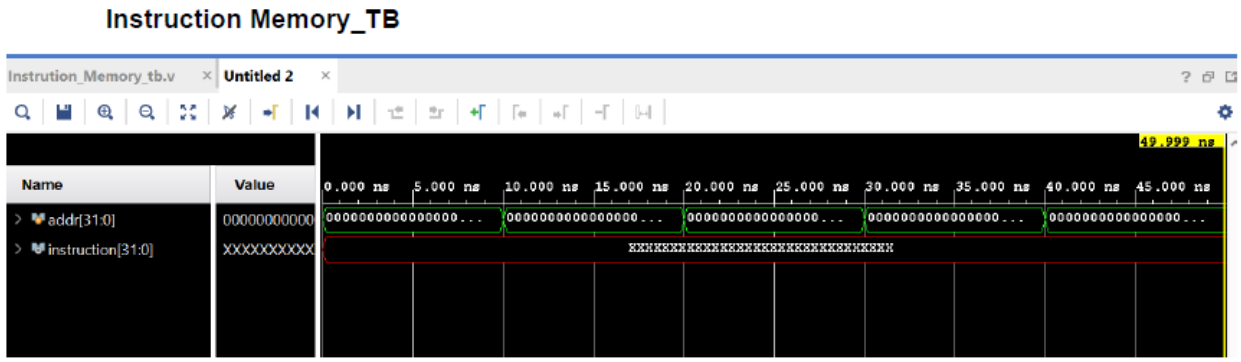


Figura 20. Testbench de la memoria de instrucciones. Fuente: Elaborada por las estudiantes.

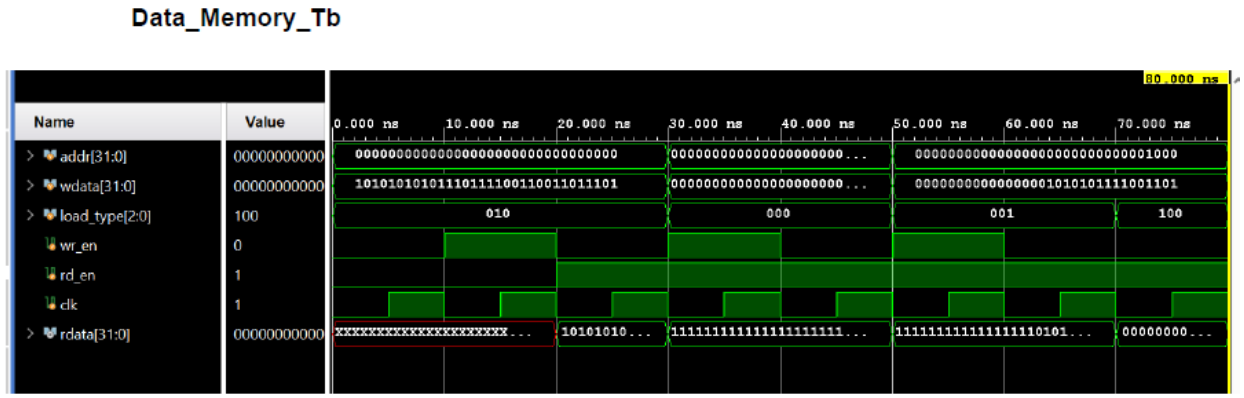


Figura 20. Testbench de la memoria de datos. Fuente: Elaborada por las estudiantes.

El principal problema que se dio fue que la memoria de instrucciones no leía correctamente el txt con las instrucciones que se deben ejecutar, por lo cual se debe corregir.

Fecha: 6 de noviembre, 2024

Participantes: Andres David Apuy Garro

Temas discutidos: Corrección de problemas leyendo las instrucciones del txt.

Acuerdos tomados: Se corrige el problema haciendo-----:

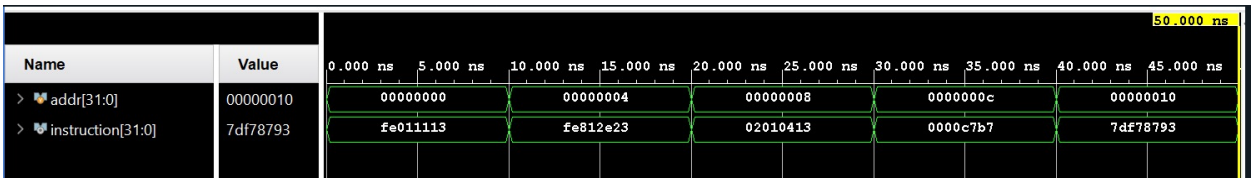


Figura 20. Testbench de la memoria de instrucciones funcionando bien. Fuente: Elaborada por las estudiantes.

Con esto se pudo proceder con las pruebas necesarias para verificar que el sistema está listo para funcionar correctamente.

Fecha: 10 de noviembre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Informe del proyecto y revisión de problema en que se encicla el valor del mem_data_out en el main por lo cual no avanza correctamente en los resultados de la ejecución de instrucciones.

Acuerdos tomados: Se redacta el informe completo y se empezó a corregir los diagramas.

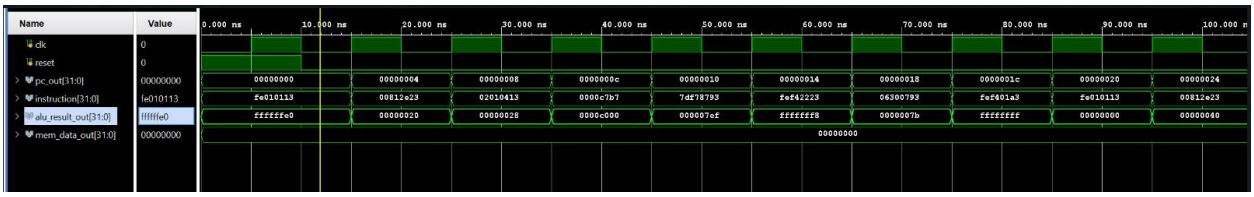


Figura 22. Testbench del main con problema del mem_data_out. Fuente: Elaborada por las estudiantes.

Fecha: 13 de noviembre, 2024

Participantes: Andres David Apuy Garro

Diego Brenes Poveda

Fátima Cárdenas Obando

Brandy Juliana Jiménez Delgado

Bruno Ramses Mora Villalobos

Ana Victoria Rojas Lazo.

Temas discutidos: Se verifica a pie que los resultados del testbench con las instrucciones esté retornando y realizando lo correspondiente según la instrucción.

Acuerdos tomados: Se verificó que efectivamente las instrucciones se corren de manera correcta. La corrección de los diagramas para las instrucciones tipo B y el general.

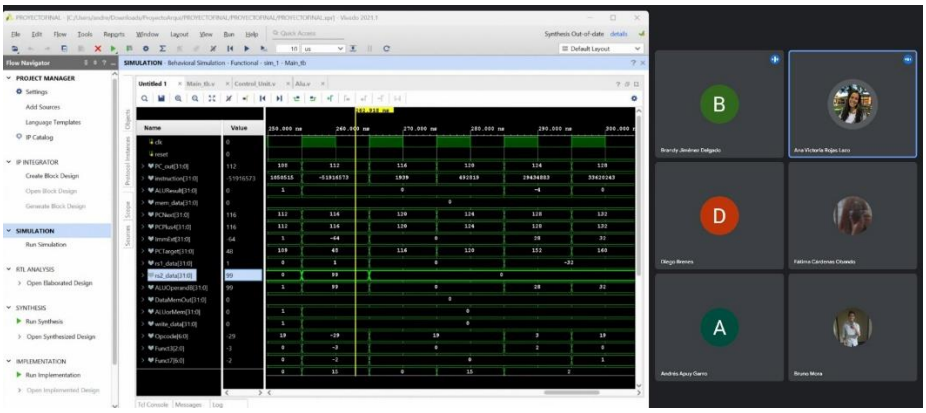


Figura 23. Reunión de verificación de testbench contra instrucciones.

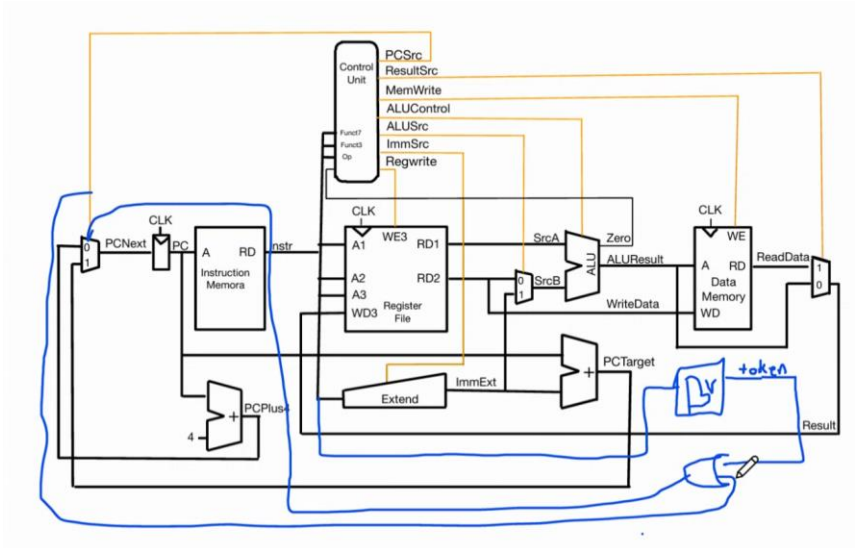


Figura 24. Corrección de diagrama general de la microarquitectura como ejemplo.

Fecha: 15 de noviembre, 2024

Participantes: Brandy Juliana Jiménez Delgado

Temas discutidos: Diagramas corregidos según los cambios hechos.

Acuerdos tomados: Se tienen los diagramas finales de la microarquitectura completa y de las instrucciones tipo B.

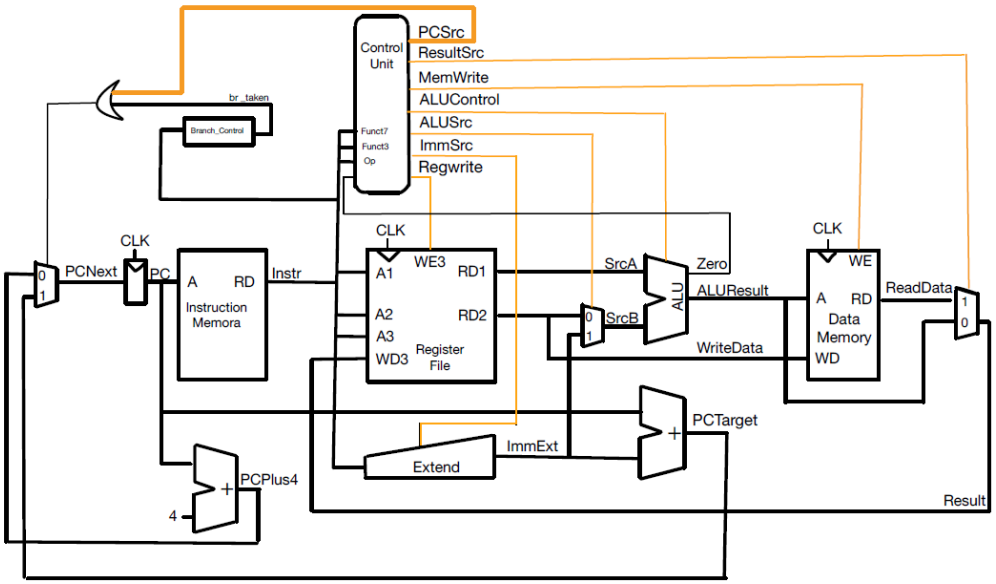


Figura 24. Diagrama final de microarquitectura implementada.

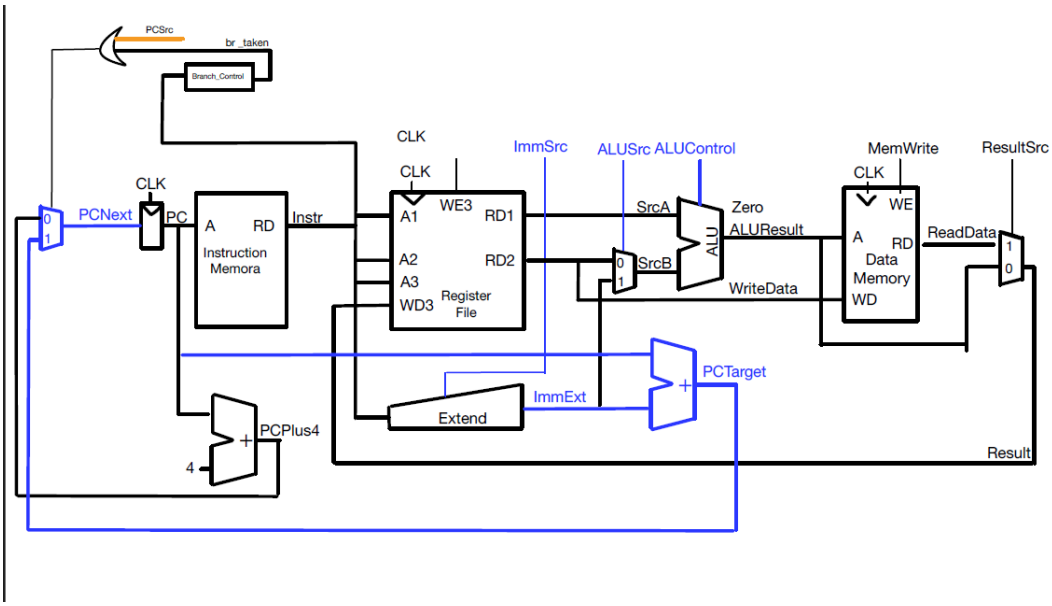


Figura 25. Diagrama final para instrucciones tipo B de microarquitectura implementada.