FASE 1
PROYECTO DE SEMINARIO DE ARQUITECTURA

EQUIPO:3

BRANCO MUSICH FLORES

DANIEL SANCHEZ ZEPEDA

JUAN PABLO CALZADA AVALOS

INTRODUCCION

Bueno a lo largo de este proyecto mi equipo y yo decimos organizarnos para tener dos reuniones para ponernos de acuerdo para la realización de la fase 1,Mi compañero musich en esta ocasión le toco hacer el código de algunos modulos y Juan pablo le toco la parte del reporte y a mi Daniel me toco ser el Admin lo cual me puse de acuerdo con mis compañeros para realizar una llamada de discord el dia 3 Y 4 de diciembre por la tarde lo cual fue la primera comunicación para que me explicaran y me implementaran su trabajo que llevaban para asi yo poder realizar la presentación debida a continuación les presentare la información recolectada de mis compañeros hasta el momento

ARQUITECTURA MIPS

La arquitectura MIPS32 (Million Instructions Per Second) es una arquitectura estándar de alto rendimiento y eficiencia que es la base de millones de productos electrónicos, desde simples microcontroladores hasta equipos de networking de alta gama.

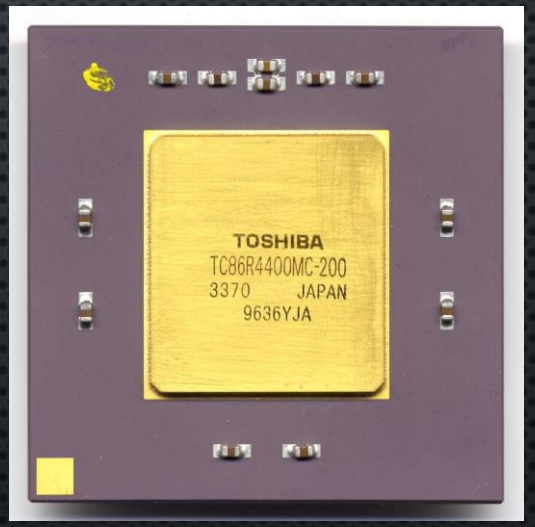
Esta arquitectura cuenta con un set de instrucciones robusto, que es escalable desde 32 hasta 64 bits., una amplia gama de herramientas de desarrollo y amplio soporte de muchas empresas y licencias.
Esta arquitectura incluye distintas funciones importantes como el SIMD (Single Instruction Multiple Data) y virtualización. Dichas tecnologías, en conjunto con otras como el multi-threading (MT), extensiones DSP y EVA (Enhanced Virtual Addressing); hacen posible que esta arquitectura siga siendo útil en las cargas de trabajo de software modernos que requieren de mayores capacidades de memoria, así como mayor velocidad en la ejecución de procesos y entornos seguros de ejecución.

MIPS32 se basa en un set de instrucciones RISC de longitud fija (32 bits) con una codificación estándar, así como un modelo de carga y almacenamiento de datos. La arquitectura está simplificada para permitir una ejecución optimizada de lenguajes de alto nivel. Las operaciones lógicas y aritméticas utilizan un formato con tres operandos, permitiéndole a los compiladores optimizar la formulación de expresiones complejas. La disponibilidad de 32 registros de uso general le permite a los compiladores optimizar aún más la generación de código al guardar datos usados frecuentemente en registros.

SET DE INSTRUCCIONES

El set de instrucciones contiene una gran variedad de operaciones, entre las más importantes se encuen

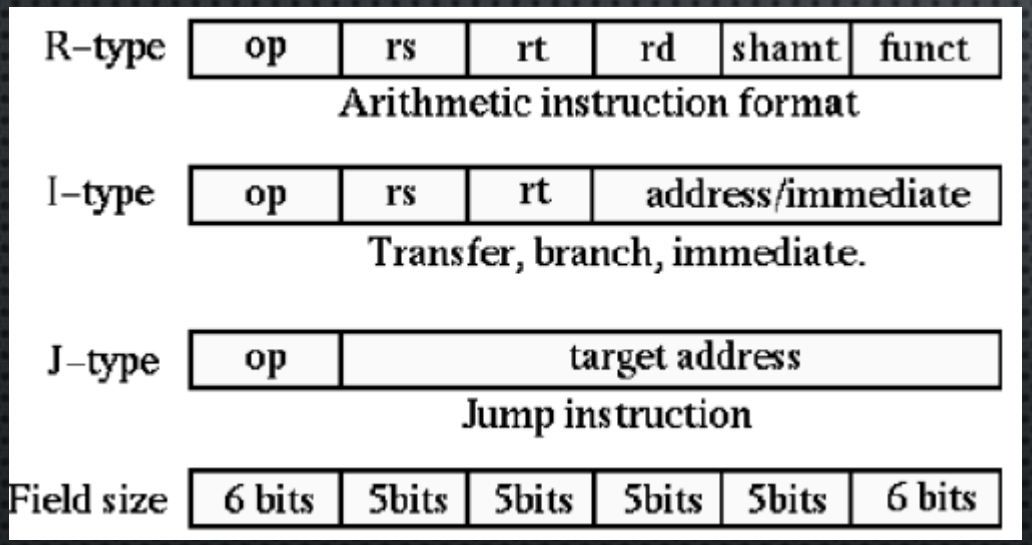
- 21 instrucciones aritméticas.
- 8 instrucciones lógicas.
- 8 instrucciones para la manipulación de bits.
- 12 instrucciones de comparación.
- 25 instrucciones de salto y branch.
- 15 instrucciones de carga (load).
- 10 instrucciones de almacenamiento.
- 8 instrucciones de desplazamiento.
- 4 instrucciones misceláneas.



DE MANERA GENERAL, LAS INSTRUCCIONES SE PUEDEN SEPARAR EN TRES CATEGORÍAS DISTINTAS:

- TIPO R: EJECUTAN OPERACIONES ARITMÉTICAS Y LÓGICAS.
- TIPO I: TRANSFIEREN DATOS ENTRE REGISTROS, EJECUTAN OPERACIONES ARITMÉTICO/LÓGICAS INMEDIATAS, SE ENCARGAN DE HACER BRANCHES.
- TIPO J: REALIZAN SALTOS UNA INSTRUCCIÓN A OTRA.

CADA TIPO DE INSTRUCCIÓN DISTRIBUYE LOS 32 BITS DE MANERA DISTINTA:

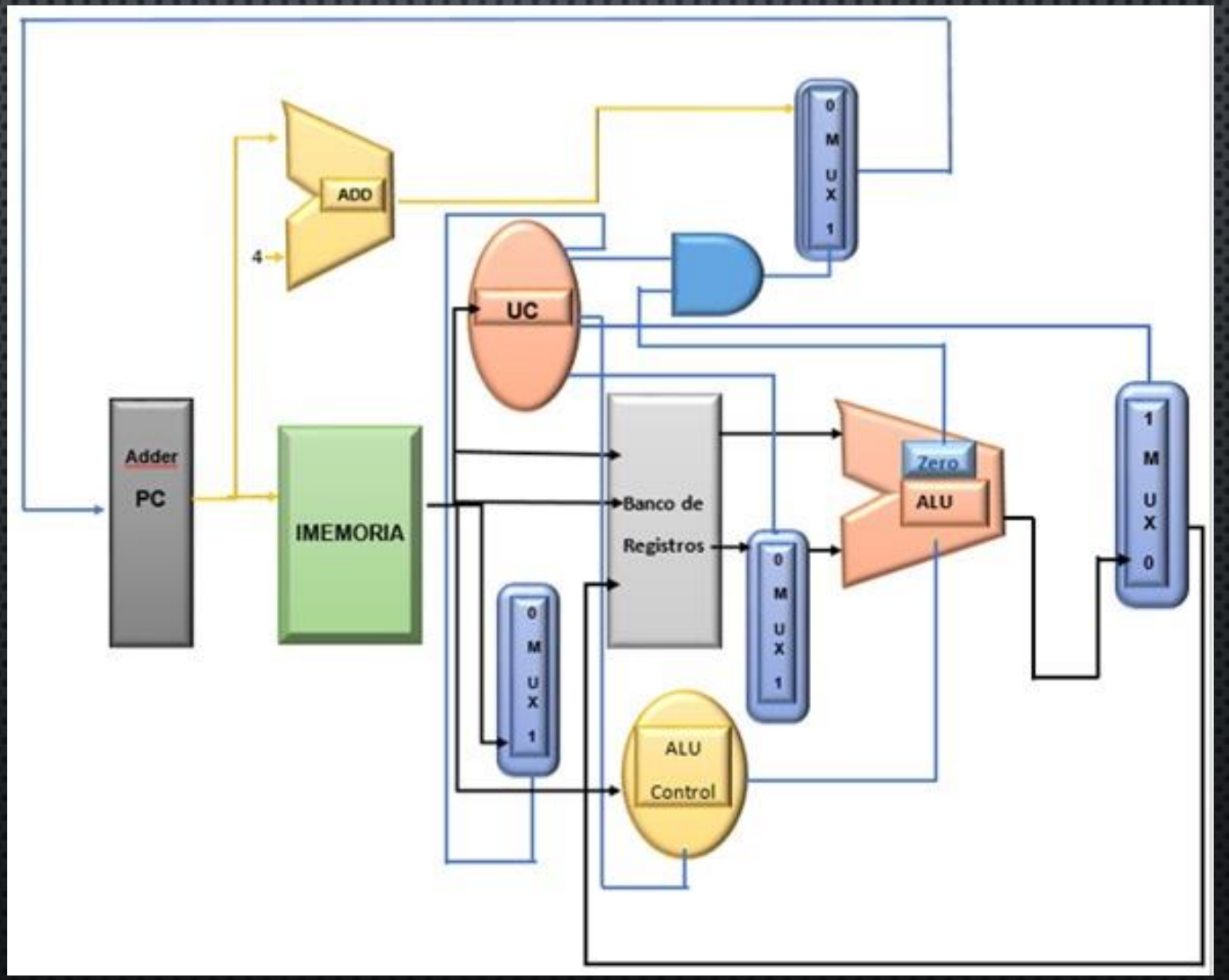


Separación de los bits en las instrucciones MIPS

INSTRUCCIONES MIPS

Instrucción	Tipo	Sintaxis
Add	R	Add $rd, $rs, $rt
Sub	R	Sub $rd, $rs, $rt
Mul	R	Mul $rd, $rs, $rt
Div	R	Div $rs, $rt
Or	R	Or $rd, $rs, $rt
And	R	And $rd, $rs, $rt
Slt	R	Slt $rd, $rs, $rt
Nop	R	NOP
addi	I	addi $rs,$rt,imm
subi	I	subi $rs,$rt,imm
ori	I	ori $rt,$rs,imm
andi	I	andi $rt,$rs,imm
lw	I	lw $rt,offset(base)
sw	I	sw $rt,offset(base)
slti	I	slti $rt,$rs,inmediato
beq	I	beq $rs,$rt,offset
bne	I	bne $rs, $rt, imm
bgtz	I	bgtz $rs,imm
adiu	I	adiu $rt,$rs,imm
J	J	j destino

PROYECTO FINAL FASE 1



CODIGOS IMPLEMENTADOS PARA LA FASE 1

C:/Users/Branco/Desktop/PROYECTOfINALpROFEsEMama/UC.v (/tb_dataPathF

```
Ln#
 1  `timescale 1ns/1ns
 2  module UC(
 3  input [5:0] opCode,
 4  output reg amemToReg, regWrite, amemToWrite,
 5  output reg branch, ALUSrc, amemToRead, regDist,
 6  output reg [2:0]aluOp
 7  );
 8
 9  always @(*)
10  begin
11          case(opCode)
12          6'b000000://Istrucciones tipo R
13          begin
14          aluOp = 3'b000;
15          amemToReg = 1'b0;
16          amemToWrite = 1'b0;
17          regWrite = 1'b1;
18          branch = 1'b1;
19          ALUSrc = 1'b0;
20          amemToRead = 1'b0;
21          regDist = 1'b1;
22          end
23          default//Instrucciones tipo ?
24          begin
25          aluOp = 3'b111;
26          amemToReg = 1'b1;
27          amemToWrite = 1'b1;
28          regWrite = 1'b1;
29          branch = 1'b0;
30          ALUSrc = 1'b1;
31          amemToRead = 1'b1;
32          regDist = 1'b0;
33          end
34          endcase
35  end
36  endmodule
37
```

C:/Users/Branco/Desktop/PROYECTOfINALpROFEsEMama/pc

```
Ln#
 1  `timescale 1ns/1ns
 2  module Pc(
 3      input [31:0] dirIn,
 4      input clk,
 5      output reg [31:0] dirOut
 6  );
 7
 8  reg [31:0] pc1;
 9
10  always@(posedge clk)
11  begin
12      if(clk == 1)
13          begin
14              dirOut = pc1;
15              pc1 = dirIn;
16          end
17  end
18  initial begin
19      pc1 = 0;
20  end
21  endmodule
22
```

C:/Users/Branco/Desktop/PROYECTOfINALpROFEsEMama/adderPc.v (/tb_dataPath

```
Ln#
 1  `timescale 1ns/1ns
 2
 3  module AdderPc(
 4      input [31:0] pcIn,
 5      output reg[31:0] dataOut
 6  );
 7  integer b4;
 8  always@(*)begin
 9      b4 = 3'b100;
10      case(b4)
11          3'b100:
12                  begin
13                          dataOut = pcIn + b4;
14                  end
15      endcase
16  end
17
18  initial
19  begin
20          dataOut = 32'd0;
21  end
22  endmodule
```

CODIGOS IMPLEMENTADOS PARA LA FASE 1

```
Ln#
 1  `timescale 1ns/1ns
 2
 3  module Imemoria(
 4  input [31:0] pcDir,
 5  output reg [31:0] instOut
 6  );
 7  //      Registro encargado de almacenar las instrucciones
 8  reg [7:0] mem_inst [511:0];
 9  //      Registros que me dice en que numero de indice voy
10  //      Ciclo encargado de mandar como una sola instruccion 4 celdas de memoria
11  always @(*)
12  begin
13      instOut = {mem_inst[pcDir], mem_inst[pcDir + 1], mem_inst[pcDir + 2], mem_inst[pcDir + 3]};
14  end
15  initial
16  begin
17  //000000_00000_00001_10100_00000_100000 SUMA $20 = $0+$1
18  mem_inst[0] = 8'b00000000;
19  mem_inst[1] = 8'b00000001;
20  mem_inst[2] = 8'b10100000;
21  mem_inst[3] = 8'b00100000;
22  //000000_00101_00110_10101_00000_100010 RESTA $21 = $5-$6
23  mem_inst[4] = 8'b00000000;
24  mem_inst[5] = 8'b10100110;
25  mem_inst[6] = 8'b10101000;
26  mem_inst[7] = 8'b00100010;
27  //000000_01010_01011_10110_00000_100100 AND $22 = $10 & $11
28  mem_inst[8] = 8'b00000001;
29  mem_inst[9] = 8'b01001011;
30  mem_inst[10] = 8'b10110000;
31  mem_inst[11] = 8'b00100100;
32  //000000_01110_01111_10111_00000_100101 OR $23 = $14 | $15
33  mem_inst[12] = 8'b00000001;
34  mem_inst[13] = 8'b11001111;
35  mem_inst[14] = 8'b10111000;
36  mem_inst[15] = 8'b00100101;
37  //000000_10010_10011_11000_00000_101010 SLT $24 = $18 < $17?1:0
38  mem_inst[16] = 8'b00000010;
39  mem_inst[17] = 8'b01010011;
40  mem_inst[18] = 8'b11000000;
41  mem_inst[19] = 8'b00101010;
42  //000000_10011_01000_11001_00000_011000 MULTIPLICACION $25 = $19 * $8
43  mem_inst[20] = 8'b00000010;
44  mem_inst[21] = 8'b01101000;
```

```
44  mem_inst[21] = 8'b01101000;
45  mem_inst[22] = 8'b11001000;
46  mem_inst[23] = 8'b00011000;
47  //000000_00011_00010_11010_00000_011010 DIVISION $26 = $3 / $2
48  mem_inst[24] = 8'b00000000;
49  mem_inst[25] = 8'b01100010;
50  mem_inst[26] = 8'b11010000;
51  mem_inst[27] = 8'b00011010;
52  //000000_10010_10011_11011_00000_000000 NOP $27 = 0
53  mem_inst[28] = 8'b00000010;
54  mem_inst[29] = 8'b01010011;
55  mem_inst[30] = 8'b11011000;
56  mem_inst[31] = 8'b00000000;
57  end
58  endmodule
```

CODIGOS IMPLEMENTADOS PARA LA FASE 1

```
Ln#
1   `timescale 1ns/1ns
2   module MUX21PC(
3   input [31:0] shiftOut,addPcOut,
4   input sel,
5   output reg [31:0] dataOut
6   );
7
8   always @(*)
9   begin
10          case(sel)
11                  1'b1:
12                  begin
13                          dataOut = shiftOut;
14                  end
15                  1'b0:
16                  begin
17                          dataOut = addPcOut;
18                  end
19          endcase
20  end
21  endmodule
22
```

C:/Users/Branco/Desktop/PROYECTOfINALpROFEsEMama/mux21BR.v (/tb_d

```
Ln#
1   `timescale 1ns/1ns
2   module MUX21BR(
3   input [31:0] signOut,brOut,
4   input sel,
5   output reg [31:0] dataOut
6   );
7
8   always @(*)
9   begin
10          case(sel)
11                  1'b1:
12                  begin
13                          dataOut = signOut;
14                  end
15                  1'b0:
16                  begin
17                          dataOut = brOut;
18                  end
19          endcase
20  end
21  endmodule
22
```

```
Ln#
1   `timescale 1ns/1ns
2   module MUX21INST(
3   input [4:0] rtOut,rdOut,
4   input sel,
5   output reg [4:0] dataOut
6   );
7
8   always @(*)
9   begin
10          case(sel)
11                  1'b1:
12                  begin
13                          dataOut = rdOut;
14                  end
15                  1'b0:
16                  begin
17                          dataOut = rtOut;
18                  end
19          endcase
20  end
21  endmodule
22
```

CODIGOS IMPLEMENTADOS PARA LA FASE 1

```
Ln#
 1  `timescale 1ns/1ns
 2  module DataPathR(
 3  input clk
 4  );
 5
 6  wire [31:0] instOut;//Memoria de instrucciones a UC, BR, MUXBR,
 7  wire regWrite;//Unidad de Control a Banco de Registro
 8  wire [31:0]RData1, RData2;//Banco de Registro a ALU y Mem, el segundo va al m
 9  wire [31:0]MuxBrOut;//MUX21BR a ALU
10  wire [2:0]UALUOp;//Unidad de Control a ALU Control
11  wire memWrite;//Unidad de Control a Mem para activar escritura
12  wire memReg;//Unidad de control a Mux21
13  wire [2:0] ALUCout;//ALU Control a ALU
14  wire [31:0] ResALU;//ALU a Mem y MUX21
15  wire [31:0] ReadData;//Mem a MUX
16  wire [31:0] MuxOut;//MUX a Banco de Registro
17  wire regDist;// Unidad de Control a MUX21InstMem
18  wire branch;//Unidad de Control a MUXPc
19  wire memRead;//Unidad de Control a Mem para activar lectura
20  wire ALUSrc;//Unidad de Control a MUX21BR
21  wire c1;//AND de branch y zeroflag
22  wire zeroF;
23  wire [31:0]Mux21Pc;//Mux21Pc a Pc
24  wire [31:0]adderOut;//adderPc a MUX21Pc
25  wire [31:0]pcOut;//Pc a memoria de instrucciones y adderPc
26  wire [4:0]muxInstOut;//MUXInstMem a banco de resgistros
27
28  Pc pc(
29  .dirIn(Mux21Pc), .clk(clk), .dirOut(pcOut)
30  );
31
32  Imemoria instMem(
33  .pcDir(pcOut), .instOut(instOut)
34  );
35
36  AdderPc add(
37  .pcIn(pcOut), .dataOut(adderOut)
38  );
39
40  UC UCDR(
41  .opCode(instOut[31:26]), .amemToReg(memReg), .regWrite(regWrite),
42  .amemToWrite(memWrite),.branch(branch), .ALUSrc(ALUSrc),
43  .amemToRead(memRead), .regDist(regDist), .aluOp(UALUOp)
44  );
```

```
43  .amemToRead(memRead), .regDist(regDist), .aluOp(UALUOp)
44  );
45
46  MUX21 mux1(
47  .memOut(ReadData), .aluOut(ResALU), .sel(memReg), .dataOut(MuxOut)
48  );
49
50  MUX21INST muxInst(
51  .rtOut(instOut[20:16]), .rdOut(instOut[15:11]), .sel(regDist), .dataOut(muxInstOut)
52  );
53
54  MUX21PC muxPc(
55  .shiftOut(32'd0), .addPcOut(adderOut), .sel(32'd0), .dataOut(Mux21Pc)
56  );
57
58  Banco_registros BR
59  (
60  .dirL1(instOut[25:21]), .dirL2(instOut[20:16]),
61  .dirW(muxInstOut),.wr(regWrite), .datoIn(MuxOut),
62  .dataOut1(RData1), .dataOut2(RData2)
63  );
64
65  MUX21BR muxBr(
66  .signOut(32'd0), .brOut(RData2), .sel(ALUSrc), .dataOut(MuxBrOut)
67  );
68
69  assign c1= branch & zeroF; //assign de la compuerta and que se conecta al muxPc
70
71  ALU_Control ALUC(.func(instOut[5:0]), .sel(UALUOp), .aluF(ALUCout));
72
73  ALU alu (.A(RData1), .B(MuxBrOut), .SEL(ALUCout), .RESULTADO(ResALU), .ZF(zeroF));
74
75  AMem mem(.dir(ResALU), .dataIn(RData2), .wr(memWrite), .rd(memRead), .dataOut(ReadData));
76
77  endmodule
```

CODIGOS IMPLEMENTADOS PARA LA FASE 1

```
Ln#
 1   `timescale 1ns/1ns
 2   module tb_dataPathR();
 3   reg clk = 0;
 4
 5   DataPathR DUV(.clk(clk));
 6
 7   always #100 clk = ~(clk);
 8
 9   initial
10   begin
11   #3300;
12   $stop;
13   end
14
15   endmodule
16
```

SIMULACION DE LOS MODULOS ANTERIORES