دانشکده مهندسی برق

گزارش کار آزمایش شماره 4

پیاده سازی تک سیکل پردازنده MIPS

محمدامین حاجی خداوردیان

97101518

استاد موحدین

نیمسال دوم 98-99

بهار99

در این آزمایش 18 دستور در پردازنده MIPS به بررسی پرداخته خواهد شد که هر دستور را به صورت جداگانه بررسی خواهیم کرد.

دستور اول:

دستور اول به صورت 0x20020005 آمده است.

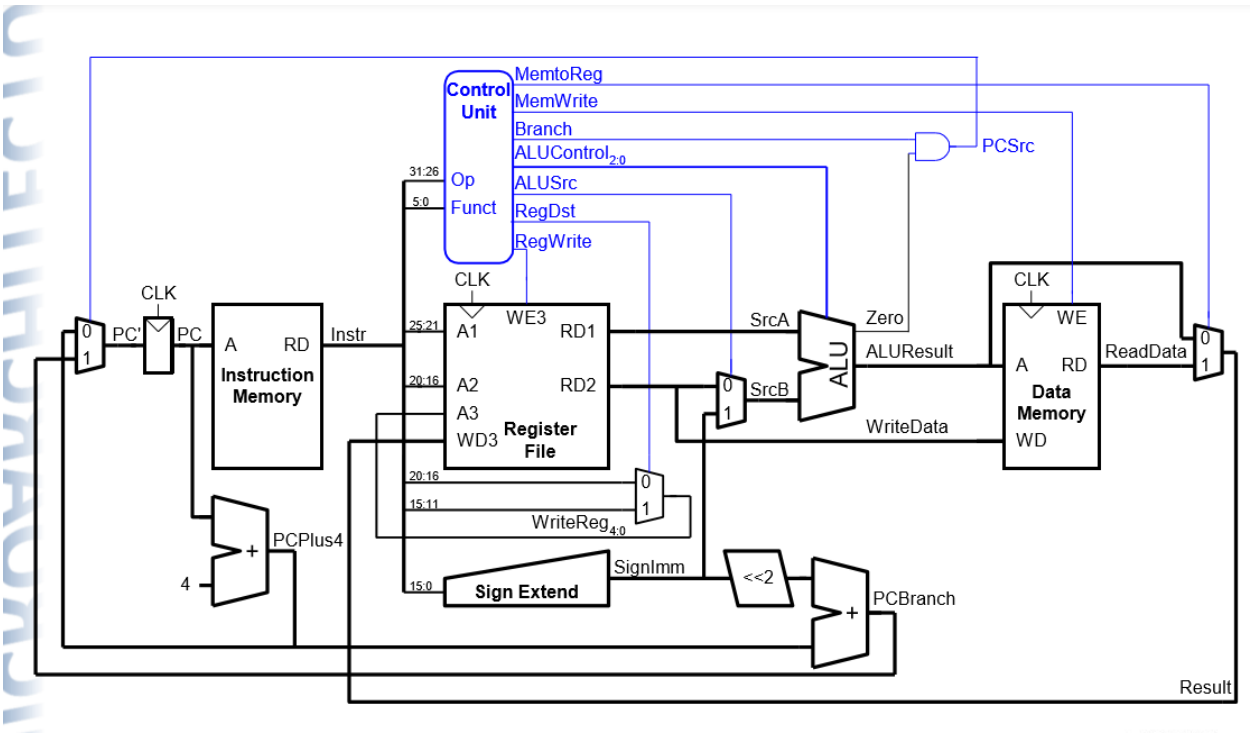
این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

001000	00000	00010	0000000000000101
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور به صورت زیراست:

Addi $rt,$rs,0x0005

حال به سراغ شبیه سازی میرویم:



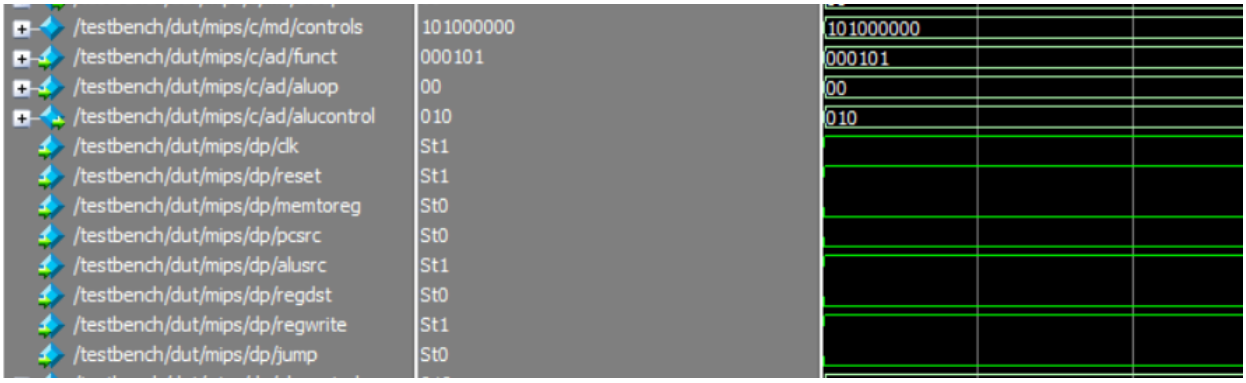
در شبیه سازی ابتدا خط های کنترلی با توجه به فیلد op تعیین میشود که این قسمت از قطعه کد زیر برداشت میشود:

```
  reg [8:0] controls;

  assign {regwrite, regdst, alusrc, branch, memwrite,
          memtoreg, jump, aluop} = controls;

  always @(*)
    case(op)
      6'b000000: controls <= 9'b110000010; // RTYPE
      6'b100011: controls <= 9'b101001000; // LW
      6'b101011: controls <= 9'b001010000; // SW
      6'b000100: controls <= 9'b000100001; // BEQ
      6'b001000: controls <= 9'b101000000; // ADDI
      6'b000010: controls <= 9'b000000100; // J
      default:   controls <= 9'bxxxxxxxxx; // illegal op
    endcase
endmodule
```

تنها بخش از کنترلر که هنوز تعیین نشده است قسمت ALUcontrol میباشد که با توجه به قطعه کد زیر تعیین میگردد:

```
module aludec(input  wire [5:0] funct,
              input  wire [1:0] aluop,
              output reg [2:0] alucontrol);

  always @(*)
    case(aluop)
      2'b00: alucontrol <= 3'b010;  // add (for lw/sw/addi)
      2'b01: alucontrol <= 3'b110;  // sub (for beq)
      default: case(funct)          // R-type instructions
          6'b100000: alucontrol <= 3'b010; // add
          6'b100010: alucontrol <= 3'b110; // sub
          6'b100100: alucontrol <= 3'b000; // and
          6'b100101: alucontrol <= 3'b001; // or
          6'b101010: alucontrol <= 3'b111; // slt
          default:   alucontrol <= 3'bxxx; // ???
        endcase
    endcase
endmodule
```

حال این مقادیر را در شبیه سازی چک میکنیم:



حال با توجه به اینکه دستور I-Format میباشد خط های کنترلی را چک میکنیم:

Memtoreg:0

از آنجایی که ما به خروجی مموری نیاز نداریم مقدار این بخش برابر 0 است

Memwrite:0

در دستور جمع نیازی به نوشتن در مموری نداریم پس 0

Branch:0

Alusrc:1

چون ما به مقدار ثابت و ساین اکستند شده نیاز داریم پس این خط کنترلی 1 است

RegDst:0

در دستورات نوع I ذخیره سازی در rt رخ میدهد که IR[20:16] میباشد

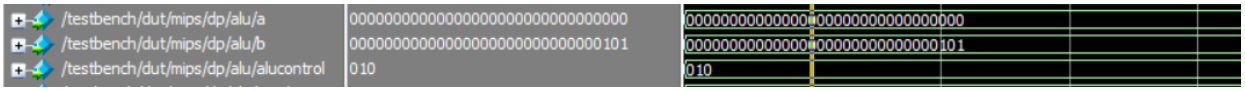
RegWrite:1

میخواهیم مقدار Aluresult در آدرس rt ذخیره شود پس باید 1 شود.

در همین حین آدرس های مورد نیاز Reg File مقدار دهی میشوند:

پس از مقداری تاخیر مقدار داده های موجود در این آدرس ها بدست می آید:

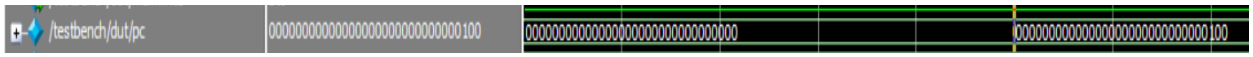
در این بخش با توجه به خطوط کنترلی مقادیری که در ALU قرار میگیرد rd1 و خروجی ساین اکستند imm-16 میباشد:



با توجه به کنترل ALU عمل جمع انجام میشود و خروجی این بخش در قسمت write data فایل Reg File قرار میگیرد و آماده میشود تا در کلاک بعدی در آدرس rt قرار گیرد.

نکته ای که به آن باید توجه داشت این است که مقدار pc در یک جمع کننده کوچک که در شکل datapath نشان داده شده است با 4 جمع میشود و آماده برای اجرای دستور بعدی میشود.

مقدار pc:0x00000004



دستور دوم:

دستور دوم به صورت 0x2003000C آمده است.

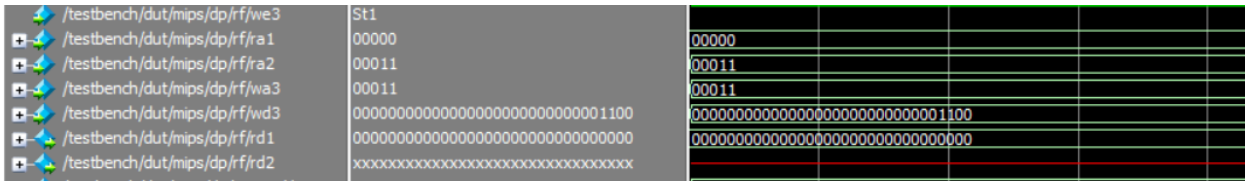
این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

001000	00000	00011	0000000000001100
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور نیز مانند دستور قبل یک addi است.

پس تمام خط های کنترلی مانند دستور اول تعیین میشوند.

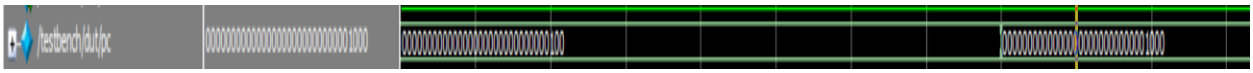
تنها تفاوت این دستور محل ذخیره سازی ALUResult میباشد که در آدرس 00011 بلوک Reg File قرار میگیرد و همچنین مقدار ثابت در این دستور 12 میباشد.



شکل بالا به طور کامل مقادیر وارد شده به Reg File را نشان میدهد که مطابق گفته های قبلی است.

در این مرحله نیز مقدار pc چون دستور j یا branch رخ نداده است پس با 4 جمع میشود و منتظر دستور بعدی میگردد.

مقدار pc در انتهای این کلاک:0x00000008



دستور سوم:

دستور سوم به صورت 0x2067fff7 آمده است.

این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

001000	00011	00111	1111111111110111
op	**rs**	**rt**	**Imm-16**

این دستور نیز یک دستور addi است.

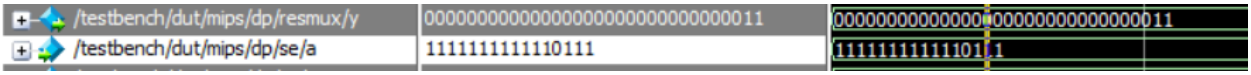
اما یک تفاوت با دستورات قبلی دارد این است که در دستورات قبلی هیچ مقداری در آدرس های داده شده به rs و rt وجود نداشت اما در این قطعه کد در آدرس rs با توجه به دستور دوم مقدار 0x0000000C ریخته شده است.

خطوط کنترلی دوباره مشابه دو دستور بالا میباشد به همین دلیل به آن اشاره نمیکنیم.

با توجه به خطوط کنترلی میدانیم که مقدارimm-16 باید در بلوک sign/zero extend ساین اکستند شود

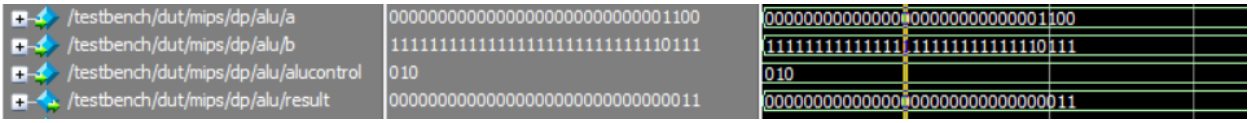
که قطعه کد و مقدار شبیه سازی شده آن را در زیرمی آوریم:

```
module signext(input  wire [15:0] a,
               output wire [31:0] y);

  assign y = {{16{a[15]}}, a};
endmodule
```

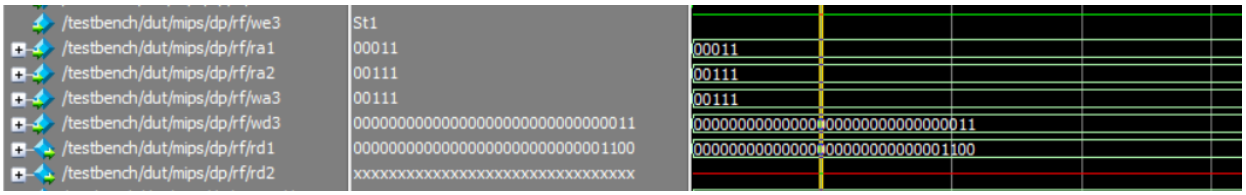


با توجه به کد بالا و خروجی شبیه سازی ساین اکستند به درستی انجام شده است.(a=0xFFF7)

حال به سراغ ورودی های ALU میرویم که طبق گفته ما باید مقادیر 0x0000000C و 0xFFFFFFF7 قرار گیرد.

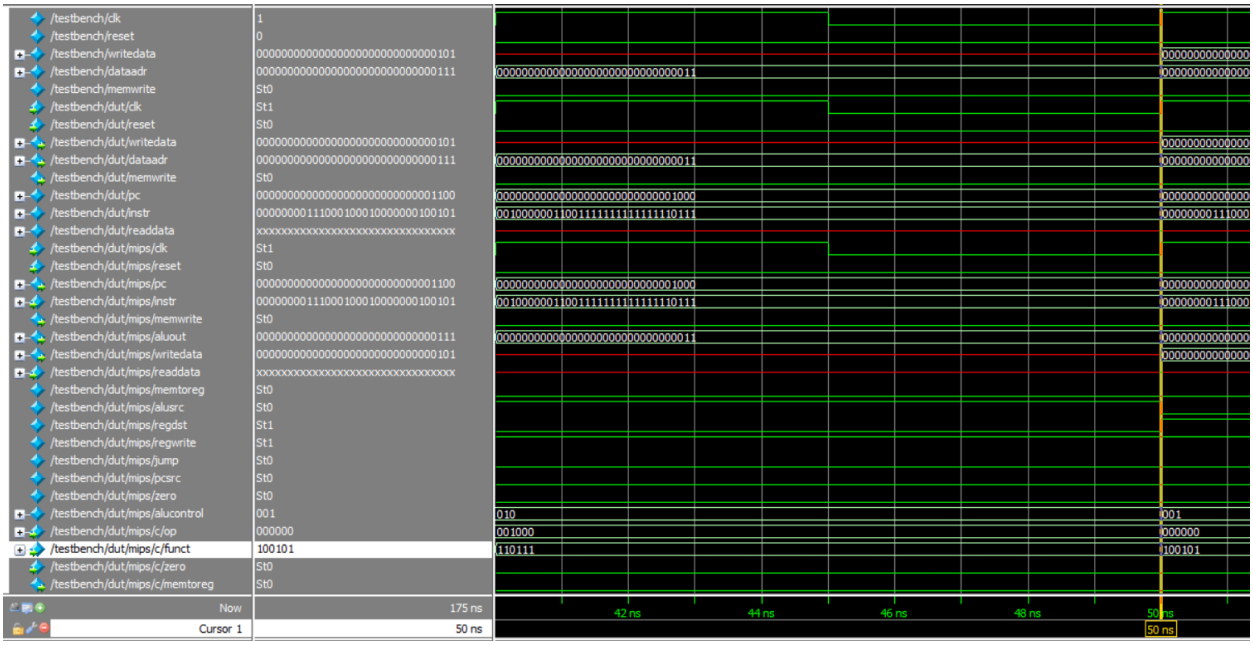


حال با توجه به شبیه سازی و دستورات ALU عمل جمع بین دو ورودی رخ داده است و خروجی ALU در write data بلوک Reg File قرار میگیرد تا در آدرس rt (0x00111) پس از کلاک نوشته شود.



حال دوباره تنها قسمتی که در این دستور به آن اشاره نکردیم pc میباشد که آن نیز مطابق قسمت قبل تنها با 4 جمع میشود

ما به این نکته اشاره نکردیم که در پیاده سازی تک سیکل هر دستور در یک پالس کلاک رخ میدهد و تمام مراحل بالا در فاصله زمانی یک کلاک رخ میدهد.(clk=10ns)



شکل بالا کاملا گفته های ما مربوط به انجام هر دستور در یک پالس کلاک را نشان میدهید که با لبه بالا رونده کلاک به سراغ دستورعمل بعدی خواهیم رفت.

مقدار pc پس از کلاک:0x0000000C

دستور چهارم:

دستور چهارم به صورت 0x00E22025 آمده است.

این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00111	00010	00100	00000	100101
op	**rs**	**rt**	**rd**	**shamt**	**func**

با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

Or $rd , $rt , $rs

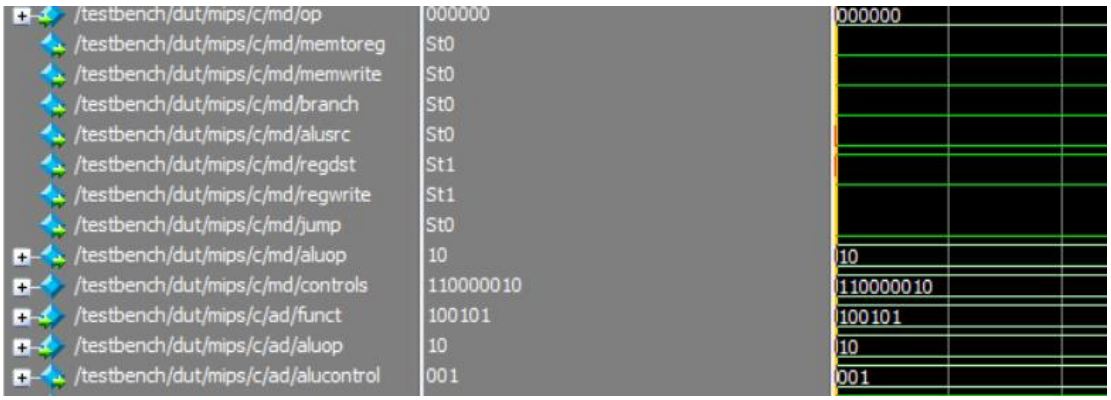
حال دوباره به خطوط کنترلی میرویم:

```
  reg [8:0] controls;

  assign {regwrite, regdst, alusrc, branch, memwrite,
          memtoreg, jump, aluop} = controls;

  always @(*)
    case(op)
      6'b000000: controls <= 9'b110000010; // RTYPE
      6'b100011: controls <= 9'b101001000; // LW
      6'b101011: controls <= 9'b001010000; // SW
      6'b000100: controls <= 9'b000100001; // BEQ
      6'b001000: controls <= 9'b101000000; // ADDI
      6'b000010: controls <= 9'b000000100; // J
      default:   controls <= 9'bxxxxxxxxx; // illegal op
    endcase
endmodule
```

با توجه به بخش بالا خطوط کنترل مشخص شده اند و تنها ALUcontrol باقی مانده است که در زیر آن نیز تعیین میشود:

```
module aludec(input  wire [5:0] funct,
              input  wire [1:0] aluop,
              output reg [2:0] alucontrol);

  always @(*)
    case(aluop)
      2'b00: alucontrol <= 3'b010;  // add (for lw/sw/addi)
      2'b01: alucontrol <= 3'b110;  // sub (for beq)
      default: case(funct)          // R-type instructions
          6'b100000: alucontrol <= 3'b010; // add
          6'b100010: alucontrol <= 3'b110; // sub
          6'b100100: alucontrol <= 3'b000; // and
          6'b100101: alucontrol <= 3'b001; // or
          6'b101010: alucontrol <= 3'b111; // slt
          default:   alucontrol <= 3'bxxx; // ???
        endcase
    endcase
endmodule
```

حال این مقادیر را در شبیه سازی چک میکنیم:



حال با توجه به اینکه دستور R-Format میباشد خط های کنترلی را چک میکنیم:

Memtoreg:0

از آنجایی که ما به خروجی مموری نیاز نداریم مقدار این بخش برابر 0 است

Memwrite:0

در دستور جمع نیازی به نوشتن در مموری نداریم پس 0

Branch:0

Alusrc:0

چون ما در اینجا رجیستر rt را نیاز داریم پس باید خط کنترلی مالتی پلکسر برای ما مقدار رجیستر را در ورودی ALU قرار دهد

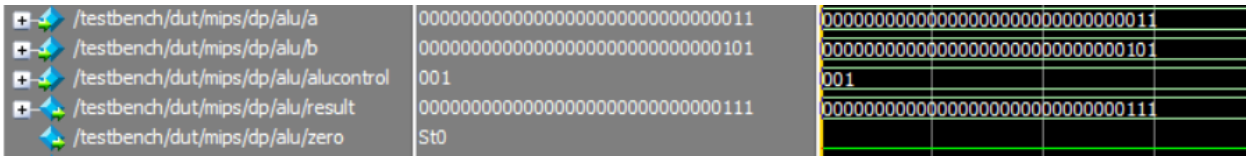
RegDst:1

در دستورات نوع R ذخیره سازی در rd رخ میدهد که IR[15:10] میباشد

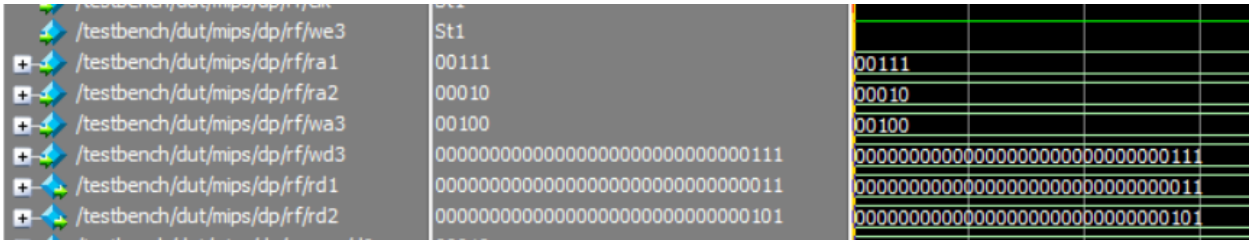
RegWrite:1

میخواهیم مقدار Aluresult در آدرس rd ذخیره شود پس باید 1 شود.

حال به بررسی بلوک ALU میپردازیم و طبق گفته ها باید مقدار آدرس 0x00111 و 0x00010 در ورودی آن قرار بگیرد که در هر دو آدرس با توجه به دستورات قبل مقادیری قرار گرفته است



حال با توجه به گفته های اول Reg File مقدار write address باید آدرس rd باشد که در شبیه سازی این موضوع را بررسی میکنیم:



حال تمام بلوک ها منتظر لبه بالا رونده کلاک میشوند تا مقادیر را بنویسند

مقدار pc پس از کلاک:0x00000010

دستور پنجم:

دستور پنجم به صورت 0x00642824 آمده است.

این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00011	00100	00101	00000	100100
op	**rs**	**rt**	**rd**	**shamt**	**func**

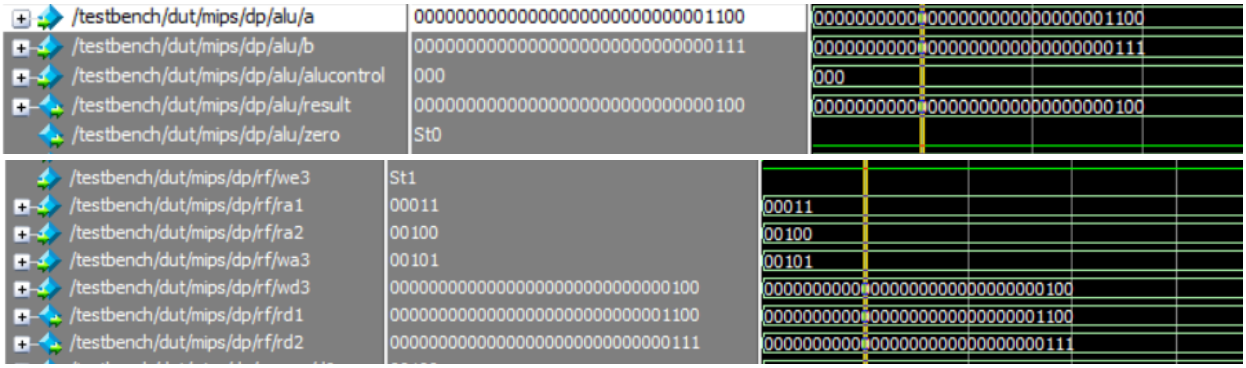
با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

And $rd , $rt , $rs

خطوط کنترلی کاملا مشابه دستور چهارم میباشد و تنها تفاوت در ALUcontrol است که در این دستور به مقدار 000 را دارد.

مانند دستور عمل قبل نیز نتیجه ALU در آدرس rd ذخیره خواهد شد.

بلوک Reg File و ALU را در شبیه سازی در دو عکس زیر آورده ایم.



در بالا مقادیری که از دستورات قبل در آدرس های rt و rs ذخیره شده بود را میتوانیم بیابیم.

تنها کاری که مانده منتظر ماندن برای کلاک برای نوشتن در آدرس مقصد است

مقدار pc پس از کلاک:0x0000014

دستور ششم:

دستور ششم به صورت 0x00A42820 آمده است.

این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00101	00100	00101	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**func**

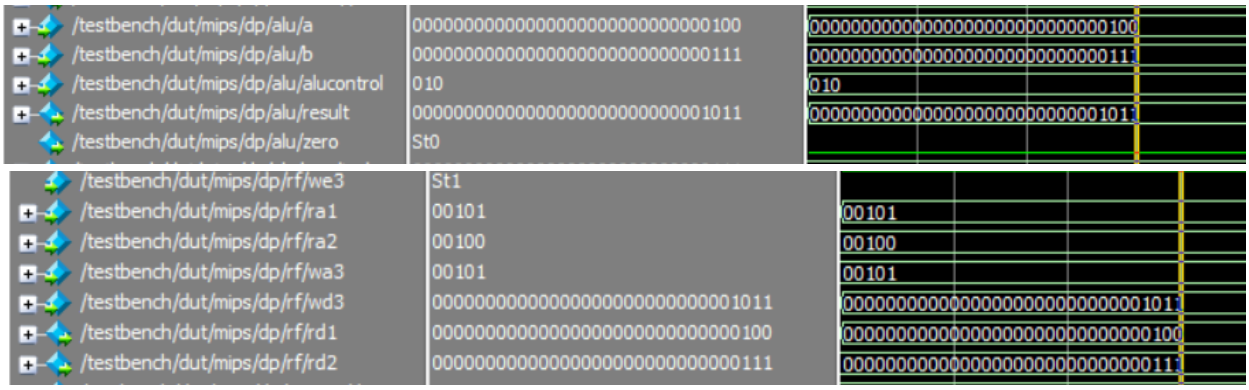
با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

Add $rd , $rt , $rs

خطوط کنترلی این دستور نیز مانند دو دستور بالا است و تنها قسمت ALU کنترلر آن عمل جمع را انجام خواهد داد.

ذخیره سازی داده ها نیز درآدرس 0x00101 رخ میدهد.

مقادیر دو بلوک Reg File و ALU در زیر آورده شده است:



با توجه به دو شکل بالا ALU خروجی به عنوان write data بلوک Reg File قرارمیگیرد.

مقدار pc پس از کلاک:0x00000018

دستور هفتم:

دستور هفتم به صورت 0x10A7000A آمده است.

این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

000100	00101	00111	0000000000001010
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور به صورت زیراست:

Beq $rs , $rt , imm-16

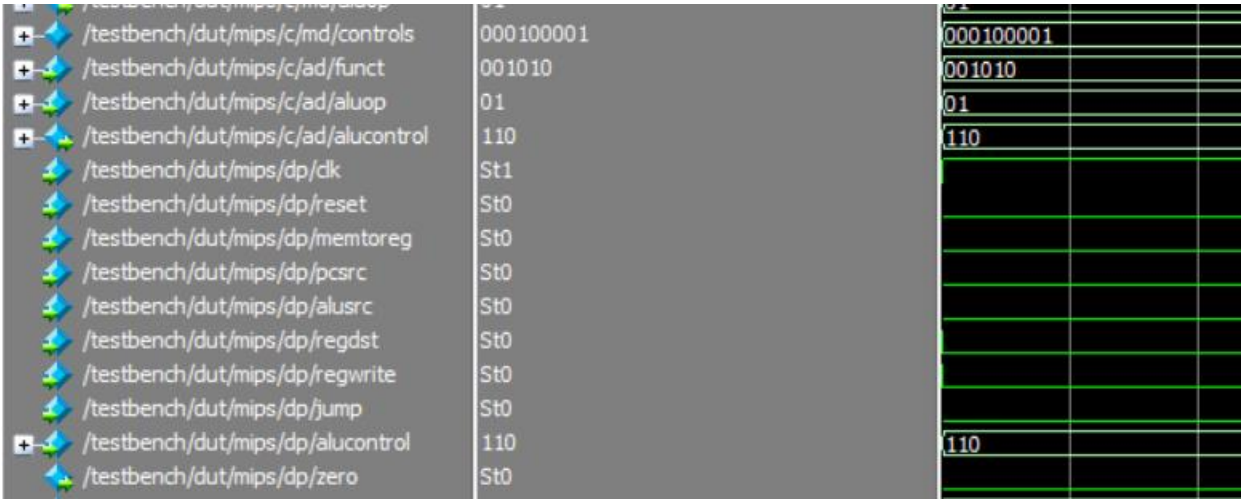
حال دوباره به خطوط کنترلی میرویم:

```
reg [8:0] controls;

assign {regwrite, regdst, alusrc, branch, memwrite,
        memtoreg, jump, aluop} = controls;

always @(*)
  case(op)
    6'b000000: controls <= 9'b110000010; // RTYPE
    6'b100011: controls <= 9'b101001000; // LW
    6'b101011: controls <= 9'b001010000; // SW
    6'b000100: controls <= 9'b000100001; // BEQ
    6'b001000: controls <= 9'b101000000; // ADDI
    6'b000010: controls <= 9'b000000100; // J
    default:   controls <= 9'bxxxxxxxxx; // illegal op
  endcase
endmodule
```

با توجه به بخش بالا خطوط کنترل مشخص شده اند و تنها ALUcontrol باقی مانده است که در زیر آن نیز تعیین میشود:

```
module aludec(input  wire [5:0] funct,
              input  wire [1:0] aluop,
              output reg [2:0] alucontrol);

  always @(*)
    case(aluop)
      2'b00: alucontrol <= 3'b010;  // add (for lw/sw/addi)
      2'b01: alucontrol <= 3'b110;  // sub (for beq)
      default: case(funct)          // R-type instructions
          6'b100000: alucontrol <= 3'b010; // add
          6'b100010: alucontrol <= 3'b110; // sub
          6'b100100: alucontrol <= 3'b000; // and
          6'b100101: alucontrol <= 3'b001; // or
          6'b101010: alucontrol <= 3'b111; // slt
          default:   alucontrol <= 3'bxxx; // ???
        endcase
    endcase
endmodule
```

حال به بررسی کد های بالا در شبیه سازی میپردازیم:



حال با توجه به اینکه دستور I-Format میباشد خط های کنترلی را چک میکنیم:

Memtoreg:0

از آنجایی که ما به خروجی مموری نیاز نداریم مقدار این بخش برابر 0 است

Memwrite:0

در دستور branch نیازی به نوشتن در مموری نداریم پس 0

Branch:1

از آنجایی که دستور branch است مقدار این خط کنترلی 1 است.

Alusrc:0

در این بخش ما به تفریق مقدار دو رجیستر داده شده نیاز داریم و خروجی zero برای ما اهمیت دارد.

RegDst:0

در این دستور ذخیره سازی رخ نمیدهد بنابراین مقدار RegDst برای ما اهمیتی ندارد ولی به طور پیش فرض در کد آن را برابر 0 قرار دادیم

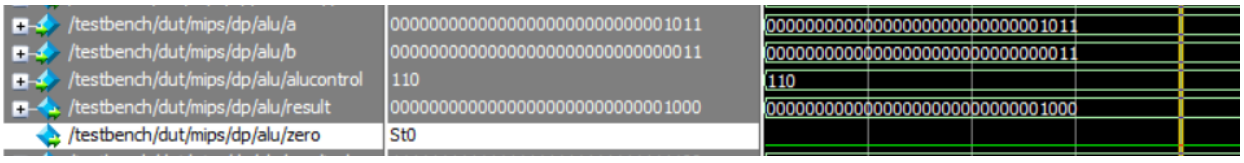
RegWrite:0

عمل نوشتنی درReg File رخ نخواهد داد پس مقدار آن 0 است.

خط کنترلی دیگری که در این دستور مهم هستند pcsrc میباشد و از آنجایی که در دستورات قبلی این مقدار برابر 0 میشد به آن اشاره نکردیم در واقع این خط کنترلی مشخص میکند که آدرس pc به اندازه 4 واحد تغییر کند یا jump رخ دهد.

برای مشخص شدن موضوع بالا باید به سراغ ALU برویم

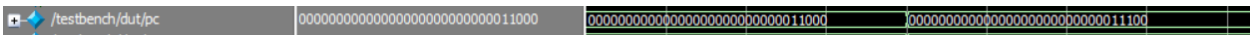
ورودی های ALU مقادیر موجود در آدرس های 0x00101 و 0x00111 میباشد. با توجه به شبیه سازی:



با توجه به نتیجه حاصل مقادیر موجود در دوآدرس با هم برابر نیستند بنابراین خط zero تغییری نمیکند

از آن جایی که خط کنترلی pcsrc با توجه به and منطقی zero و branch رخ میدهد این خط کنترلی باز صفر است و پرشی رخ نمیدهد

بنابراین تنها pc با 4 جمع میشود.



در بالا مقدار pc در کلاک فعلی و بعدی آورده شده است که با توجه به آن صحت حرف ما مشخص میشود

مقدار pc در کلاک بعدی:0x0000001C

دستور هشتم:

دستور هشتم به صورت 0x0064202A آمده است.

این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00011	00100	00100	00000	101010
op	**rs**	**rt**	**rd**	**shamt**	**func**

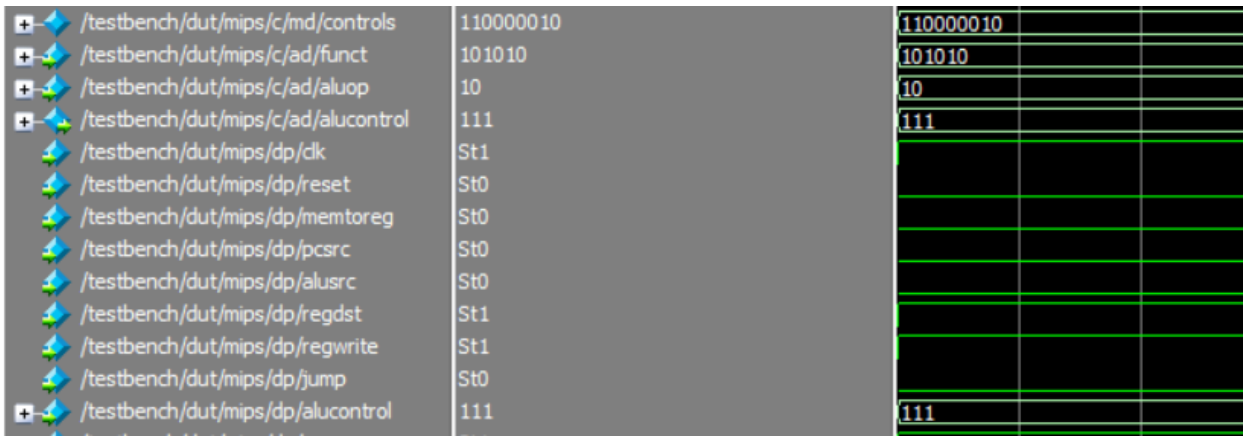
با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

Slt $rd , $rs , $rt

برای اجرای دستور بالا به قطعه کد آن رجوع میکنیم:

```
module alu(input  [31:0] a, b,
           input  [2:0]  alucontrol,
           output reg [31:0] result,
           output        zero);

  wire [31:0] condinvb, sum;

  assign condinvb = alucontrol[2] ? ~b : b;
  assign sum = a + condinvb + alucontrol[2];

  always @(*)
    case (alucontrol[1:0])
      2'b00: result = a & b;
      2'b01: result = a | b;
      2'b10: result = sum;
      2'b11: result = sum[31];
    endcase

  assign zero = (result == 32'b0);
endmodule
```

با توجه به کد بالا و از آن جایی که کنترل alu را برای slt 11 قرار دادیم به این صورت عمل میکنیم که عدد دوم را از عدد اول کم میکنیم اگر حاصل منفی شد خروجی را 1 و اگر مثبت شد خروجی را صفر میکنیم که این همان علامت حاصل تفریق است.



حال با توجه به اینکه دستور R-Format میباشد خط های کنترلی را چک میکنیم:

Memtoreg:0

از آنجایی که ما به خروجی مموری نیاز نداریم مقدار این بخش برابر 0 است

Memwrite:0

در دستور slt نیازی به نوشتن در مموری نداریم پس 0

Branch:0

Alusrc:0

چون ما در اینجا رجیستر rt را نیاز داریم پس باید خط کنترلی مالتی پلکسر برای ما مقدار رجیستر را در ورودی ALU قرار دهد

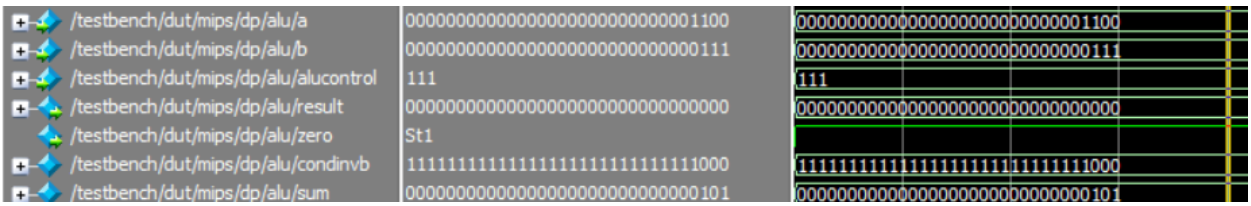
RegDst:1

در دستورات نوع R ذخیره سازی در rd رخ میدهد که IR[15:10] میباشد

RegWrite:1

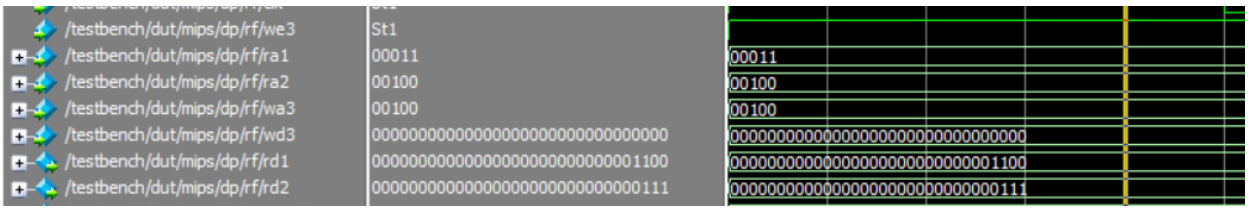
میخواهیم مقدار Aluresult در آدرس rd ذخیره شود پس باید 1 شود.

حال به سراغ ALU میرویم و با توجه به گفته هایمان خروجی ALU 0 یا 1 میتواند باشد



با توجه به بالا حاصل تفریق دو عدد در sum قرار گرفته است و از آنجایی که عدد اول بزرگتر است بنابراین مقدارsum مثبت است که این نتیجه میدهد خروجی ALU صفر باشد

حال به سراغ Reg File میرویم:



که همان چیزی که در ابتدا گفتیم حاصل خروجی ALU در آدرس $rs(0x00100) است قرار میگیرد.

مقدار pc بعد از کلاک:0x00000020

دستور نهم:

دستور نهم به صورت 0x10800001 آمده است.

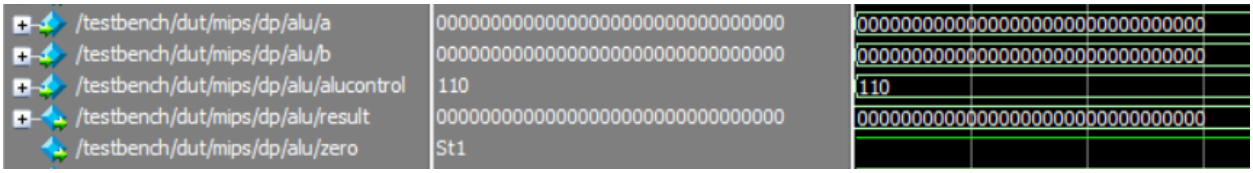
این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

000100	00100	00000	0000000000000001
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور به صورت زیراست:

Beq $rs , $rt , imm-16

توضیحات این دستور در دستور هفتم آمده است و دیگر خطوط کنترلی و ALU آن را چک نمیکنیم و مستقیم به بررسی این میپردازیم که خط کنترلی pcsrc چه مقداری دارد:

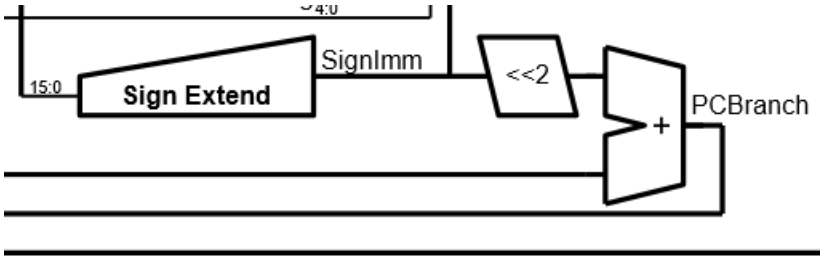


در دستور قبلی مقدار آدرس 0x00100 برابر0 شد و مقدار آدرس 0x00000 نیز همان صفر مانده بود با توجه به کنترل ALU تفریق این دو کاملا صفر میشود و خط zero فعال می گردد.

این رخداد نتیجه میدهد که pcsrc 1 شود و اتفاق زیر بیفتد:

Pc ⟵ pc + signExtend imm-16 * 4

اتفاق بالا در datapath به صورت زیر است:



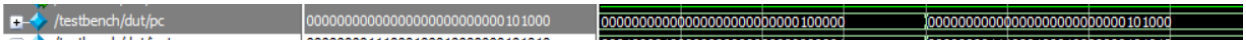
که در کد به صورت زیر است:

```
module adder(input  wire [31:0] a, b,
             output wire [31:0] y);

  assign y = a + b;
endmodule

module sl2(input  wire [31:0] a,
           output wire [31:0] y);

  // shift left by 2
  assign y = {a[29:0], 2'b00};
endmodule

module signext(input  wire [15:0] a,
               output wire [31:0] y);

  assign y = {{16{a[15]}}, a};
endmodule
```

بنابراین مقدار pc در این مرحله به اندازه 8 واحد افزوده میشود

اتفاق بالا باعث میشود که دستورالعمل دهم که به صورت 0x20050000 است اجرا نشود و از روی آن پرش کنیم

مقدار pc بعد از کلاک:0x00000028



دستور دهم: اجرا نمیشود به علت پرش در مرحله قبل

دستور یازدهم:

دستور یازدهم به صورت 0x00E2202A آمده است.

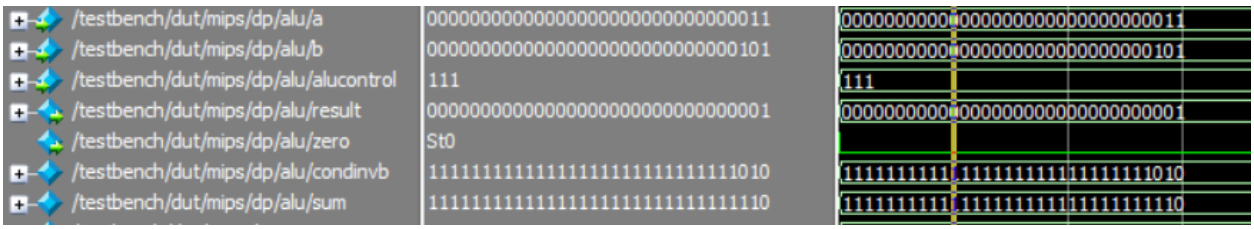
این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00111	00010	00100	00000	101010
op	**rs**	**rt**	**rd**	**shamt**	**func**

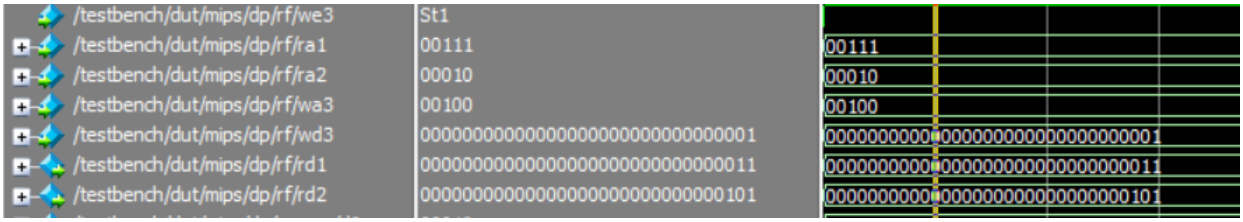
با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

Slt $rd , $rs , $rt

توضیحات این دستور نیز در دستور هستم آمده است.بنابراین خطوط کنترلی و نحوه عملکرد را ارجاع میدهیم به همان بخش و فقط نتایج ALU و Reg File را بررسی میکنیم



با توجه به عملکرد این دستور تفریق عدد اول و دوم در sum قرار دارد و اگر حاصل منفی شود عدد اول کوچکتر است و خروجی ALU برابر 1 میشود که در اینجا نیز حاصل sum منفی است و خروجی 1 شده است.



پس در آدرس $rd(0x00100) مقدار خروجی ALU ذخیره میشود.

مقدار pc پس از کلاک:0x0000002C

دستور دوازدهم:

دستور دوازدهم به صورت 0x00853820 آمده است.

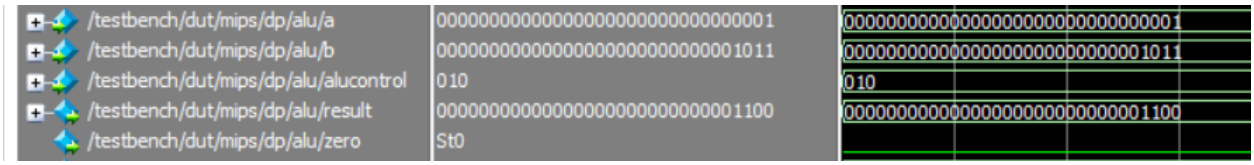
این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00100	00101	00111	00000	100000
op	**rs**	**rt**	**rd**	**shamt**	**func**

با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

Add $rd , $rt , $rs

دستور بالا خطوط کنترلی آن کاملا مشابه دستور ششم میباشد و فقط به خروجی های بلوک های Reg File و ALU میپردازیم:



ALU مقادیر آدرس $rs و $rt را با هم جمع میکند و خروجی به عنوان write data آدرس $rd در Reg File قرار میگیرد.

مقدار pc پس از کلاک:0x00000030

دستور سیزدهم:

دستور سیزدهم به صورت 0x00E23822 آمده است.

این دستوراز نوع R-Format میباشد و آن را به صورت زیر جدا میکنیم.

000000	00111	00010	00111	00000	100010
op	**rs**	**rt**	**rd**	**shamt**	**func**

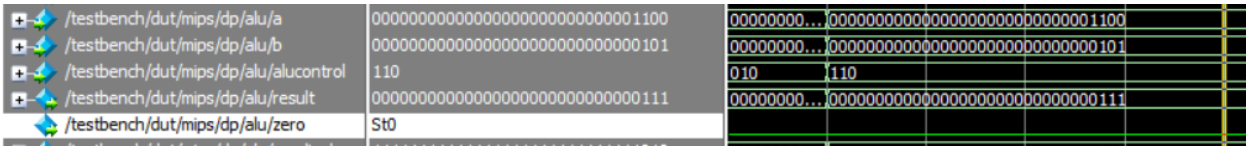
با توجه به مجموعه دستورات MIPS این دستور به شکل زیر است:

Sub $rd , $rs , $rt

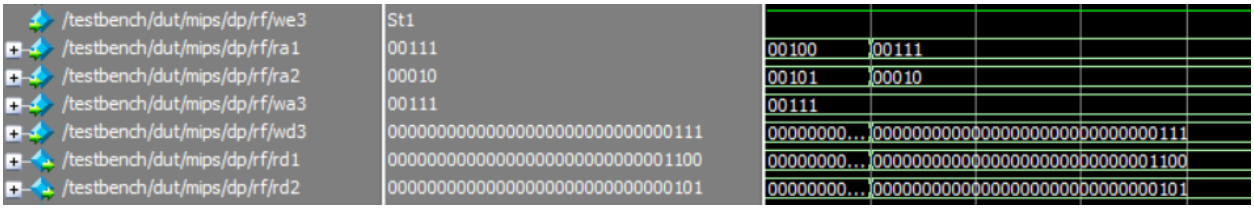
خطوط کنترلی این قسمت مشابه دستور بالا میباشد تنها مقدار ALUcontrol تغییر میکند و برابر با 110 میشود.

در ALU مقادیر $rs و $rt از هم تفریق میشوند و خروجی باید در $rd قرار گیرد.

دستور قبل مقدار موجود در آدرس $rs را تغییر داد و برابر با 0x0000000C شده است و مقدار آدرس $rt نیز داریم بنابراین:



حال به سراغ بررسی Reg File میپردازیم و این نکته نیز واضح است که مقدار regwrite 1 است با توجه به خطوط کنترلی:



پس مقدار 0x00000007 در آدرس 0x00111 پس از لبه بالارونده کلاک نوشته میشود.

مقدار pc پس از کلاک:0x00000034

دستور چهاردهم:

دستور چهاردهم به صورت 0xAC670044 آمده است.

این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

101011	00011	00111	0000000001000100
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور به صورت زیراست:

Sw $rt , imm($rs)

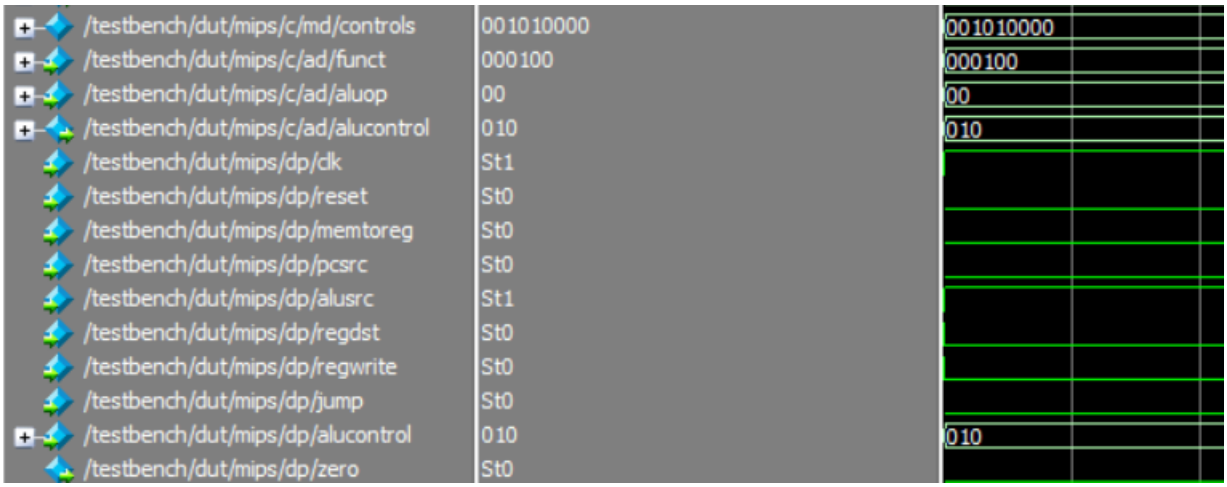
در شبیه سازی ابتدا خط های کنترلی با توجه به فیلد op تعیین میشود که این قسمت از قطعه کد زیر برداشت میشود:

```
  reg [8:0] controls;

  assign {regwrite, regdst, alusrc, branch, memwrite,
          memtoreg, jump, aluop} = controls;

  always @(*)
    case(op)
      6'b000000: controls <= 9'b110000010; // RTYPE
      6'b100011: controls <= 9'b101001000; // LW
      6'b101011: controls <= 9'b001010000; // SW
      6'b000100: controls <= 9'b000100001; // BEQ
      6'b001000: controls <= 9'b101000000; // ADDI
      6'b000010: controls <= 9'b000000100; // J
      default:   controls <= 9'bxxxxxxxxx; // illegal op
    endcase
endmodule
```

تنها بخش از کنترلر که هنوز تعیین نشده است قسمت ALUcontrol میباشد که با توجه به قطعه کد زیر تعیین میگردد:

```
module aludec(input  wire [5:0] funct,
              input  wire [1:0] aluop,
              output reg [2:0] alucontrol);

  always @(*)
    case(aluop)
      2'b00: alucontrol <= 3'b010;  // add (for lw/sw/addi)
      2'b01: alucontrol <= 3'b110;  // sub (for beq)
      default: case(funct)          // R-type instructions
          6'b100000: alucontrol <= 3'b010; // add
          6'b100010: alucontrol <= 3'b110; // sub
          6'b100100: alucontrol <= 3'b000; // and
          6'b100101: alucontrol <= 3'b001; // or
          6'b101010: alucontrol <= 3'b111; // slt
          default:   alucontrol <= 3'bxxx; // ???
        endcase
    endcase
endmodule
```

حال این مقادیر را در شبیه سازی چک میکنیم:



حال با توجه به اینکه دستور I-Format میباشد خط های کنترلی را چک میکنیم:

Memtoreg:0

از آنجایی که ما به خروجی مموری نیاز نداریم مقدار این بخش برابر 0 است

Memwrite:1

در دستور store میخواهیم در بلوک memory داده ذخیره کنیم پس 1

Branch:0

Alusrc:1

چون ما به مقدار ثابت و ساین اکستند شده نیاز داریم پس این خط کنترلی 1 است

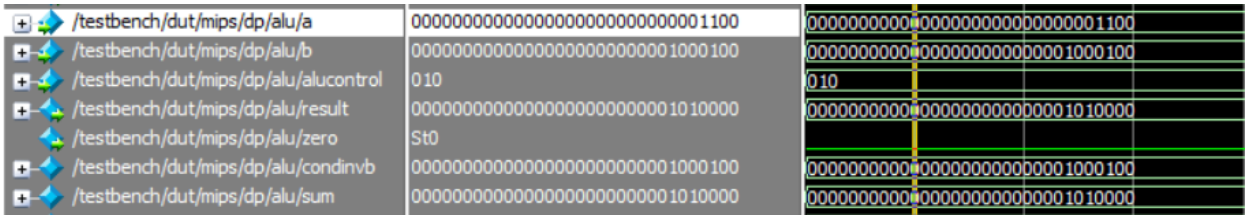
RegDst:0

در اینجا ما چیزی در بلوک رجیستر ها نمینویسیم بنابراین اهمیتی ندارد چه باشد

RegWrite:0

در Reg File چیزی نمینویسیم

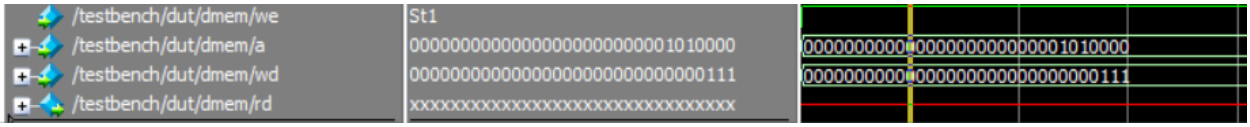
حال به بررسی بلوک ALU میپردازیم:



دراین دستور خروجی ALU به عنوان dataadr مموری استفاده خواهد شد و از آن جایی که در دستور sw آدرس برابر است با:

$rs + SE[imm-16]

بنابراین آدرس ذخیره سازی را یافتیم حال به سراغ بلوک مموری



مقدار write data از rd2 رجیستر فایل ناشی میشود و a نیز همان dataadr است که در ALU محاسبه گردید حال با لبه بالارونده کلاک مقدار در مموری نوشته میشود

مقدار pc پس از کلاک:0x00000038

دستور پانزدهم:

دستور پانزدهم به صورت 0x8C020050 آمده است.

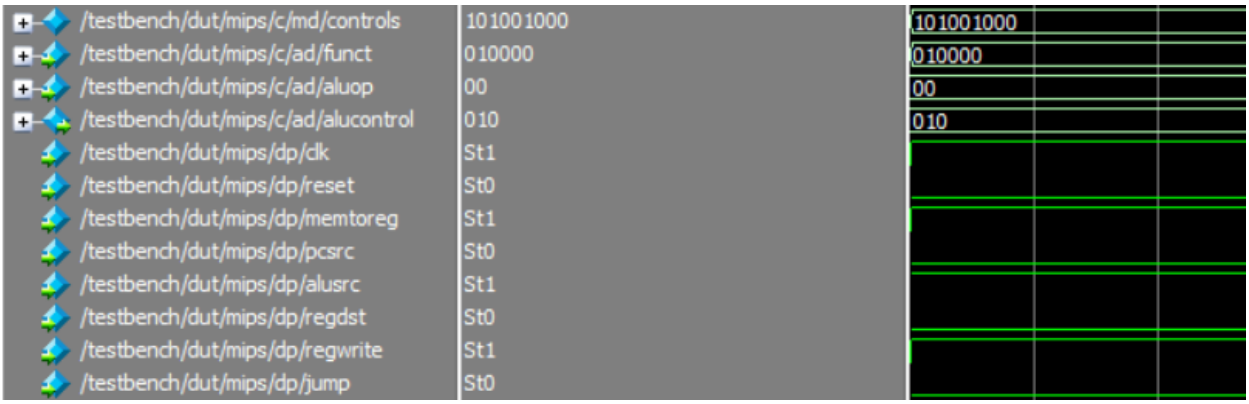
این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

100011	00000	00010	0000000001010000
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور به صورت زیراست:

Lw $rt ,$rs(imm-16)

به سراغ مقادیر خطوط کنترلی می رویم:



حال با توجه به اینکه دستور I-Format میباشد خط های کنترلی را چک میکنیم:

Memtoreg:1

از آنجایی که ما به خروجی مموری نیاز داریم مقدار این بخش برابر 1 است

Memwrite:0

در دستور load میخواهیم در بلوک memory چیزی ذیره نشود پس 0

Branch:0

Alusrc:0

ما هیچ کاری با ALU نداریم و برای ما اهمیتی ندارد که ALUsrc چه مقداری داشته باشد

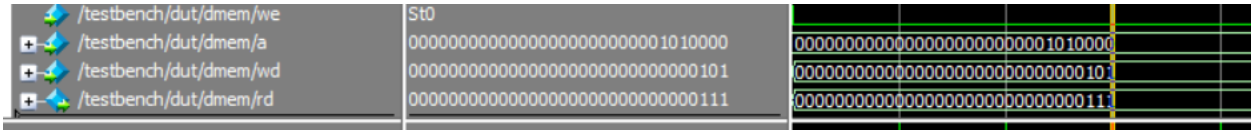
RegDst:0

در اینجا ما چیزی در بلوک رجیستر ها نمینویسیم بنابراین اهمیتی ندارد چه باشد

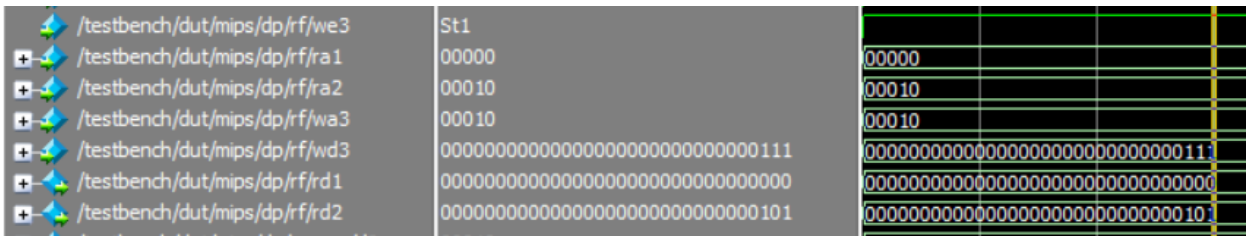
RegWrite:1

در Reg File، read data بدست آمده از مموری را میخواهیم بنویسیم

حال به سراغ بلوک مموری میرویم:



حال read data پیدا شد و این مقدار به عنوان write data رجیستر فایل قرار است استفاده گردد بنابراین به سراغ بلوک Reg File میرویم:



طبق آن چیزی که انتظار میرفت شبیه سازی رخ داده است و پس از کلاک مقدار read data در رجیستر با آدرس 0x00010 نوشته میشود

مقدار pc پس از کلاک:0x0000003C

دستور شانزدهم:

دستور شانزدهم به صورت 0x08000011 آمده است.

این دستوراز نوع J-Format میباشد و آن را به صورت زیر جدا میکنیم:

000010	00000000000000000000010001
op	**Imm-26**

دستور بالا با توجه به MIPS به صورت زیر است:

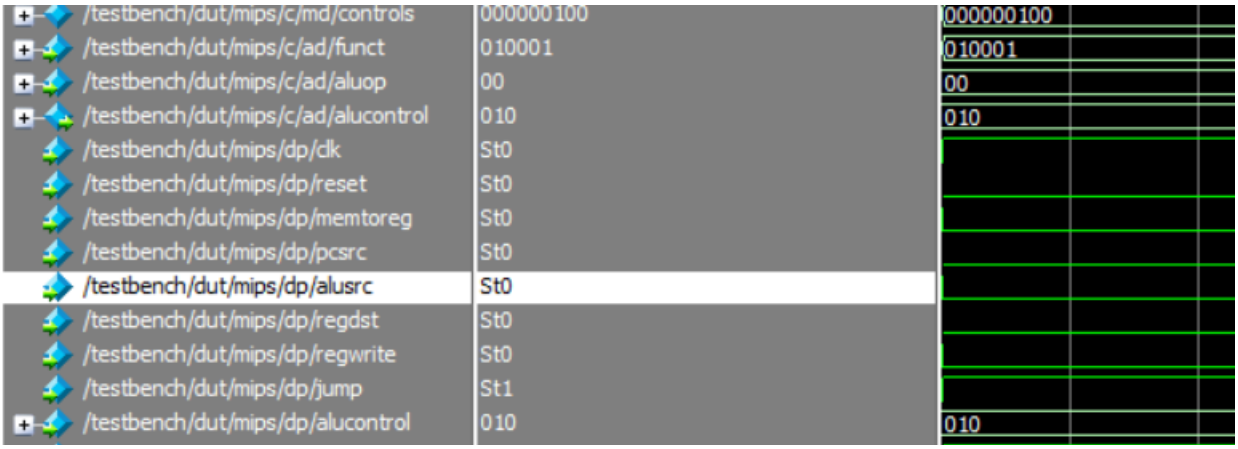
J imm-26 # PC⟵ {PC[31:28],imm-26,2’b00}

حال به سراغ کد میرویم تا خطوط کنترلی را بیابیم:

```
  reg [8:0] controls;

  assign {regwrite, regdst, alusrc, branch, memwrite,
          memtoreg, jump, aluop} = controls;

  always @(*)
    case(op)
      6'b000000: controls <= 9'b110000010; // RTYPE
      6'b100011: controls <= 9'b101001000; // LW
      6'b101011: controls <= 9'b001010000; // SW
      6'b000100: controls <= 9'b000100001; // BEQ
      6'b001000: controls <= 9'b101000000; // ADDI
      6'b000010: controls <= 9'b000000100; // J
      default:   controls <= 9'bxxxxxxxxx; // illegal op
    endcase
endmodule
```

با توجه به اینکه دستور J-format میباشد تمام خطوط کنترلی به غیر از jump برابر 0 هستند.

در شبیه سازی این موضوع واضح است:



حال به بررسی قطعه کدی که مربوط به jump میشود میپردازیم:

```
// next PC logic
flopr #(32) pcreg(clk, reset, pcnext, pc);
adder       pcadd1(pc, 32'b100, pcplus4);
sl2         immsh(signimm, signimmsh);
adder       pcadd2(pcplus4, signimmsh, pcbranch);
mux2 #(32)  pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
mux2 #(32)  pcmux(pcnextbr, {pcplus4[31:28],
                  instr[25:0], 2'b00}, jump, pcnext);
```

با توجه به قسمت هایلایت شده اگر jump 1 باشد مالتی پلکسر مقدار درست پرش را تحت عنوان pcnext ذخیره میکند که این مقدار را در شبیه سازی میبینیم:

از طرفی میدانیم مقدار فعلی pc:0x0000003C است و مقدار بعدی آن در لبه بالا رونده کلاک 0x000044 خواهد بود که 8 واحد با فعلی اختلاف دارد پس از دستور هفدهم میپرد و به سراغ دستور هجدهم میرود

مقدار pc پس از کلاک:0x00000044

دستور هفدهم:با توجه به دستور قبل از روی آن پریدیم!

دستور هجدهم:

دستور هجدهم به صورت 0xAC020054 آمده است.

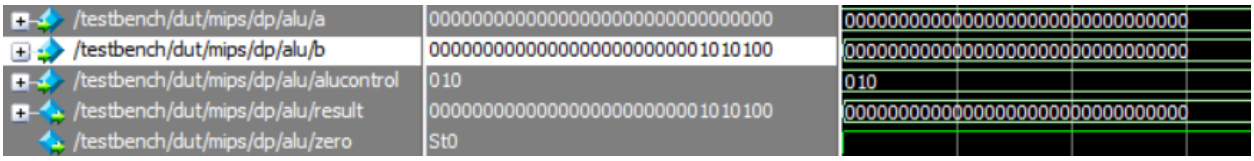
این دستوراز نوع I-Format میباشد و آن را به صورت زیر جدا میکنیم.

101011	00000	00010	0000000001010100
op	**rs**	**rt**	**Imm-16**

با استفاده از مجموعه دستورات متوجه میشویم این دستور به صورت زیراست:

Sw $rt , $rs(imm-16)

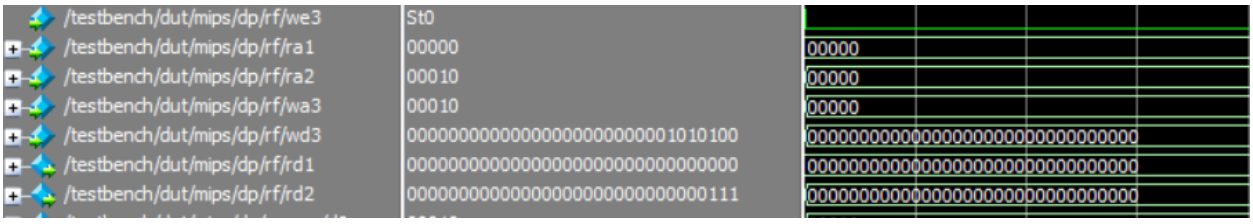
این دستور مشابه دستور چهاردهم است پس خطوط کنترلی و ALU آن را ذکر نمیکنیم و به سراغ بلوک ALU میرویم:



دراین دستور خروجی ALU به عنوان dataadr مموری استفاده خواهد شد و از آن جایی که در دستور sw آدرس برابر است با:

$rs + SE[imm-16]

حال به سراغ write data که خروجی رجیستر فایل میباشد میرویم:



Rd2 در واقع write data ما میباشد

پس : $dataadr=2^6+2^4+2^2=84$ write data = 7

پس شروط مدنظر تست بنچ درست است و کد به صورت کاملا صحیح اجرا شد.

پردازنده ساخته شده کاملا صحیح رفتار میکند و نتیجه مطلوب را دریافت کردیم.