۱. توضیح ماژول ها

module ALU(z,c,a,b,alusel);

این ماژول محساباتی ما هست، a و b دو ورودی ۳۲ بیت آن هستند و c خروجی ۳۲ بیت آن.

اگر c برابر صفر بشود ، پرچم z به عنوان خروجی یک خواهد شد.

نوع عملیات محاسباتی بر اساس ورودی ۳ بیت alusel تعیین خواهد شد :

000 : جمع - 001 : تفاضل – 010 : اندمنطقی – 011 : اورمنطقی – 100 : slt(اگر a کوچکتراز b باشد c یک میشود)

---

module REG32(parout,parin,ld,clk,rst);

این ماژول یک رجیستر ۳۲ بیتی می باشد،

در صورت فعال بودن rst خروجی parout صفرخواهد شد.

اگر rst فعال نباشد ، و ld یک باشد ، آنگاه در هر لبه مثبت کلاک ، ۳۲ بیت parin در ۳۲ بیت parout قرار داده می شود.

از این ماژول در ساخت رجیسترهای PC , IR , MDR , A , B , ALUOUT استفاده شده است. در رجیسترهای A,B,ALUOUT,MDR ورودیهای rst و ld همواره 0 و 1 قرار داده شده اند. هم چنین در رجیستر IR ورودی rst همواره 0 خواهد بود.

---

module MUX(c,a,b,sel);

این ماژول مالتیپلکسر ۳۲ بیت دو به یک است. دو ورودی ۳۲ بیت a و b را می گیرد اگر مقدار sel صفر باشد ، a را روی خروجی c قرار خواهد داد و اگر sel یک باشد ، b را روی خروجی c قرارمی دهد.

علاوه بر این مالتیپلکسر۳۲ بیتی ، از مالتیپلکسرهای ۵ بیت و ۱ بیت هم در برنامه استفاده شده است.

---

module MUX4_1(e,a,b,c,d,sel);

این ماژول مالتیپلکسر ۳۲ بیت چهار به یک است. چهار ورودی ۳۲ بیت a,b,c,d را می گیرد اگر مقدار sel به ترتیب صفر ، یک ، دو و سه باشد ، به ترتیب a ، b ، c ، d را روی خروجی e قرار می دهد.

---

module MUX3_1(d,a,b,c,sel);

این ماژول مالتیپلکسر ۳۲ بیت سه به یک است. سه ورودی ۳۲ بیت a,b,c را می گیرد اگر مقدار sel به ترتیب صفر ، یک ، دو باشد ، به ترتیب a ، b ، c را روی خروجی d قرار می دهد.

---

module MEM_FILE(memory,readdata,addr,writedata,memread,memwrite,clk);

این ماژول حافظه را نگه داری می کند. به دلیل محدودیت حافظه مدلسیم ، ما حافظه را ۱۶ کیلوبایت معادل ۶۵۵۳۶ کلمه تعریف کرده ایم یعنی :

```
output logic [31:0]memory[0:65536];
```

ما داده ها را از فایل memory.txt که در محل پوشه پروژه قرار دارد فراخوانی کرده و در حافظه وارد می کنیم :

```
initial begin

    $readmemb("memory.txt",memory);

End
```

ورودی ۳۲ بیت addr ، آدرس محل حافظه را مشخص می کند ، سپس در صورت فعال بودن سیگنال کنترلی memread ، محتوای آن آدرس در خروجی ۳۲ بیت readdata نمایش داده خواهد شد.

برای نوشتن در حافظه به صورت سنکرون عمل میکنیم ، به این صورت که اگر سیگنال ورودی memwrite فعال باشد ، با اولین لبه مثبت کلاک ، مقدار ورودی ۳۲ بیت writedata در خانه حافظه به آدرس addr نوشته خواهد شد.

ما برای نشان دادن مقادیر حافظه در تست بنچ ، کل حافظه memory را از ماژول خروجی گرفتیم.

---

module REG_FILE(register,readdata1,readdata2,readreg1,readreg2,writereg,writedata,regwrite,clk);

این ماژول دربردارنده ی رجیستر فایل می باشد. متغیر **register** یک آرایه ۳۲ در ۳۲ بیت است ، که کلمه نخست آن یا همان R0 همواره صفر می باشد. یعنی :

```
initial begin

        register[0]=32'b0;

 end
```

می توان همزمان دو رجیستر را خواند ، برای خواندن دو ورودی ۵ بیت readreg1 و readreg2 آدرس محل رجیسترها را مشخص کنند، سپس مقادیر آنها در دو خروجی ۳۲ بیت readdata1 و readdata2 نمایش داده خواهد شد.

عمل نوشتن در رجیسترفایل به صورت سنکرون می باشد ، به این صورت که در صورت فعال بودن سیگنال کنترلی regwrite ، با اولین لبه مثبت کلاک ، مقدار ورودی ۳۲ بیت writedata در آدرس ورودی writereg نوشته خواهد شد.

ما برای تست کردن و دیدن محتویات رجیستر فایل ، حافظه register که دربردارنده تمام رجیسترفایل می باشد را از ماژول خروجی گرفته ایم.

---

module ALU_CONTROL(alusel,func,aluop);

این ماژول کنترل کننده عملیات واحد محاسباتی پردازنده می باشد.

خروجی ۳ بیت alusel به ALU وارد خواهد شد و نوع دستور محاسباتی را برای ALU مشخص خواهد کرد بدین صورت که اگر 000 : جمع – 001 : تفاضل – 010 : اند – 011 : اور – 100 : slt خواهد بود.

ورودی ۲ بیت aluop یک سیگنال کنترلی است که از سمت کنترلر اصلی می آید اگر :

00 : alusel را 000 خواهدکرد. این حالت برای دستور addi استفاده می شود.(حالت default)

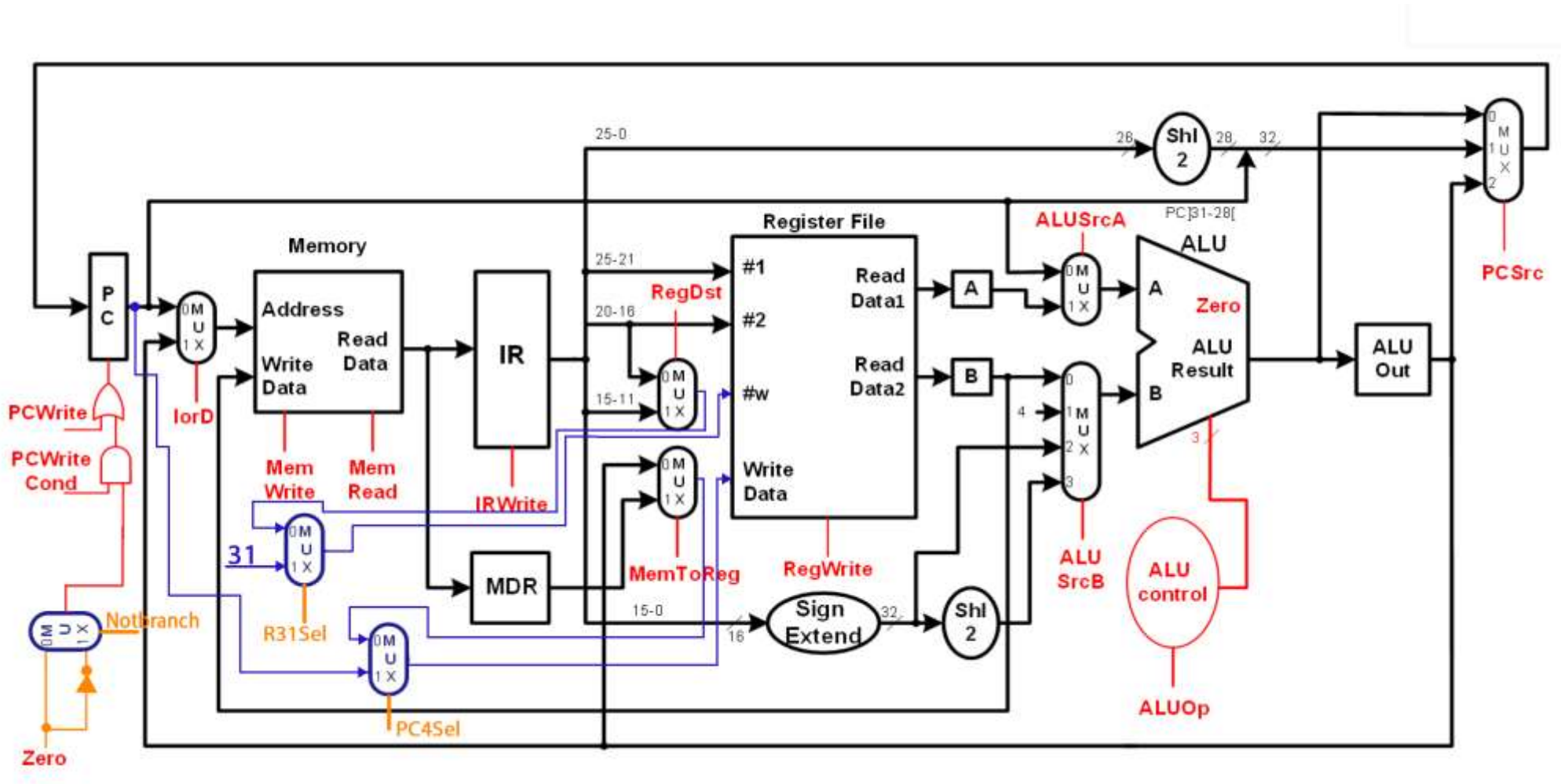
01: نوع دستورما R-type است پس به مقدار 6 بیت ورودی func که درواقع ۶ بیت کم ارزش IR می باشد نگاه می کند. باتوجه به آن جمع یا تفاضل یا ... را مشخص می کند.

10: alusel را 001 خواهد کرد. این حالت برای دستورات beq و bne استفاده می شود.

11: alusel را 010 خواهد کرد. این حالت برای دستور andi استفاده می شود.

---

۲. مسیر داده



دیتاپث ما همان دیتاپث اصلی MIPS است که دکتر تدریس کرده اند ، سیم های آبی و نارنجی(کنترلی) و mux ها جدید نشان دهنده قسمت های جدیدی هستند که به مسیر داده اضافه شده اند.

توضیح مسیر داده دستورات جدید اضافه شده :

bne : برای این دستور که عکس دستور beq عمل می کند کافیست از سیگنال Zero که از ALU خارج می شود یک قرینه بگیریم تا درست اجرا شود ، برای تشخیص اینکه چه زمانی باید Zero یا قرینه اش را بفرستیم ، یک MUX سر راهشان قرار دادیم. این MUX توسط سیگنال کنترلی NotBranch کنترل خواهد شد، بدین صورت که اگر دستور ما bne باشد ، NotBranch برابر یک خواهد شد و قرینه Zero را عبور می دهد. تعداد کلاک ها برابر کلاکهای beq است.

andi ,addi : در این دستورات به مسیر داده جدیدی برای عبور داده ها به alu و انجام عملیات و ریختن در رجیستر مقصد نیاز نداریم ، اما نمی توان نوع عملیات محاسباتی را برای ALU مشخص کرد ، به همین دلیل ما سیگنال کنترلی یک بیت ALUOp را که از قبل داشتیم تبدیل به دو بیت کردیم ، تا بتوانیم با این دوبیت دستور جمع برای addi و اند برای andi را مستقیما به ALUCONTROl بدهیم تا ALUCONTROl به ALU دستورات را برساند. هم چنین باتوجه به فرمت دستور ، برخلاف دستورات RT ، باید RegDst صفر باشد. در مجموع برای اجرای این دستورات به ۴ کلاک نیاز داریم.

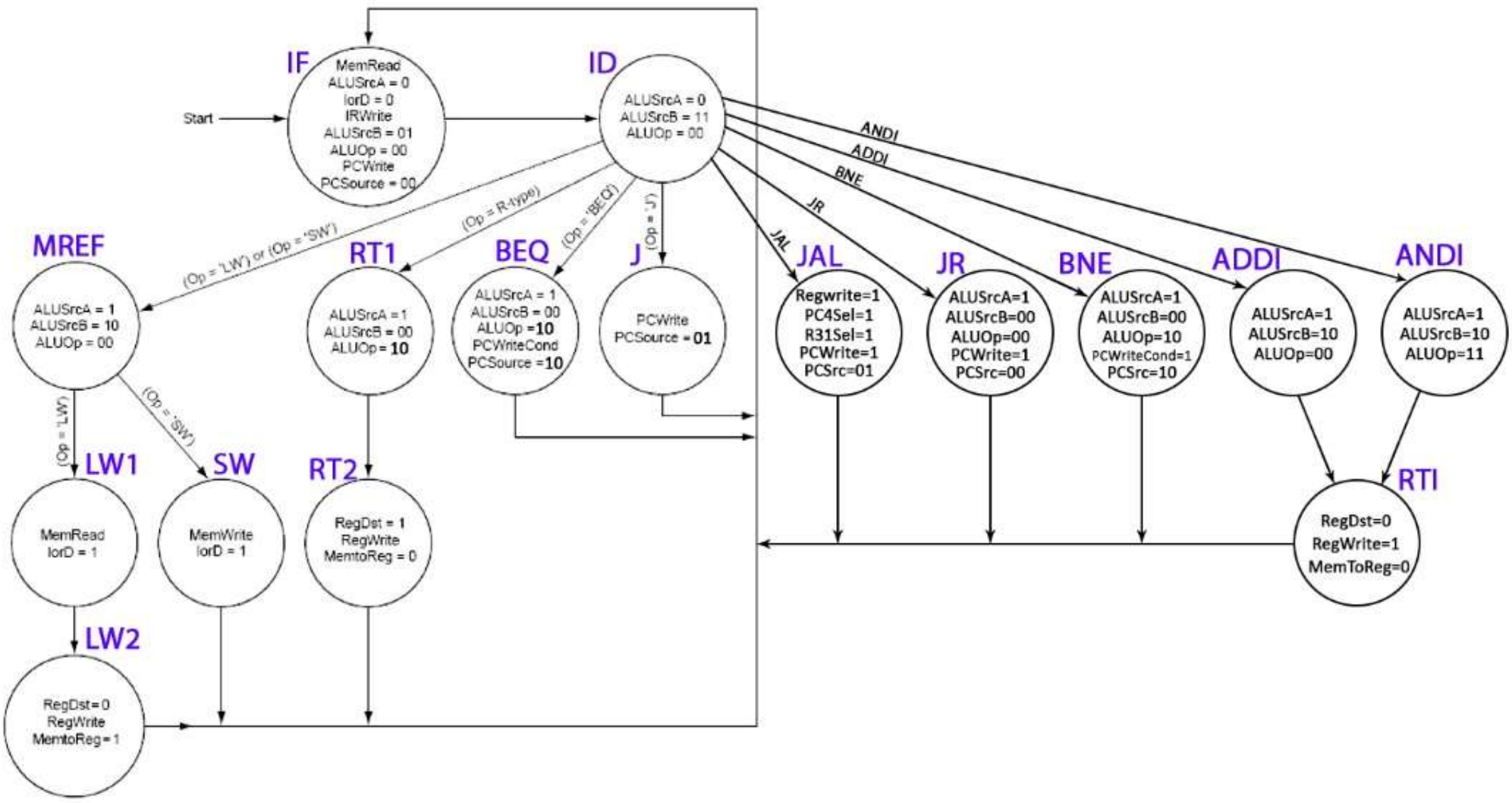
jal : این دستور همانند دستور j است با این تفاوت که بایستی آدرس آخرین دستور قبل از پرش را در رجیستر ۳۱ ذخیره کند. برای اینکار ابتدا باید عدد ۳۱ را به WriteReg بدهیم ، بنابراین قبل از ورودی WritReg یک MUX قرار می دهیم که توسط

سیگنال کنترلی R31Sel کنترل می شود ، اگر یک باشد عدد ۳۱ وارد آن می شود. در ادامه کار بایستی مقدار PC+4 را در رجیستر ۳۱ ذخیره کنیم پس قبل از ورودی WriteData یک MUX قرار می دهیم که اگر سیگنال کنترلی PC4Sel یک باشد ،آنگاه مقدار PC+4 را وارد WriteData خواهد کرد. تعداد کلاک ها برابر کلاکهای j است.

jr : برای اجرای این دستور به مسیر داده جدیدی نیاز نداریم ، آدرس رجیستر از طریق #۱ وارد می شود ، رجیستر A حاوی آدرس پرش و رجیستر B صفر است. پس از جمع A و B در ALU ، خروجی از طریق PCSrc=0 وارد رجیستر pc خواهد شد. مجموع تعداد کلاکها ، ۳ کلاک می باشد.

---

۳. کنترلر و سیگنال های کنترلی :



OPC	ALUOp	func	ALU Action	ALUOperation
RT	01	100000	add	000
	01	100010	sub	001
	01	100100	and	010
	01	100101	or	011
	01	101010	slt	100
lw	00	XXXXXX	add	000
sw	00	XXXXXX	add	000
beq	10	XXXXXX	sub	001
bne	10	XXXXXX	sub	001
addi	00	XXXXXX	add	000
andi	11	XXXXXX	and	010
Jr	00	XXXXXX	add	000

۴. طراحی برنامه های تست و سیمولیشن آنها :

در ابتدا یک فایل memory.txt در پوشه پروژه ایجاد کردیم. سپس ۵۰۱ خط عدد ۳۲ بیتی(معادل ۲۰۰۴ آدرس) در آن نوشتیم ، مقادیر آدرس های ۱۰۰۰ تا ۱۰۷۶ این فایل حافظه به ترتیب دسیمال برابر است با :

مقدار	آدرس	خانه
1	1000	250
2	1004	251
3	1008	252
4	1012	253
5	1016	254
6	1020	255
7	1024	256
8	1028	257
9	1032	258
10	1036	259
32	1040	260
16	1044	261
8	1048	262
4	1052	263
24	1056	264
8	1060	265
256	1064	266
64	1068	267
32	1072	268
1	1076	269

توجه داشته باشید داده های هردو برنامه همین مقادیر هستند و همچنین این دو برنامه در یک سیمولیشن اجرا شده یعنی ابتدا دستورات برنامه اول ، سپس دستورات برنامه دوم اجرا خواهد شد.

برنامه اول (مجموع اعضای یک آرایه ۱۰ بیتی از ۱۰۰۰ تا ۱۰۳۶) :

ما مقادیر آدرس های ۱۰۰۰ تا ۱۰۳۶(معادل خانه ۲۵۰ تا ۲۵۹) را میخوانیم و حاصل جمع را در آدرس ۲۰۰۰ (معادل خانه ۵۰۰) خواهیم ریخت.

```
add R1, R0, R0
addi R2, R0, 40
add R4, R0, R0
LOOP: slt R3, R1, R2
beq R3, R0, END-LOOP
lw R5, 1000(R1)
add R4, R4, R5
addi R1, R1, 4
J LOOP
END-LOOP: sw R4, 2000(R0)
```

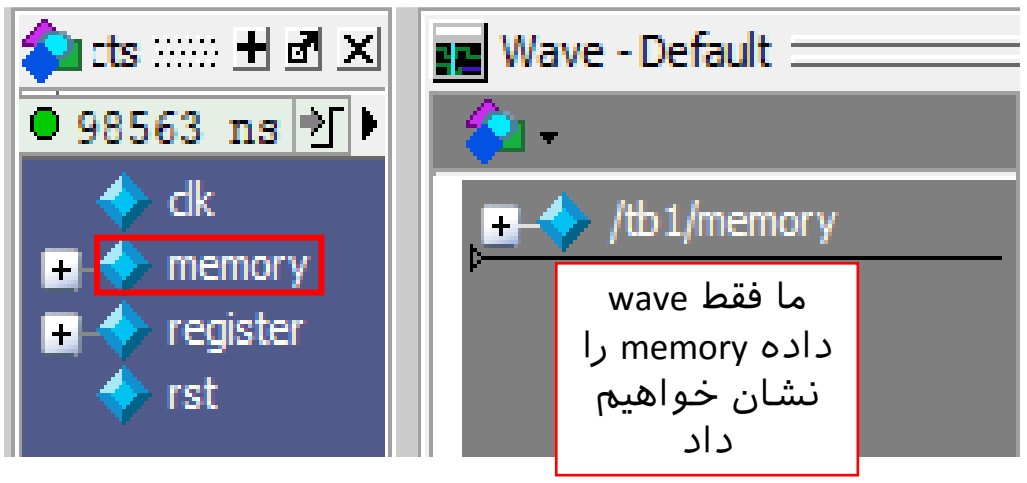
دستورات فوق در آدرس های 0 تا 36 حافظه در فایل memory.txt نوشته شده اند.

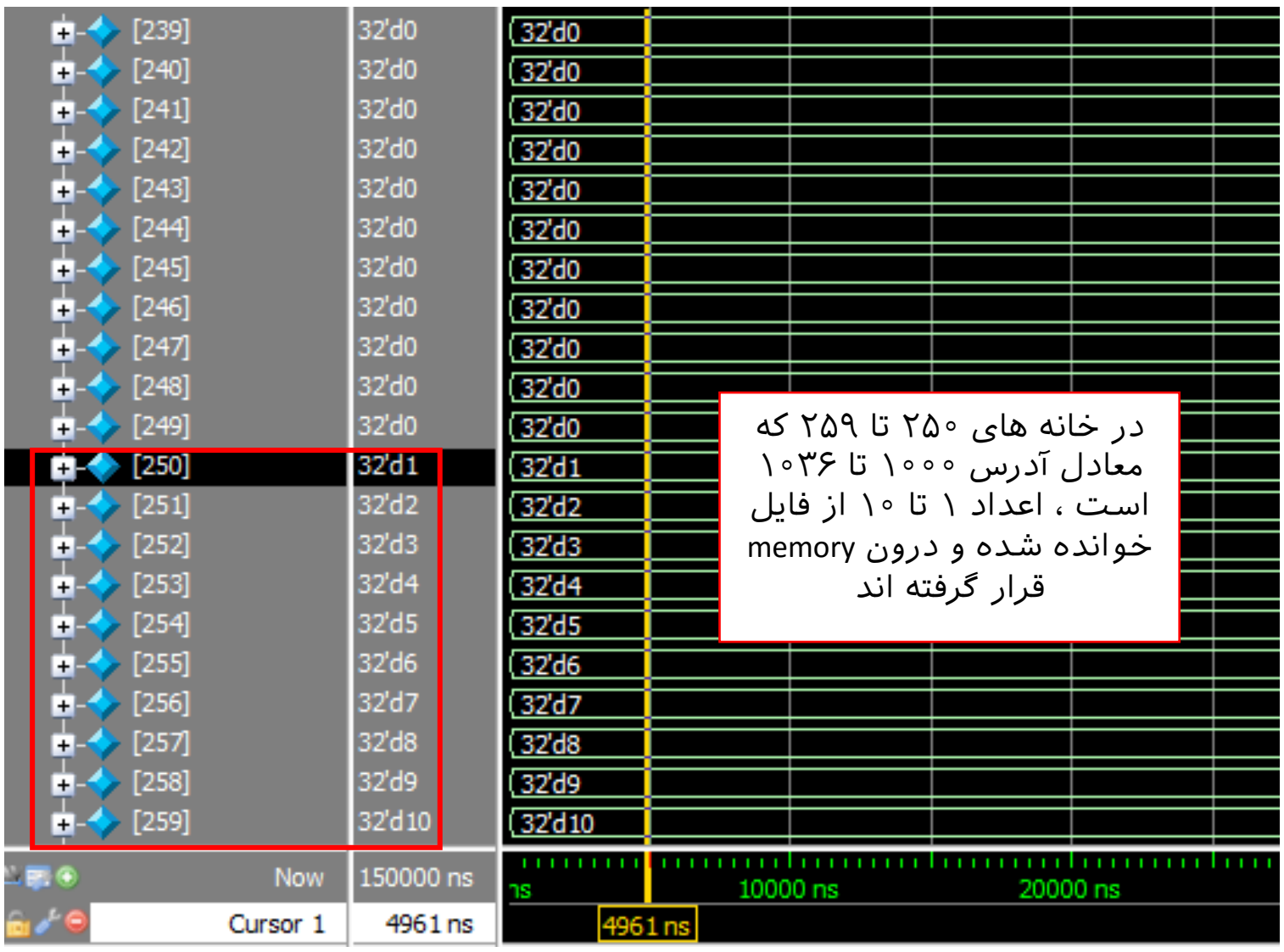
در فایل Testbenches.sv یک ماژول تست بنچ به نام tb1() وجود دارد که ماژول زیر را سیمولیت می کند :

```
datapath uut0(register,clk,rst, memory);
```

register خروجی تمام حافظه regfile می باشد ،

memory خروجی تمام حافظه Mem-File می باشد ما در سیمولیشن فقط به مقادیر Mem-File نگاه می کنیم.


98563 ns
clk
memory
register
rst
Wave - Default
/tb1/memory
ما فقط wave داده memory را نشان خواهیم داد


[239] 32'd0
[240] 32'd0
[241] 32'd0
[242] 32'd0
[243] 32'd0
[244] 32'd0
[245] 32'd0
[246] 32'd0
[247] 32'd0
[248] 32'd0
[249] 32'd0
[250] 32'd1
[251] 32'd2
[252] 32'd3
[253] 32'd4
[254] 32'd5
[255] 32'd6
[256] 32'd7
[257] 32'd8
[258] 32'd9
[259] 32'd10
Now 150000 ns
Cursor 1 4961 ns
10000 ns
20000 ns
4961 ns
در خانه های ۲۵۰ تا ۲۵۹ که معادل آدرس ۱۰۰۰ تا ۱۰۳۶ است ، اعداد ۱ تا ۱۰ از فایل خوانده شده و درون memory قرار گرفته اند


[491] 32'd0
[492] 32'd0
[493] 32'd0
[494] 32'd0
[495] 32'd0
[496] 32'd0
[497] 32'd0
[498] 32'd0
[499] 32'd0
[500] 32'd55
[501] 32'd0
[502] 32'dx
[503] 32'dx
[504] 32'dx
[505] 32'dx
[506] 32'dx
[507] 32'dx
[508] 32'dx
[509] 32'dx
[510] 32'dx
[511] 32'dx
Now 150000 ns
Cursor 1 25350 ns
25000 ns
30000 ns
25350 ns
در زمان ۲۵۳۵۰ نانوثانیه ، مجموع مقادیر آدرسهای ۱۰۰۰ تا ۱۰۳۶ که همان اعداد ۱ تا ۱۰ بودند ، در خانه شماره ۵۰۰ که معادل آدرس ۲۰۰۰ حافظه است ، ریخته شد.
همانطور که می بینید این مقدار ۵۵ است که همان مجموع اعداد ۱ تا ۱۰ می باشد.
پس برنامه درست کار کرد☺

برنامه دوم (پیداکردن بزرگترین عنصر یک آرایه ۲۰ بیتی از ۱۰۰۰ تا ۱۰۷۶) :

ما مقادیر آدرس های ۱۰۰۰ تا ۱۰۷۶(معادل خانه ۲۵۰ تا ۲۶۹) را میخوانیم و مقدار بزرگترین عنصر و اندیس آنرا به ترتیب در آدرسهای ۲۰۰۰ و ۲۰۰۴ (معادل خانه ۵۰۰ و ۵۰۱) حافظه میریزیم.

```
add R1, R0, R0
addi R2, R0, 80
add R8, R0, R0
lw R4, 1000(R1)
LOOP: slt R3, R1, R2
beq R3, R0, END-LOOP
lw R5, 1000(R1)
slt R6, R4, R5
beq R6, R0, CONTINUE
lw R4, 1000(R1)
add R7, R8, R0
CONTINUE: addi R8, R8, 1
addi R1, R1, 4
J LOOP
END-LOOP: sw R4, 2000(R0)
sw R7, 2004(R0)
```

دستورات فوق در آدرس های 40 تا 100 حافظه در فایل memory.txt نوشته شده اند.

در خانه های ۲۵۰ تا ۲۶۹ که معادل آدرس ۱۰۰۰ تا ۱۰۷۶ است ، اعدادی که مشاهده می کنید ، از فایل خوانده شده و درون memory قرار گرفته اند.

بزرگترین عدد ۲۵۶ است

اندیس آن در این آرایه ۲۰ تایی شماره ۱۶ می باشد

در زمان ۹۸۰۵۰ نانوثانیه ، خانه شماره ۵۰۰ که همان آدرس ۲۰۰۰ می باشد برابر ۲۵۶ می شود.

همچنین در زمان ۹۸۴۵۰ نانوثانیه ، خانه ۵۰۱ که همان آدرس ۲۰۰۴ است ، برابر ۱۶ می شود.

همانطور که در عکس قبل مشاهده کردید ۲۵۶ بزرگترین عدد است و ۱۶ اندیس آن است!

پس برنامه درست کار کرد☺