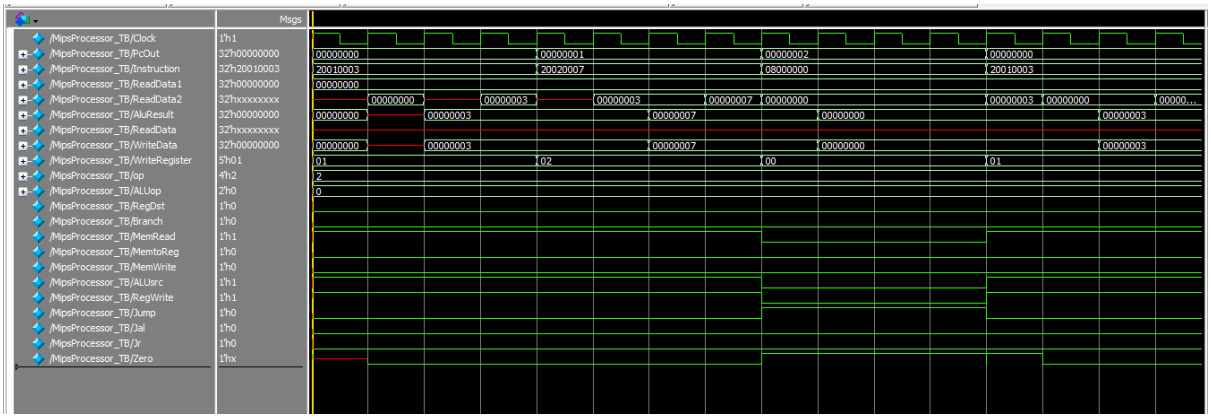
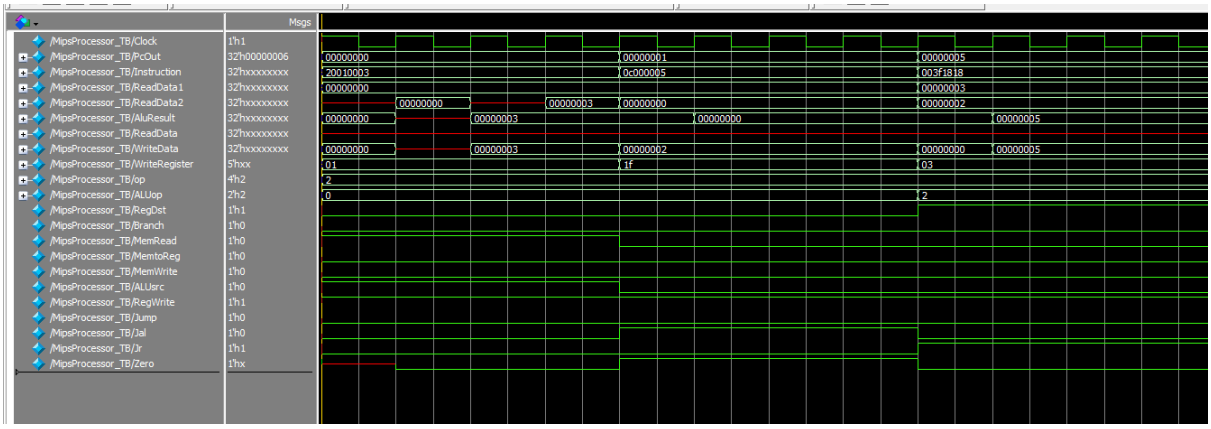



```
memory[0] = 32'b001000_00000_00001_0000000000000011;     // addi, R1, R0, 3

memory[1] = 32'b001000_00000_00010_0000000000000111;     // addi R2 , R0, 7

memory[2] = 32'b000010_00000000000000000000000000;       // j 0
```

در شکل بالا مشخص است که پس از اجرای دستورات 0 و 1 در دستور شماره 2 قصد داریم به ابتدای برنامه جامپ کنیم همانطور ک معلوم است این جامپ به درستی انجام شده روش جامپ به این نحو است که 6 بیت پر ارزش رجیستر pc + 1 با 26 بیت اول دستور concat شده و این حاصل وارد یک ماکس میشود که در صورتی که select آن یک باشد این آدرس در گام بعدی وارد pc می شود. Select را کافی است کنترل jump انتخاب کنیم البته در قسمت بعد برای jal هم این حاصل را نیاز داریم به همین دلیل select در نهایت jump or jal خواهد شد. ورودی دیگر ماکس نیز همان خروجی ماکس موجود در سخت افزار اولیه بود ک حاصل آن وارد رجیستر pc می شد.



memory[0] = 32'b001000_00000_00001_0000000000000011; // addi R1, R0, 3//

memory[1] = 32'b000011_00000000000000000000000101; // jal 5//

memory[5] = 32'b000000_00001_11111_00011_00000_011000; // addi R3, R1, R31//

در شکل بالا مشخص است که ابتدا عدد 3 داخل رجیستر 1 ریخته شده سپس با فراخوانی jal آدرس pc + 1 یعنی 2 باید در رجیستر شماره 31 ذخیره شود و سپس به آدرس پرش یعنی 5 پرش کنیم. در گام بعد دستور شماره fetch 5 می شود که نشان می دهد پرش صحیح بوده و سپس مقادیر رجیستر های1 و 31 را میخواند که همانطور که مشخص است read data1 , read dat2 مقادیر 3 و 2 را اتخاذ کرده اند که نشان می دهد هم دستور اول به درستی اجرا شده و هم دستور دوم علاوه بر پرش آدرس بازگشت را درستی ذخیره کرده است.

memory[0] = 32'b001000_00000_00001_0000000000000011 // addi R1, R0, 3

memory[1] = 32'b001000_00000_00010_0000000000000011 // addi R2, R0, 3

memory[2] = 32'b000000_00001_00010_00011_00000_011000 // add R3, R1, R2

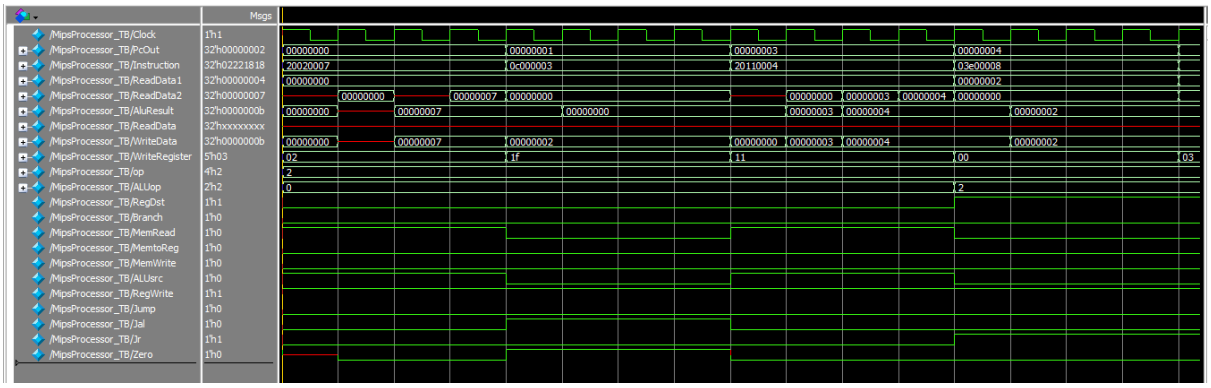
memory[3] = 32'b101011_00001_00010_0000000000001010 // sw R2, 10(R1)

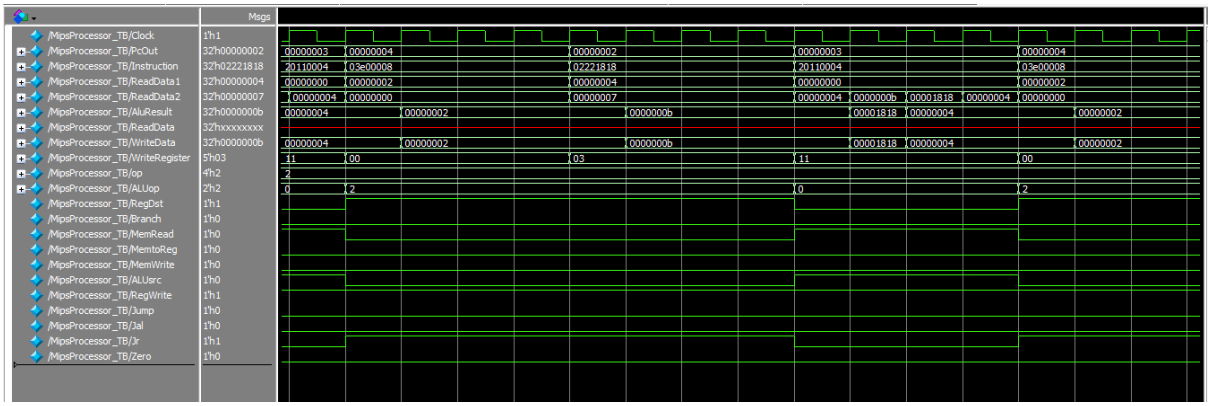
memory[4] = 32'b100011_00010_00001_0000000000001010 // lw R1, 10(R2)

memory[5] = 32'b000100_00001_00010_0000000000010100 // beq R1, R2, 20

در شکل بالا مشخص است که ابتدا مقدار 3 در رجیستر 1 ذخیره شده سپس 3 در رجیستر 2 ذخیره شده و در دستورشماره 2 حاصل دو رجیستر خوانده می شود و با هم جمع زده می شود در گام بعد حاصل alu وارد write data بانک رجیستر شده و مقدار6 در رجیستر 3 ذخیره می گردد.

در گام بعدی دستور شماره 3 اجرا شده و این دستور برای ذخیره کردن مقدار رجیستر 2 در خانه شماره 10 + 3 حافظه است در دستور بعد قصد داریم همین مقدار را از حافظه بخوانیم و روی رجیستر 1 ذخیره کنیم و با توجه به اینکه خروجی read_data مقدار 3 شده یعنی دستور sw به درستی اجرا شده بود سپس رجیستر های 1 و 2 خوانده شده و با هم مقایسه می شوند چون دستور beq است کنترل branch فعال است و zero ی خروجی alu نیز فعال می شود پس در گام بعد به دستور 20 + 6 یعنی 26 پرش می کنیم که در این خانه از دستورات دستوری وجود ندارد. به طور کلی پرش beq از نوع نسبی و پرش های jal و jump از نوع مطلق است.





memory[0] = 32'b001000_00000_00010_0000000000000111;//// Addi R2, R0 , 7

memory[1] = 32'b000011_00000000000000000000000011;//jal 3//

memory[2] = 32'b000000_10001_00010_00011_00000_011000;//add R3, R17, R2//

memory[3] = 32'b001000_00000_10001_0000000000000100;//addi R17, 4//

memory[4] = 32'b000000_11111_00000_00000_00000_001000;//jr R31//

همانطور که درشکل مشخص است ابتدا عدد 7 در رجیستر 2 ذخیره شده سپس در دستور شماره 1 jal 3 موجود است که در طی اجرا ان باید به دستور شماره 3 پرش و آدرس 2 در رجیستر 31 ذخیره شود. همانطور که مشخص است دستور بعدی دستور شماره 3 است که اجرا میشود پس پرش درست بوده به علاوه در دستور شماره 3 مقدار 4 در داخل رجیستر 17 قرار میگیرد و در دستور بعد باید مقدار رجیستر بازگشت یعنی رجیستر شماره 31 خوانده شود و در گام بعد pc برابر این مقدار قرار بگیرد همانطور که مشخص است در گام بعد دستور شماره 2 اجرا میشود پس هم jal آدرس بازگشت را به درستی ذخیره کرده و هم jr به درستی پرش می کند. نحوه پیاده سازی این 2 دستور به این صورت خواهد بود که اگر دستور از نوع jal باشد پرش دقیقا مشابه jump توضیح داده شده خواهد بود ولی ذخیره آدرس بازگشت در رجیستر شماره 31 نیازمند 2 ماکس یکی در ورودی write register و دیگری در ورودی write data ی بانک رجیستر خواهد بود. که سلکت هر دو همان کنترل jal بوده و ماکس ورودی write data از میان pc + 1 و ورودی ای که در سخت افزار اولیه بود یکی را انتخاب می کند و ماکس موجود در ورودی write register از میان رجیستر 31 و حاصل ماکس سخت افزار اولیه (که از میان 2 بخش دستور رجیستر مقصد را مشخص میکرد) یکی را انتخاب می کند به علاوه دستور jr با گذاشتن یک ماکس بر سر راه ورودی pc صورت گرفته که ورودی 1 آن read data1 و ورودی دوم همان مقدار ماکس سخت افزار مرحله قبل است که از میان ادرس پرش و

pc + 1 یکی را انتخاب میکرد. سلکت این ماکس زمانی 1 می شود که دستور از نوع jr باشد یعنی func دستور r type متعلق به دستور jr باشد.