به نام خدا

موضوع:

گزارش کار پروژه آخر

نام استاد : محمد لالی

نام درس : آزمایشگاه مدارهای منطقی و معماری کامپیوتر

رشته : مهندسی کامپیوتر

نام دانشجو : محمد پوربخت

شماره دانشجویی : ۹۸۲۰۲۳۰۰۷

زمستان ۱۴۰۰

پارت ۱:

ابتدا یک CPU کامل تک سایکل را طراحی و آن را با زبان Verilogپیاده سازی کردم.

این cpu از چندین ماژول مختلف تشکیل شده است که هرکدام به صورت جداگانه پیاده سازی و تست شده اند.

این ماژول ها عبارت اند از :

۱) Adder : در datapath دو adder داریم که یکی از آنها خروجی pc را با ۴ جمع می کند و دیگری خروجی pc را با خروجی immgen جمع می کند.

۲) Pc : این ماژول درواقع یک رجیستر است که با توجه به ورودی که از multiplexer می گیرد خروجی آدرس را به instruction memory می دهد.

۳) Instruction memory : با گرفتن ورودی از pc دستوری که در آن آدرس نوشته شده را می خواند.

۴) Register file : دستور خوانده شده به چند بخش تقسیم شده و بخشی از آن وارد این ماژول می شود تا read data1 و read data2 را تولید کند.

۵) Control : این ماژول با ورودی opcode ۷ داده تولید می کند که عبارتند از : branch، memRead، memToReg، AluOp، MemWrite، AluSrc، RegWrite . این داده ها را به ترتیب به and ، data memory ، multiplexer ، alu control ، data memory و data memory ، register file می دهد.

۶) Alu : خروجی تولید شده در ماژول Register file و یک multiplexer وارد این ماژول شده و zero و alu result را به عنوان خروجی برمی گرداند.

۷) Alu control : با گرفتن aluop از ماژول control و func3 و func7 استخراج شده از instruction ، خروجی را تولید کرده و به alu می دهد.

۸) Data memory : با دریافت address و write data داده read data را خروجی می دهد تا به همراه alu result برای multiplexer ارسال شوند.

۹) Multiplexer : در این طرح ۳ عدد از این ماژول داریم که با توجه به داده s خروجی را از بین ورودی ها انتخاب می کند. برای این پروژه این ماژول را به صورت ۶۴ بیتی پیاده سازی کردم. هر کدام از آن ها دارای یک ورودی اضافه به نام flag هستند که اگر مقدار آن ۱ باشد یعنی این ماژول همان مالتی پلکسری است که قبل از pc قرار دارد و اگر ۰ باشد یعنی یکی از دو مالتی پلکسر دیگر است.

۱۰) Immgen : این ماژول instruction را دریافت کرده و خروجی خود را به adder و multiplexer می دهد.

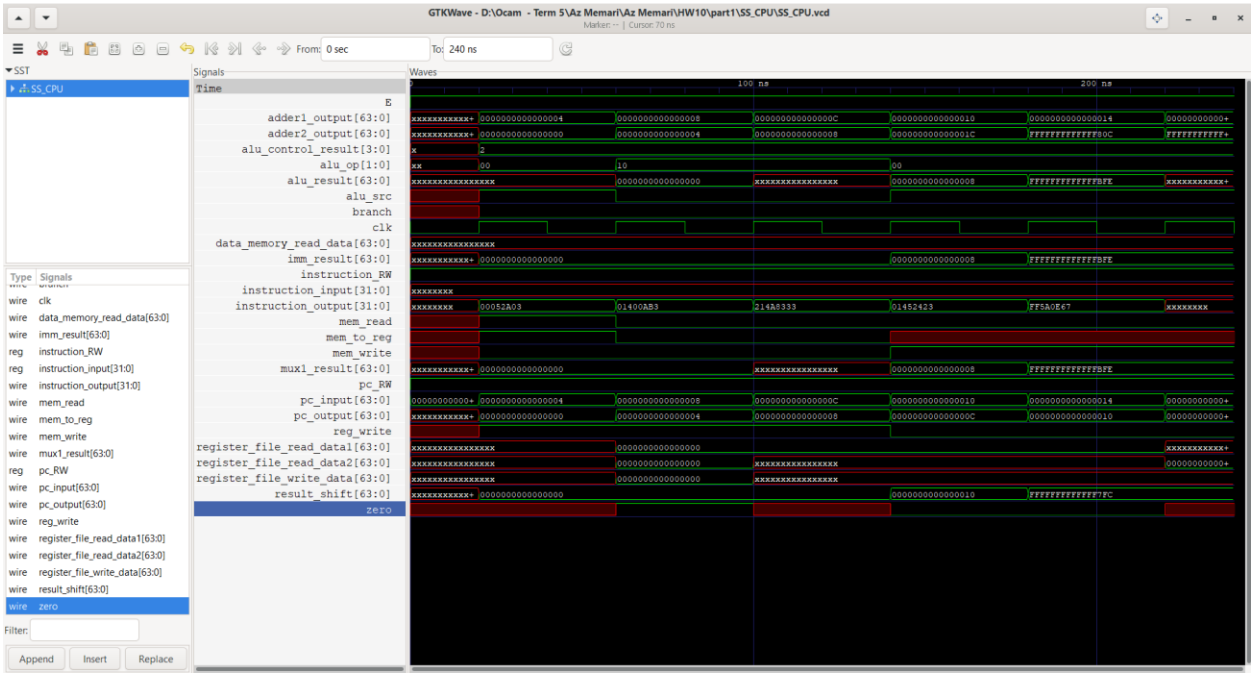
با توجه به دستور کار داده شده باید ۵ دستور زیر را به صورت پیشفرض به برنامه میدادیم تا اجرا شوند. لذا این دستورات را به فرم باینری تبدیل کرده و به instruction memory دادم.

```
lw x20, 0(x10)
add x21, 0, x20
sub x6, x21, x20
sw x20, 8(x10)
beq x20,x21,-4
```

فرم باینری دستورهای بالا:

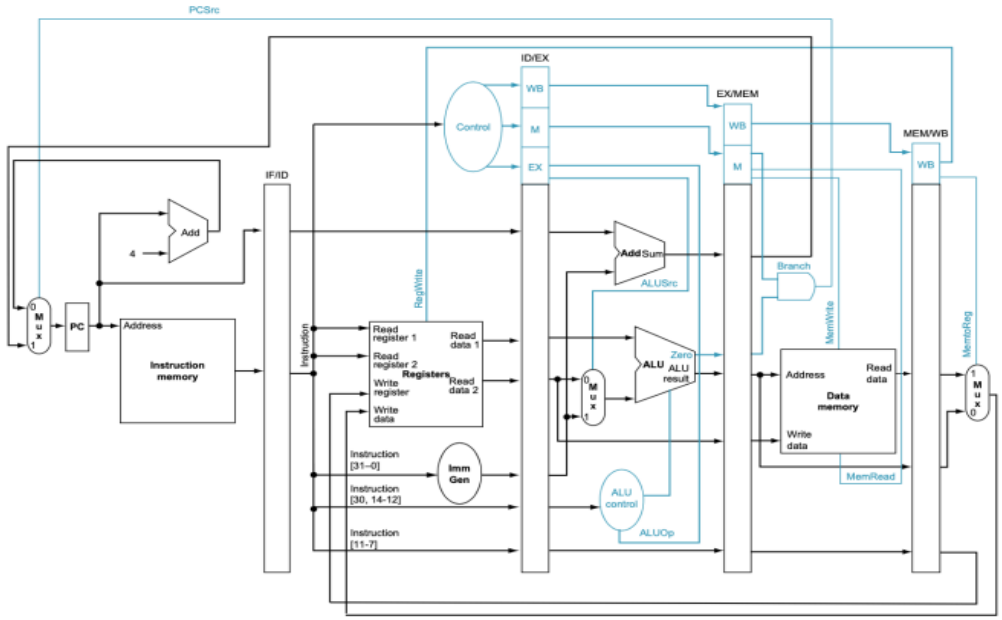
```
00000000000001010010101000000011
00000001010000000001010110011
01000010100101010000011100110011
00000001010001010010010000100011
11111111010110100000111001100111
```

پس از اجرای SS_CPU نتیجه به شکل زیر است:



پارت ۲ :

برای این بخش یک pipeline cpu را طراحی و پیاده سازی کردم که طرح آن به شکل زیر است:



تمامی ماژول های ۱ تا ۱۰ در این طرح هم وجود دارند با این تفاوت که با وجود ۴ رجیستر از هم جدا شده اند. این ۴ رجیستر CPU را به ۵ بخش تقسیم کرده اند. پیاده سازی هرکدام دقیقا مشابه رجیستر است اما دارای تعداد بیت ورودی و خروجی متفاوتی هستند.

رجیستر اول که IF/ID نام دارد دارای ورودی و خروجی ۹۶ بیتی است. رجیستر دوم که ID/EX نام دارد دارای ورودی و خروجی ۲۷۲ بیتی است. رجیستر سوم که EX/MEM نام دارد دارای ورودی و خروجی ۲۰۳ بیتی و رجیستر آخر به نام MEM/WB ورودی و خروجی ۱۳۵ بیتی دارد.

ورودی این رجیستر ها با توجه به datapath داده شده با چسباندن سیم های ورودی به وجود آمده و خروجی تولید شده با توجه به سیم های خروجی به قسمت های مختلف تقسیم شده و به بخش مورد نظر متصل شده است. If/id ورودی به ترتیب ۶۴ و ۳۲ بیت دریافت کرده و خروجی های ۶ ۵ ۵ ۳۲ ۳ و ۵ بیتی را تولید میکند. Id/ex ورودی های به ترتیب ۲ ۳ ۳ ۶۴ ۶۴ ۶۴ ۶۴ ۳ و ۵ بیتی را دریافت کرده و خروجی ۲۷۲ بیتی تولید کرده به صورت ۲ ۳ ۲ ۱ ۶۴ ۶۴ ۶۴ ۶۴ ۳ و ۵ بیتی تفکیک می کند و به ماژول های مشخص شده می دهد. Ex/mem ورودی های به ترتیب ۲ ۳ ۶۴ ۱ ۶۴ ۶۴ ۵ را دریافت و خروجی ۲۰۳ بیتی تفکیک شده به صورت ۲ ۱ ۱ ۱ ۶۴ ۱ ۶۴ ۶۴ ۵ بیتی میدهد. Mem/wb ورودی های به ترتیب ۲ ۶۴ ۶۴ ۵ را میگیرد و خروجی ۱۳۵ بیتی می دهد که به صورت ۱ ۱ ۶۴ ۶۴ ۵ بیتی تفکیک شده و به ماژول های مشخص شده در شکل متصل می کند.

در این روش با توجه به عدم وجود واحد forwarding ، hazard های زیر در برنامه وجود دارند:

۱) X20 در دستورات lw و add

۲) X21 در دستورات add و sub

با پیاده سازی واحد forwarding می توان hazard های احتمالی برنامه ها را مدیریت و رفع کرد.

پس از اجرای PL_CPU نتیجه به صورت زیر است: