به نام خدا

گزارشکار فاز اول پروژه درس معماری کامپیوتر

طراحی و پیاده‌سازی Data Path

استاد

دکتر حمید سربازی آزاد

اعضای گروه

محمدپارسا بشری ۴۰۰۱۰۴۸۱۲

محسن قاسمی ۴۰۰۱۰۵۱۶۶

امیرحسین رازلیقی ۹۹۱۰۲۴۲۳

بهار ۱۴۰۲

فهرست

مقدمه و هدف فاز اول ۲

طراحی Register File ۲

طراحی Memory ۲

طراحی ALU ۳

طراحی منطق Program Counter ۴

طراحی Control Unit ۵

طراحی ALU Control ۶

تست عملکرد ۷

تست عملکرد اجزای مدار ۷

تست عملکرد نهایی پردازنده ۸

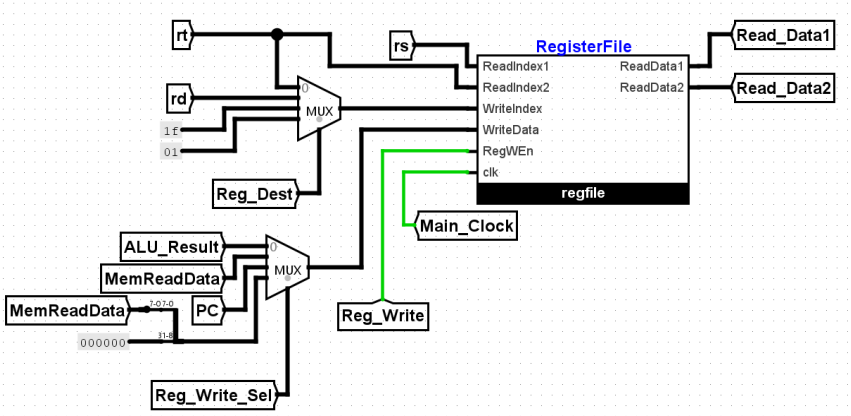
منابع و مراجع ۹

مقدمه و هدف فاز اول

هدف کلی این پروژه، طراحی و پیاده‌سازی یک پردازنده MIPS است. در فاز اول قصد داریم Datapath و Control Unit این پردازنده را به صورت Single Cycle طراحی و پیاده‌سازی کنیم. همچنین برای اطمینان از صحت عملکرد Component های پردازنده به صورت خودکار، تعدادی تست نیز برای ماژول‌هایمان می‌نویسیم. ماژول‌های Register File و Memory از قبل در اختیارمان قرار گرفته بود، بنابراین به توضیح مختصری درباره آن‌ها اکتفا می‌کنیم.

طراحی Register File

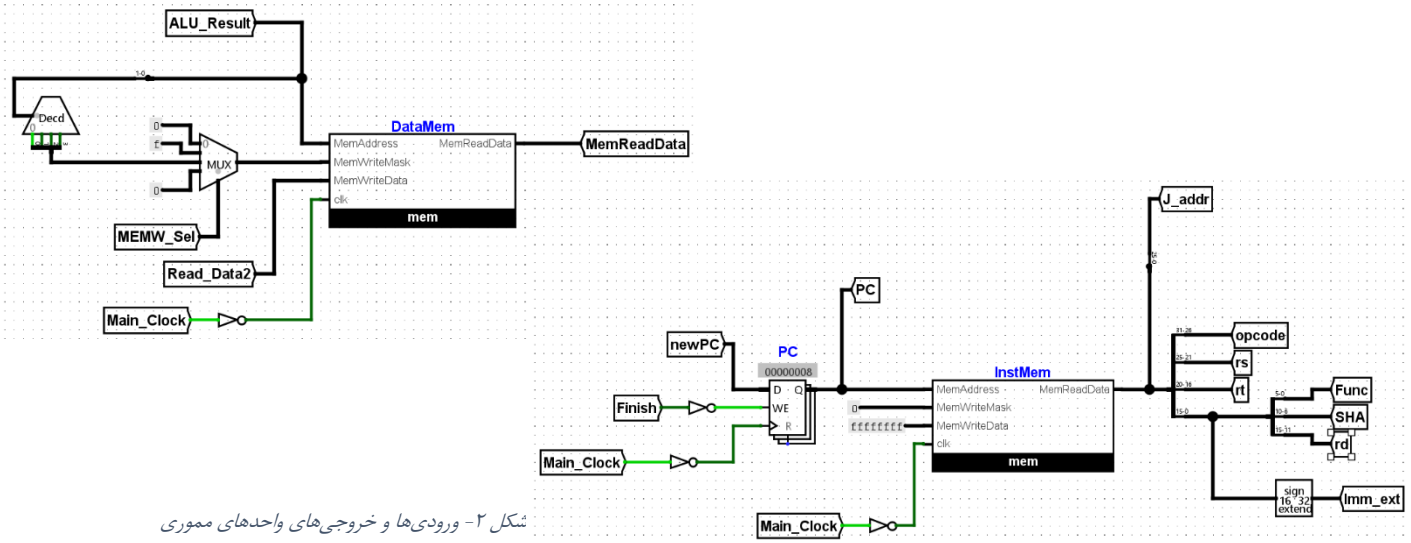
پردازنده MIPS پردازنده‌ای ۳۲ بیتی با معماری RISC است که ۳۲ عدد ثبات عمومی دارد. طراحی Register File به سادگی شامل ۳۲ عدد ثبات می‌شود که ورودی Write Enable آنها با استفاده از یک دیکودر 5×32 تولید می‌شود (ثبات شماره 0 قابل نوشتن نیست و همواره مقدار 0 را نگه می‌دارد). دو ورودی ۵ بیتی ReadIndex1 و ReadIndex2 نیز به عنوان ورودی select به دو MUX 5×32×32 داده می‌شوند تا محتویات ثبات‌هایی که باید خوانده شوند را انتخاب کنند.



شکل ۱- ورودی‌ها و خروجی‌های Register File

طراحی Memory

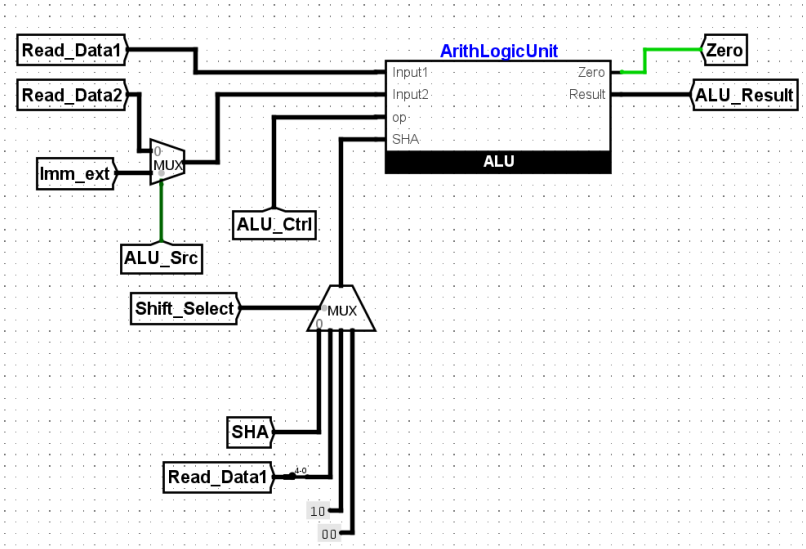
از دو حافظه مجزا برای داده‌ها و دستورات استفاده می‌کنیم (Data Memory و Instruction Memory) که هر کدام شامل 16K کلمه هستند و از آنجایی که هر کلمه ۴ بایت است، پس هر ماژول حافظه 64KB ظرفیت دارد که با استفاده از ۴ عدد RAM 16K×8 و ساختاری شبیه low-order interleaving ساخته می‌شود. به این صورت که دو بیت سمت راست را از Address حذف می‌کنیم و ۱۴ بیت باقی‌مانده را به هر چهار ماژول می‌دهیم. سپس برای نوشتن یک (یا چند) بایت خاص داخل یک کلمه، از ورودی Mask استفاده می‌کنیم.



شکل ۲- ورودی‌ها و خروجی‌های واحدهای ممووری

طراحی ALU

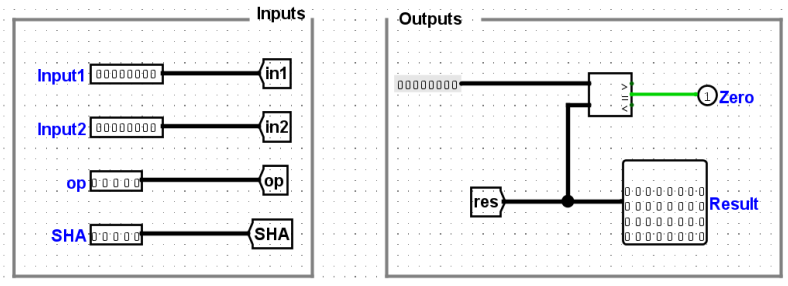
واحد محاسبات و منطق (ALU) وظیفه انجام عملیات‌های اصلی روی داده‌ها را دارد. این واحد دو ورودی دیتا (۳۲ بیتی) دریافت می‌کند و با توجه به ورودی operation (۵ بیتی)، عملیاتی که باید روی این دو داده انجام شود را انتخاب می‌کند. سپس خروجی عملیات انجام شده را از طریق خروجی Result (۳۲ بیتی) اعلام می‌کند. همچنین خروجی Zero (۱ بیتی) صفر بودن حاصل محاسبه را نشان می‌دهد.



شکل ۳- ورودی‌ها و خروجی‌های ALU

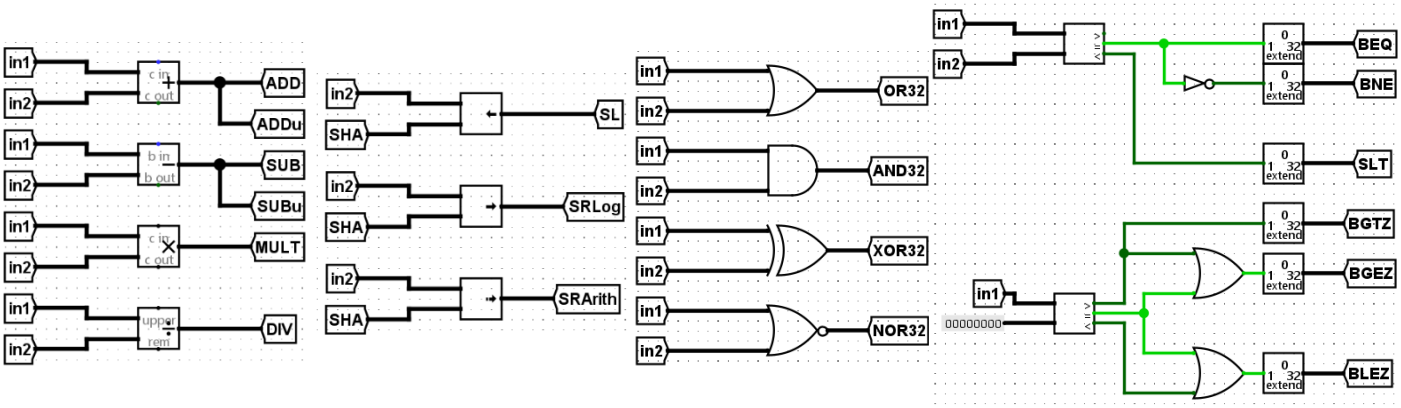
همچنین ورودی SHA مقدار شیفت را مشخص می‌کند. توجه کنید که ورودی‌های ALU توسط CU تولید می‌شوند.

حالا به طراحی داخلی ALU می‌پردازیم. ابتدا ترمینال‌های ورودی و خروجی را می‌سازیم. توجه کنید که خروجی Zero صرفا چک می‌کند که آیا Result صفر است یا خیر.



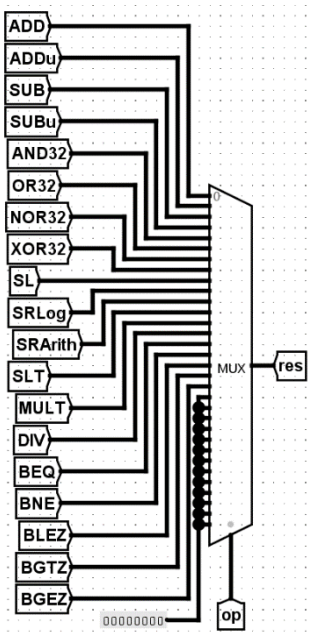
شکل ۴- ترمینال‌های ورودی و خروجی ALU

بدنه اصلی ALU شامل کامپوننت‌هایی است که محاسبات را انجام می‌دهند. در این قسمت تمام خروجی‌های ممکن را تولید می‌کنیم.



شکل ۵- تولید تمام خروجی‌های ممکن در ALU

سپس با استفاده از یک MUX 5×32×32 و با توجه به ورودی operation خروجی مورد نظرمان را انتخاب می‌کنیم.



شکل ۶- انتخاب خروجی مورد نظر با توجه به بیت‌های op

به ورودی‌های خالی MUX هم عدد 0 را وصل می‌کنیم. دلیل این کار را در قسمت طراحی ALU Control متوجه خواهیم شد.

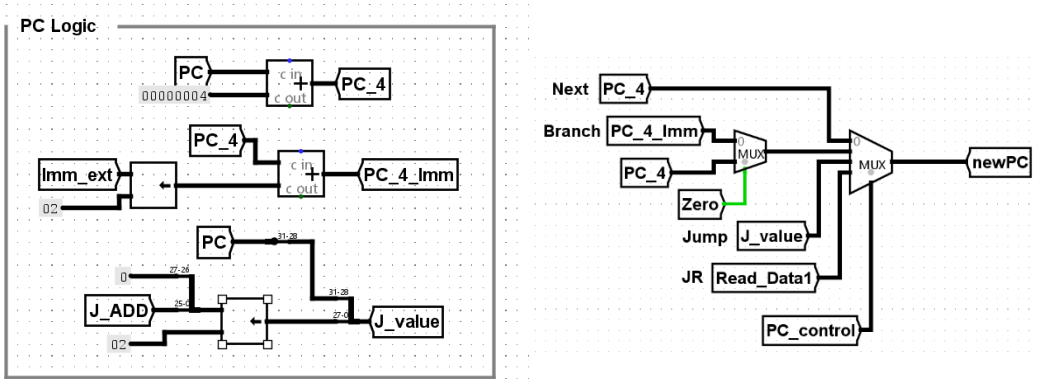
طراحی منطق Program Counter

همانطور که در شکل ۲ مشاهده کردیم، ثبات Program Counter یا به اختصار PC، یک ثبات ۳۲ بیتی است که آدرس دستور بعدی را در خود نگه می‌دارد. با توجه به طراحی Single Cycle در این فاز، PC باید با هر کلاک تغییر کند. مقدار جدید PC باید از بین چهار مقدار زیر انتخاب شود:

Next PC	Case
PC + 4	**Normal execution (next instruction)**
PC + 4 + SIGN_EXTEND(Imm|00)	**Branch instructions in I-format**
PC [31:28] | Address | 00	**Jump instructions in J-format (J and JAL)**
$rs	**JR (jump register) instruction in R-format**

جدول 1 - مقادیر جدید PC و روش انتخاب از بین آنها

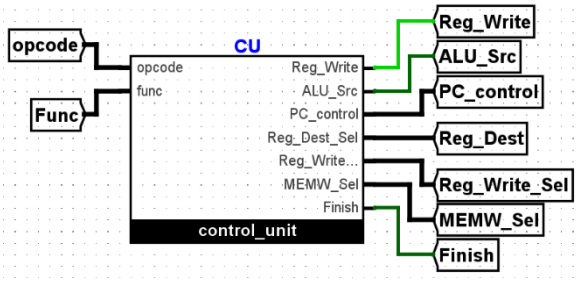
برای پیاده‌سازی منطق این کار، ابتدا چهار مقدار بالا را می‌سازیم و سپس با استفاده از یک MUX 2×4×1 از بین این چهار مقدار یکی را انتخاب کرده و به ورودی PC وصل می‌کنیم تا با کلاک بعدی وارد PC شود. توجه کنید که ورودی select این مالتی پلکسر از CU می‌آید.



شکل ۷- طراحی منطق PC

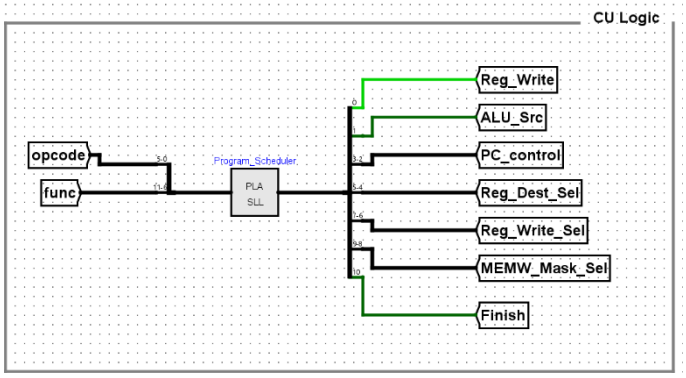
طراحی Control Unit

وظیفه واحد کنترل یا CU این است که با گرفتن Opcode و Func (هر دو ۶ بیتی) که از قالب دستور می‌آیند، خروجی‌های کنترلی لازم برای بقیه مدار (Datapath) را تولید کند. خروجی‌هایی که باید تولید شوند به مدارهایی که در قسمت‌های قبل آمده‌اند، داده می‌شوند.



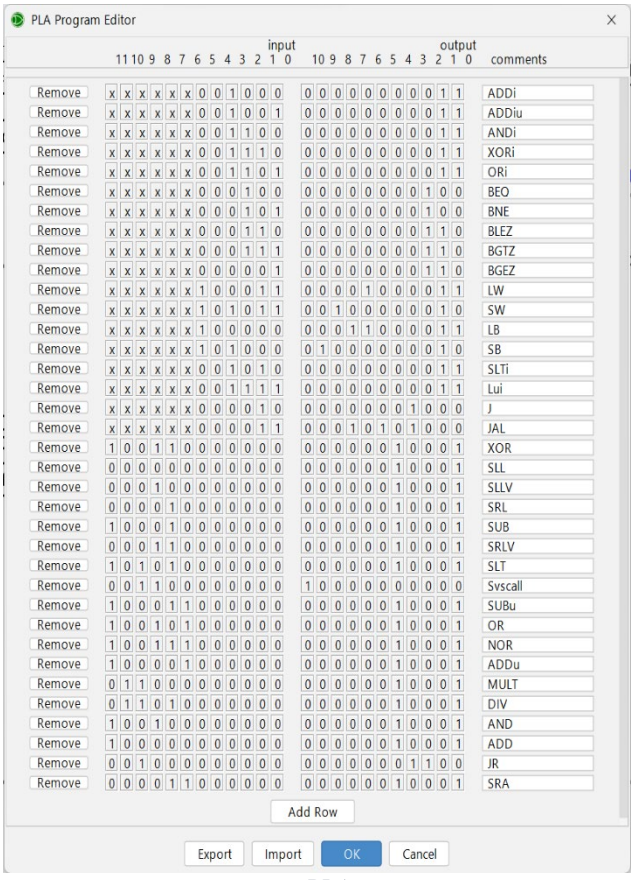
شکل ۸- ورودی‌ها و خروجی‌های Control Unit

طراحی داخلی CU را با استفاده از یک PLA انجام می‌دهیم. پس داخل ماژول CU به شکل زیر خواهد بود.



شکل ۹- استفاده از PLA در طراحی Control Unit

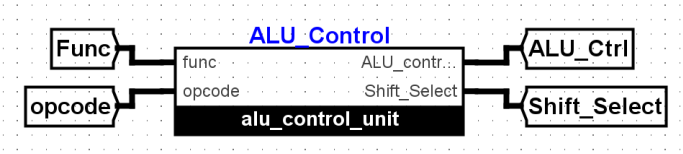
داخل PLA را با توجه به عملکرد مورد نیاز هر دستور، program می‌کنیم.



شکل ۱۰- مشخص کردن منطق داخل PLA

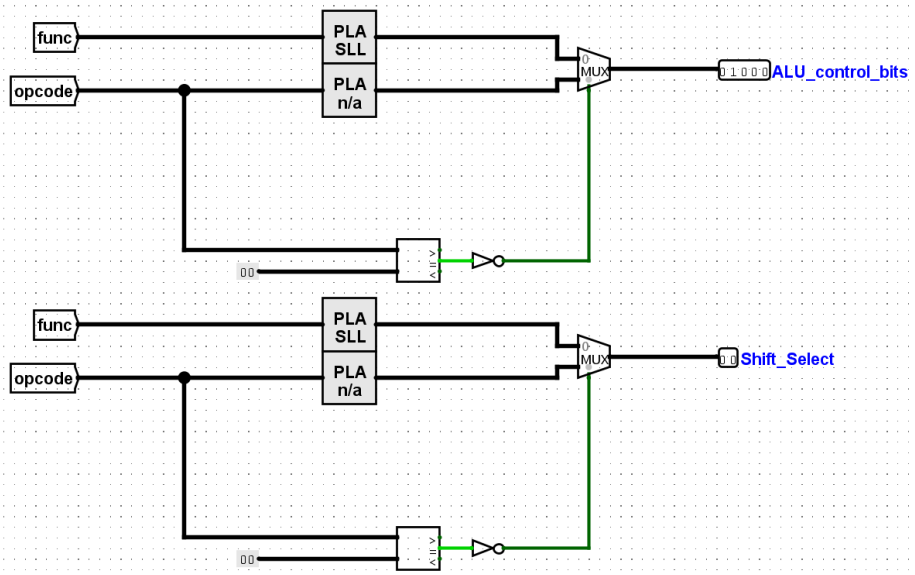
طراحی ALU Control

عملکرد ALU Control هم مانند CU است؛ یعنی به عنوان ورودی Opcode و Func را گرفته و خروجی‌های کنترلی مربوط به ALU را تولید می‌کند.



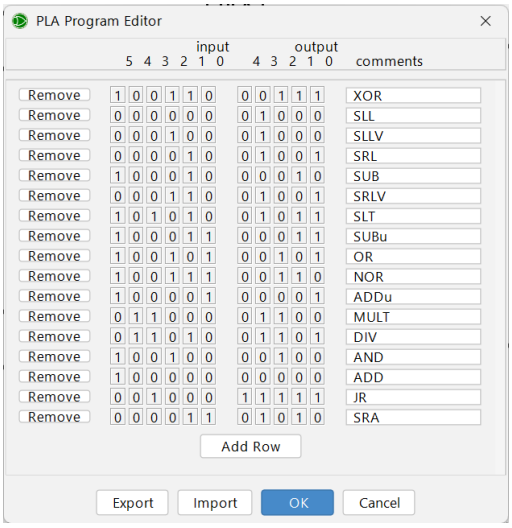
شکل ۱۱- ورودی‌ها و خروجی‌های ALU Control

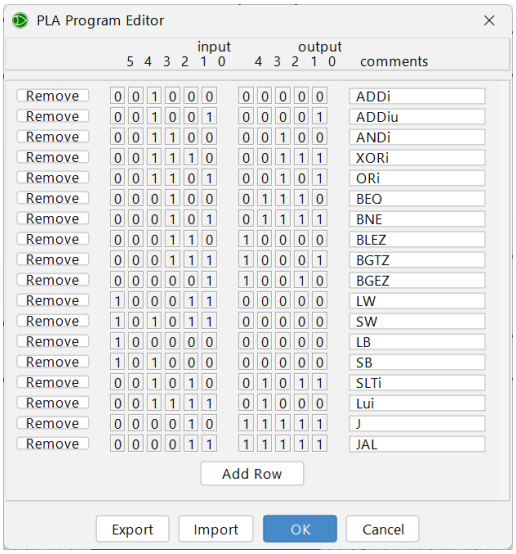
برای طراحی داخلی ALU Control از ۴ عدد PLA استفاده می‌کنیم. که به صورت زیر به خروجی‌ها متصل شده‌اند.

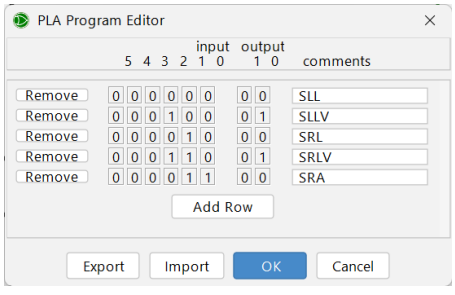


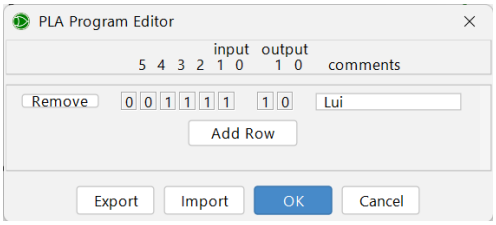
شکل ۱۲- طراحی داخل ALU Control

سپس منطق داخل هر PLA را مشخص می‌کنیم.









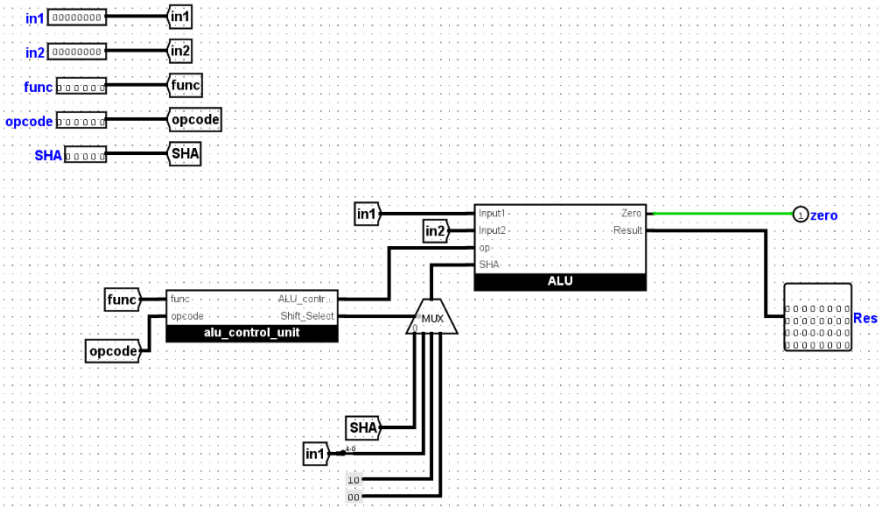
شکل ۱۳- مشخص کردن منطق داخل PLA ها

تست عملکرد

برای اطمینان از عملکرد صحیح اجزای مدار و همچنین عملکرد نهایی پردازنده، تست‌هایی طراحی میکنیم که بتوانیم به صورت خودکار عملکرد مدار را بررسی کنیم.

تست عملکرد اجزای مدار

در این بخش، عملکرد هر کدام از اجزای مدار مثل ALU و CU را می‌سنجیم. برای این کار، ابتدا یک ماژول جدید ساخته و از جزء مورد نظرمان یک نمونه می‌سازیم. به عنوان نمونه، برای تست عملکرد ALU، محتویات ماژول جدید به شکل زیر است:



شکل ۱۴- طراحی ماژول ALU Test

سپس در یک فایل با فرمت txt. تست‌ها را می‌نویسیم.

alu_test_vec.txt

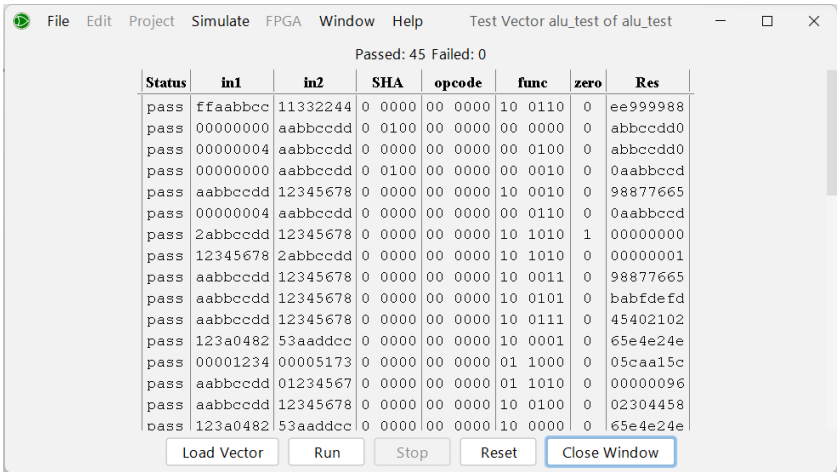
File Edit View

```
in1[32]     in2[32]     SHA[5]  opcode[6]  func[6]  zero[1]  Res[32]

# R-format
0xFFAABBCC  0x11332244  00000   000000     100110   0        0xee999988 # XOR
0x00000000  0xAABBCCDD  00100   000000     000000   0        0xABBCCDD0 # SLL
0x00000004  0xAABBCCDD  00000   000000     000100   0        0xABBCCDD0 # SLLV
0x00000000  0xAABBCCDD  00100   000000     000010   0        0x0AABBCCD # SRL
0xAABBCCDD  0x12345678  00000   000000     100010   0        0x98877665 # SUB
0x00000004  0xAABBCCDD  00000   000000     000110   0        0x0AABBCCD # SRLV
0x2ABBCCDD  0x12345678  00000   000000     101010   1        0x00000000 # SLT (more)
0x12345678  0x2ABBCCDD  00000   000000     101010   0        0x00000001 # SLT (less)
0xAABBCCDD  0x12345678  00000   000000     100011   0        0x98877665 # SUBu
0xAABBCCDD  0x12345678  00000   000000     100101   0        0xBABFDEFD # OR
0xAABBCCDD  0x12345678  00000   000000     100111   0        0x45402102 # NOR
0x123A0482  0x53AADDCC  00000   000000     100001   0        0x65E4E24E # ADDu
0x00001234  0x00005173  00000   000000     011000   0        0x05CAA15C # MULT
0xAABBCCDD  0x01234567  00000   000000     011010   0        0x00000096 # DIV
0xAABBCCDD  0x12345678  00000   000000     100100   0        0x02304458 # AND
0x123A0482  0x53AADDCC  00000   000000     100000   0        0x65E4E24E # ADD
0x123A0482  0x53AADDCC  00000   000000     001000   1        0x00000000 # JR
0x00000000  0xAABBCCDD  00100   000000     000011   0        0xFAABBCCD # SRA

# I-format
0x123A0482  0x53AADDCC  00000   001000     000000   0        0x65E4E24E # ADDi
0x123A0482  0x53AADDCC  00000   001001     000000   0        0x65E4E24E # ADDiu
0xAABBCCDD  0x12345678  00000   001100     000000   0        0x02304458 # ANDi
0xFFAABBCC  0x11332244  00000   001110     000000   0        0xee999988 # XORi
0xAABBCCDD  0x12345678  00000   001101     000000   0        0xBABFDEFD # ORi

0xAABBCCDD  0xAABBCCDD  00000   000100     000000   0        0x00000001 # BEQ (equal)
0xAABBCCDD  0xAABBCCD0  00000   000100     000000   1        0x00000000 # BEQ (not equal)
0xAABBCCDD  0xAABBCCDD  00000   000101     000000   1        0x00000000 # BNE (equal)
0xAABBCCDD  0xAABBCCD0  00000   000101     000000   0        0x00000001 # BNE (equal)
0xAABBCCDD  0x00000000  00000   000110     000000   0        0x00000001 # BLEZ (less than zero)
0x00000000  0x00000000  00000   000110     000000   0        0x00000001 # BLEZ (equal zero)
```

Ln 1, Col 1 100% Windows (CRLF) UTF-8

شکل ۱۵- فایل تست‌های مربوط به ALU

حالا از منوی Simulate روی Test Vector کلیک کرده و در پنجره‌ای که باز می‌شود، فایلی که ساخته‌ایم را Load می‌کنیم. سپس می‌توانیم نتیجه تست‌هایی که نوشته‌ایم را ببینیم:



Status	in1	in2	SHA	opcode	func	zero	Res
pass	ffaabbcc	11332244	0 0000	00 0000	10 0110	0	ee999988
pass	00000000	aabbccdd	0 0100	00 0000	00 0000	0	abbccdd0
pass	00000004	aabbccdd	0 0000	00 0000	00 0100	0	abbccdd0
pass	00000000	aabbccdd	0 0100	00 0000	00 0010	0	0aabbccd
pass	aabbccdd	12345678	0 0000	00 0000	10 0010	0	98877665
pass	00000004	aabbccdd	0 0000	00 0000	00 0110	0	0aabbccd
pass	2abbccdd	12345678	0 0000	00 0000	10 1010	1	00000000
pass	12345678	2abbccdd	0 0000	00 0000	10 1010	0	00000001
pass	aabbccdd	12345678	0 0000	00 0000	10 0011	0	98877665
pass	aabbccdd	12345678	0 0000	00 0000	10 0101	0	babfdefd
pass	aabbccdd	12345678	0 0000	00 0000	10 0111	0	45402102
pass	123a0482	53aaddcc	0 0000	00 0000	10 0001	0	65e4e24e
pass	00001234	00005173	0 0000	00 0000	01 1000	0	05caa15c
pass	aabbccdd	01234567	0 0000	00 0000	01 1010	0	00000096
pass	aabbccdd	12345678	0 0000	00 0000	10 0100	0	02304458
pass	123a0482	53aaddcc	0 0000	00 0000	10 0000	0	65e4e24e

شکل ۱۶- نتیجه اجرای تست‌های ALU

دیدیم که همه تست‌ها pass شدند. برای CU هم به همین صورت تست می‌نویسیم که به دلیل طولانی شدن گزارش و مشابهت بسیار بالا با روش تست ALU در اینجا به آن نمی‌پردازیم.

با استفاده از ترمینال و اجرای دستور زیر هم می‌توانستیم تست‌هایمان را اجرا کنیم.

```
~/nothing/mips-processor-team-3/circuits/tests | on main  logisim-evolution -w alu_test alu_test_vec.txt alu_test.circ
Loading test vector "alu_test_vec.txt" ...
Running 45 vectors ...
45
Passed: 45, Failed: 0
```

شکل ۱۷- اجرای تست‌ها با استفاده از ترمینال

تست عملکرد نهایی پردازنده

برای تست عملکرد نهایی پردازنده، باید یک برنامه را با آن اجرا کنیم. فرض کنید می‌خواهیم کد زیر را اجرا کنیم:

```
1     .globl main
2   main:
3       addi $s0, $zero, 1234
4       sw $s0, 1($zero)
5       lw $t0, 1($zero)
6       beq $s0, $t0, next
7       addi $t1, $zero, 12
8   next:
9       addi $t2, $zero, 15
10      sw $t2, 5($zero)
11      jr $ra
```

شکل ۱۸- قطعه کد استفاده شده در تست عملکرد پردازنده

ابتدا با استفاده از سایت https://alanhogan.com/asu/assembler.php که یک اسمبلر آنلاین است، کد ماشین را تولید می‌کنیم:

```
00400000: <main> ; <input:0> main:
00400000: 201004d2 ; <input:1> addi $s0, $zero, 1234
00400004: ac100001 ; <input:2> sw $s0, 1($zero)
00400008: 8c080001 ; <input:3> lw $t0, 1($zero)
0040000c: 12080001 ; <input:4> beq $s0, $t0, next
00400010: 2009000c ; <input:5> addi $t1, $zero, 12
00400014: <next> ; <input:6> next:
00400014: 200a000f ; <input:7> addi $t2, $zero, 15
00400018: ac0a0005 ; <input:8> sw $t2, 5($zero)
0040001c: 03e00008 ; <input:9> jr $ra
```

شکل ۱۹- خروجی اسمبلر

سپس با استفاده از دستور `cat code.txt | awk '{print $2}' | tail -n +2` کد باینری را از فایل جدا می‌کنیم. خروجی این دستور را در یک فایل ذخیره می‌کنیم و سپس اسکریپت پایتونی که نوشتیم را اجرا می‌کنیم. این قطعه کد، بایت‌های هر دستور را جدا کرده، چهار فایل ایجاد می‌کند و هر بایت از هر دستور را در یک فایل می‌نویسد.

Byte 3

```
v2.0 raw
20 ac 8c 12 20 20 ac 03
```

Byte 2

```
v2.0 raw
10 10 08 08 09 0a 0a e0
```

Byte 1

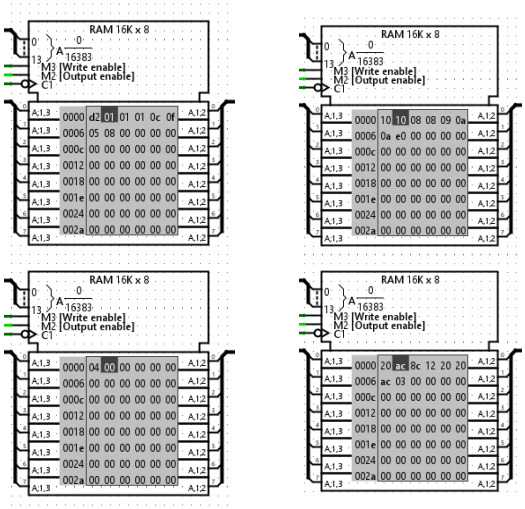
```
v2.0 raw
04 00 00 00 00 00 00 00
```

Byte 0

```
v2.0 raw
d2 01 01 01 0c 0f 05 08
```

شکل ۲۰- چهار فایل ایجاد شده توسط کد پایتون

در نهایت با راست کلیک روی هر ماژول حافظه (instruction memory) و انتخاب گزینه load image فایل مربوط به آن ماژول را در آن بارگذاری می‌کنیم.



شکل ۲۱- ماژول‌های حافظه بعد از لود کردن دستورات

حالا با هر بار زدن کلاک، یک دستور اجرا می‌شود. نتیجه اجرای دستورات را می‌توانیم با توجه به تاثیرشان روی یکی از ماژول‌های PC، Data Memory و یا Register File ببینیم. با توجه به طولانی بودن این فرآیند و تعداد بالای screenshot مورد نیاز، از آوردن نتایج در این گزارش صرف نظر می‌کنیم.

منابع و مراجع

- اسلایدهای درس
- https://www.cs.fsu.edu/~zwang/files/cda3101/Fall2017/Lecture5_cda3101.pdf
- https://alanhogan.com/asu/assembler.php
- https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html