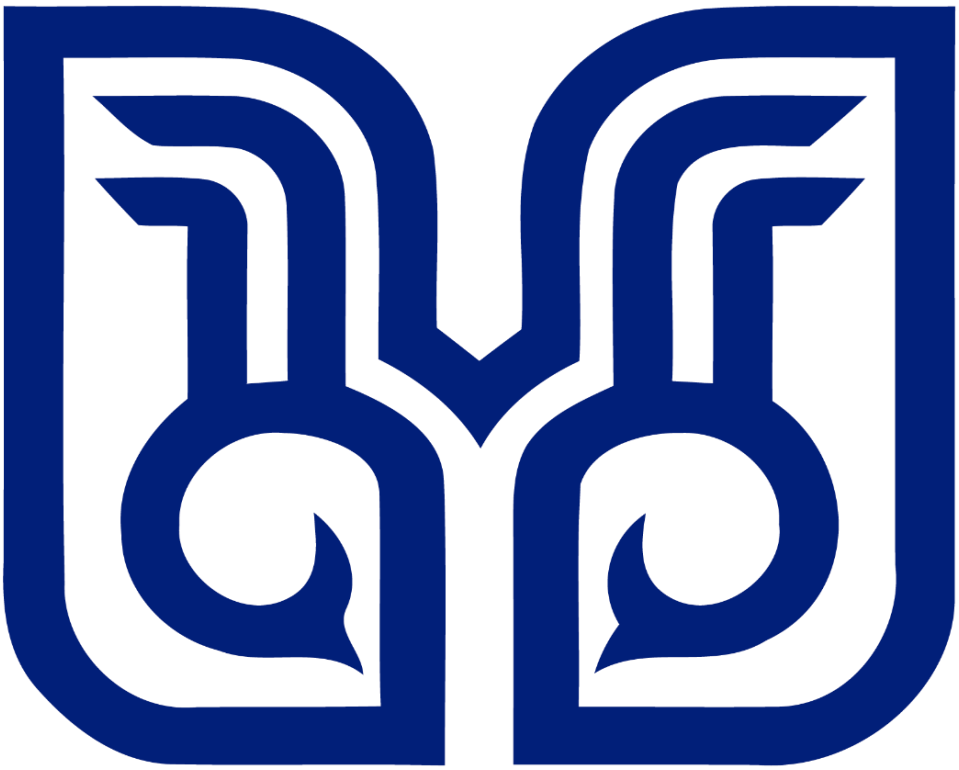

درس : معماری کامپیوتر

پروژه : طراحی پردازنده RISC-V 32Bit

استاد : دکتر مهدیه قزوینی

اعضا : نیما غفاری و پارسا زندی

تاریخ تحویل : ۱۴۰۴/۰۳/۲۰

Table of Contents

1. Introduction

طراحی و پیاده‌سازی یک پردازنده ۳۲ بیتی مبتنی بر معماری RISC-V با استفاده از نرم‌افزار Logisim Evolution گامی مهم در درک عمیق مفاهیم معماری رایانه و سیستم‌های دیجیتال به شمار می‌رود. معماری RISC-V به عنوان یک استاندارد باز و رایگان، فرصت‌های گسترده‌ای را برای پژوهشگران و توسعه‌دهندگان فراهم می‌سازد.

در این پروژه هدف اصلی ساخت یک هسته پردازنده ۳۲ بیتی بر اساس مجموعه دستورالعمل‌های RV32I است که شامل ۴۰ دستورالعمل پایه می‌باشد. برای تحقق این هدف، شش مؤلفه اصلی باید طراحی و یکپارچه‌سازی شوند : منطق شمارنده برنامه (PC) ، فایل ثبات‌ها (Register File) ، حافظه دستورالعمل (I-Mem) ، حافظه داده (D-Mem) ، واحد حساب و منطق (ALU) هر یک از این مؤلفه‌ها نقش حیاتی در عملکرد صحیح پردازنده ایفا می‌کنند و طراحی دقیق آن‌ها برای اطمینان از اجرای صحیح دستورالعمل‌ها ضروری است.

استفاده از Logisim Evolution به عنوان یک ابزار آموزشی و شبیه‌سازی، امکان طراحی و آزمایش مدارهای منطقی را به صورت بصری و تعاملی فراهم می‌سازد .این نرم‌افزار با ارائه محیطی گرافیکی، به دانشجویان و علاقه‌مندان اجازه می‌دهد تا مفاهیم پیچیده‌ای مانند مسیر داده، کنترل جریان، و تعامل بین مؤلفه‌های مختلف پردازنده را به صورت عملی تجربه کنند.

یکی از ویژگی‌های برجسته این پروژه، تمرکز بر ساخت اجزای پایه‌ای مانند فلیپ‌فلاپ‌ها و جمع‌کننده‌ها از ابتدا است .این رویکرد نه تنها درک عمیق‌تری از عملکرد داخلی پردازنده فراهم می‌آورد، بلکه مهارت‌های طراحی مدارهای دیجیتال را نیز تقویت می‌کند . همچنین، با پیاده‌سازی این پردازنده در Logisim Evolution ، امکان مشاهده و تحلیل رفتار پردازنده در پاسخ به دستورالعمل‌های مختلف فراهم می‌شود که برای آموزش و پژوهش بسیار مفید است.

در نهایت، طراحی و پیاده‌سازی یک پردازنده ۳۲ بیتی RISC-V با استفاده از Logisim Evolution ، نه تنها به عنوان یک پروژه آموزشی مفید است، بلکه بستری مناسب برای توسعه و آزمایش ایده‌های نوین در زمینه معماری رایانه فراهم می‌سازد .این پروژه می‌تواند پایه‌ای برای تحقیقات پیشرفته‌تر در زمینه‌هایی مانند بهینه‌سازی عملکرد، امنیت سخت‌افزاری، و طراحی سیستم‌های نهفته باشد.

1.1. RISC-V Green Card

در راستای طراحی و پیاده‌سازی پردازنده‌های مبتنی بر معماری‌های ساده و مؤثر، معماری**RISC-V** به عنوان یکی از مهم‌ترین دستاوردهای حوزه معماری مجموعه‌دستورالعمل‌ها (Instruction Set Architecture) در دهه اخیر مطرح شده است. این معماری با رویکردی بازمتن(Open Source) ، ماژولار و مقیاس‌پذیر طراحی شده و بستر مناسبی را برای تحقیقات دانشگاهی، توسعه صنعتی و آموزش مفاهیم معماری کامپیوتر فراهم ساخته است.

در این راستا، یکی از ابزارهای کلیدی برای درک عمیق‌تر ساختار و عملکرد این معماری، مستندات مرجع رسمی آن است . فایل**"RISC-V Green Card"** که به‌صورت فشرده و ساختاریافته، اطلاعات جامع و دقیقی از مجموعه دستورالعمل‌ها و قالب‌های کدگذاری آن را ارائه می‌دهد، یکی از این مراجع بسیار پرکاربرد به شمار می‌آید. این سند نقش یک راهنمای مرجع (Reference Guide) را ایفا کرده و به ویژه در مراحل طراحی و پیاده‌سازی پردازنده‌های RISC-V بسیار سودمند است.

محتوای این فایل شامل بخش‌های متعددی است که به تفصیل فرمت‌های مختلف دستورات معماری RISC-V را معرفی می‌کند. این فرمت‌ها شامل انواع مختلفی نظیر **R-type**، **I-type**، **S-type**، **B-type**، **U-type** و **J-type** هستند که هر یک ساختار کدگذاری خاص خود را دارند و در طراحی مسیر داده (Datapath) و واحد کنترل (Control Unit) اهمیت بالایی دارند.

علاوه بر مجموعه دستورالعمل‌های پایه‌ای **RV32I**، این مرجع شامل افزونه‌های استاندارد معماری نیز می‌باشد؛ از جمله:

- **RV32M** برای دستورات ضرب و تقسیم؛
- **RV32A** برای عملیات اتمی در پردازش‌های هم‌زمان؛
- **RV32F/D** برای پشتیبانی از محاسبات ممیز شناور تک‌دقت و دودقتی؛
- **RV32C** برای مجموعه دستورالعمل‌های فشرده که بهینه‌سازی مصرف حافظه را هدف قرار می‌دهند

از دیگر ویژگی‌های مهم این سند می‌توان به فهرست شبه‌دستورالعمل‌ها (Pseudo-Instructions) اشاره کرد که نگاشتی ساده‌تر و قابل فهم‌تر از ترکیب‌های پیچیده‌تر درون‌ساختی هستند. همچنین نگاشت کامل ثبات‌های معماری (Register Mapping) برای ثبات‌های عمومی و ثبات‌های ممیز شناور ارائه شده است که درک نقش و نحوه استفاده از آن‌ها در زمان اجرای برنامه‌ها را تسهیل می‌کند.

استفاده از این مرجع در روند طراحی یک پردازنده ۳۲ بیتی مبتنی برRISC-V ، به دانشجویان این امکان را می‌دهد که ضمن بهره‌گیری از استانداردهای جهانی در پیاده‌سازی، درک عمیق‌تری از نحوه رمزگشایی(Decoding) ، اجرای دستورالعمل‌ها، طراحی ALU و کنترلر و نیز ارتباط اجزای مختلف معماری پیدا کنند.

از این رو، فایل RISC-V Green Card به عنوان بخش مکمل و مرجع بنیادین پروژه حاضر، مورد استفاده قرار گرفته است و در تحلیل، پیاده‌سازی و اعتبارسنجی طراحی پردازنده ۳۲ بیتی مبتنی برRISC-V ، نقش محوری خواهد داشت.

RISC-V Reference

RISC-V Instruction Set

Core Instruction Formats

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

RV32I Base Integer Instructions

Inst	Name	FMT	Opcode	funct3	funct7	Description (C)	Note
add	ADD	R	0110011	0x0	0x00	rd = rs1 + rs2	
sub	SUB	R	0110011	0x0	0x20	rd = rs1 - rs2	
xor	XOR	R	0110011	0x4	0x00	rd = rs1 ^ rs2	
or	OR	R	0110011	0x6	0x00	rd = rs1 | rs2	
and	AND	R	0110011	0x7	0x00	rd = rs1 & rs2	
sll	Shift Left Logical	R	0110011	0x1	0x00	rd = rs1 << rs2	
srl	Shift Right Logical	R	0110011	0x5	0x00	rd = rs1 >> rs2	
sra	Shift Right Arith*	R	0110011	0x5	0x20	rd = rs1 >> rs2	msb-extends
slt	Set Less Than	R	0110011	0x2	0x00	rd = (rs1 < rs2)?1:0	
sltu	Set Less Than (U)	R	0110011	0x3	0x00	rd = (rs1 < rs2)?1:0	zero-extends
addi	ADD Immediate	I	0010011	0x0		rd = rs1 + imm	
xori	XOR Immediate	I	0010011	0x4		rd = rs1 ^ imm	
ori	OR Immediate	I	0010011	0x6		rd = rs1 | imm	
andi	AND Immediate	I	0010011	0x7		rd = rs1 & imm	
slli	Shift Left Logical Imm	I	0010011	0x1	imm[5:11]=0x00	rd = rs1 << imm[0:4]	
srli	Shift Right Logical Imm	I	0010011	0x5	imm[5:11]=0x00	rd = rs1 >> imm[0:4]	
srai	Shift Right Arith Imm	I	0010011	0x5	imm[5:11]=0x20	rd = rs1 >> imm[0:4]	msb-extends
slti	Set Less Than Imm	I	0010011	0x2		rd = (rs1 < imm)?1:0	
sltiu	Set Less Than Imm (U)	I	0010011	0x3		rd = (rs1 < imm)?1:0	zero-extends
lb	Load Byte	I	0000011	0x0		rd = M[rs1+imm][0:7]	
lh	Load Half	I	0000011	0x1		rd = M[rs1+imm][0:15]	
lw	Load Word	I	0000011	0x2		rd = M[rs1+imm][0:31]	
lbu	Load Byte (U)	I	0000011	0x4		rd = M[rs1+imm][0:7]	zero-extends
lhu	Load Half (U)	I	0000011	0x5		rd = M[rs1+imm][0:15]	zero-extends
sb	Store Byte	S	0100011	0x0		M[rs1+imm][0:7] = rs2[0:7]	
sh	Store Half	S	0100011	0x1		M[rs1+imm][0:15] = rs2[0:15]	
sw	Store Word	S	0100011	0x2		M[rs1+imm][0:31] = rs2[0:31]	
beq	Branch ==	B	1100011	0x0		if(rs1 == rs2) PC += imm	
bne	Branch !=	B	1100011	0x1		if(rs1 != rs2) PC += imm	
blt	Branch <	B	1100011	0x4		if(rs1 < rs2) PC += imm	
bge	Branch ≤	B	1100011	0x5		if(rs1 >= rs2) PC += imm	
bltu	Branch < (U)	B	1100011	0x6		if(rs1 < rs2) PC += imm	zero-extends
bgeu	Branch ≥ (U)	B	1100011	0x7		if(rs1 >= rs2) PC += imm	zero-extends
jal	Jump And Link	J	1101111			rd = PC+4; PC += imm	
jalr	Jump And Link Reg	I	1100111	0x0		rd = PC+4; PC = rs1 + imm	
lui	Load Upper Imm	U	0110111			rd = imm << 12	
auipc	Add Upper Imm to PC	U	0010111			rd = PC + (imm << 12)	
ecall	Environment Call	I	1110011	0x0	imm=0x0	Transfer control to OS	
ebreak	Environment Break	I	1110011	0x0	imm=0x1	Transfer control to debugger	

1

Figure 1-1

2. Program Counter (PC)

معرفی :

Program Counter (PC) یکی از اصلی‌ترین ثبات‌ها (registers) در هر معماری پردازنده است. در معماری RISC-V نیز، PC نقش حیاتی در کنترل جریان اجرای برنامه دارد. این ثبات، آدرس دستورالعملی که باید در چرخه بعدی CPU اجرا شود را نگه می‌دارد.

وظایف اصلی PC در RISC-V :

- نگهداری آدرس دستور فعلی : آدرس دستورالعملی را نگه می‌دارد که در حال اجراست یا قرار است اجرا شود.
- افزایش آدرس به صورت ترتیبی : در اکثر موارد، بعد از اجرای یک دستور، PC به اندازه ۴ بایت (در حالت دستورالعمل ۳۲ بیتی) افزایش می‌یابد تا به دستور بعدی اشاره کند.
- انجام پرش‌ها (jumps) یا انشعابات (branches) : اگر دستور پرش یا شرطی اجرا شود، مقدار PC ممکن است به‌صورت غیرخطی تغییر کند و به آدرس جدیدی تنظیم شود.

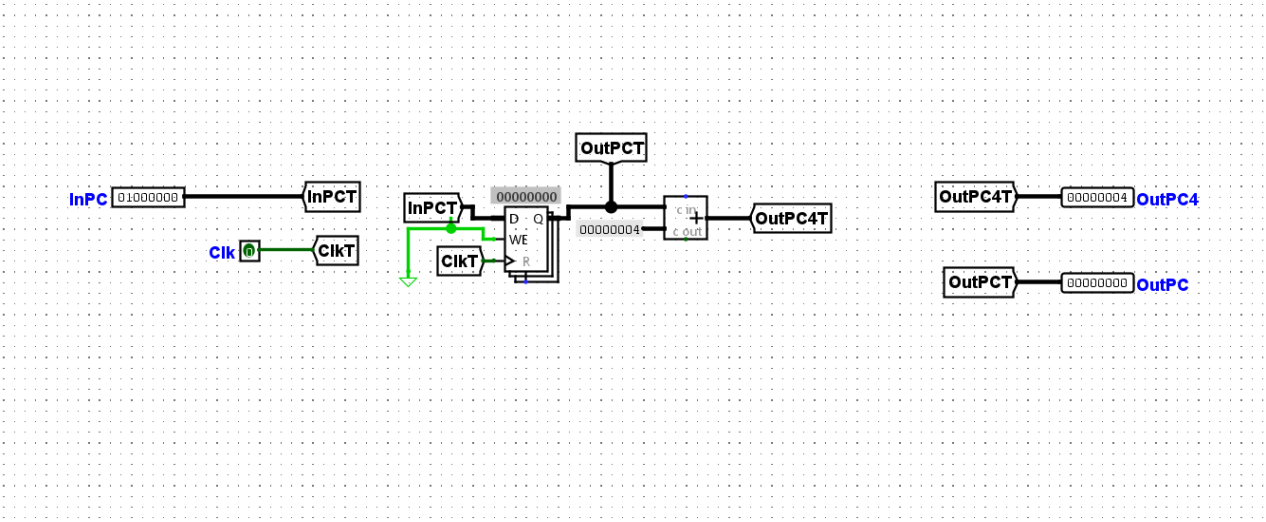

Figure 2-1

مراحل ساخت :

- ابتدا به یک ورودی در قالب Hexadecimal نیاز داریم که وارد رجیستری برای ذخیره سازی دستورالعمل می‌شود.

- خروجی رجیستر را به یکی از پایه‌های Full Adder متصل می‌کنیم و پایه‌ی دیگر را به یک ثبات با مقدار۴ برای مقدار ترتیبی دستورالعمل‌ها وصل می‌کنیم.
- OutPc حاوی مقدار دستورالعمل و OutPc4 حاوی مقدار offset خواهد بود

3. Arithmetic Logic Unit (ALU)

معرفی :

ALU (Arithmetic Logic Unit) وظیفه انجام عملیات‌های ریاضی و منطقی را بر عهده دارد. در معماری RISC-V، ALU می‌تواند انواع دستورها مانند جمع، تفریق، AND، OR، شیفت بیت‌ها و مقایسه‌ها را انجام دهد. این واحد معمولاً ورودی‌های خود را از رجیسترها دریافت می‌کند و نتیجه را نیز در رجیستری ذخیره می‌نماید.

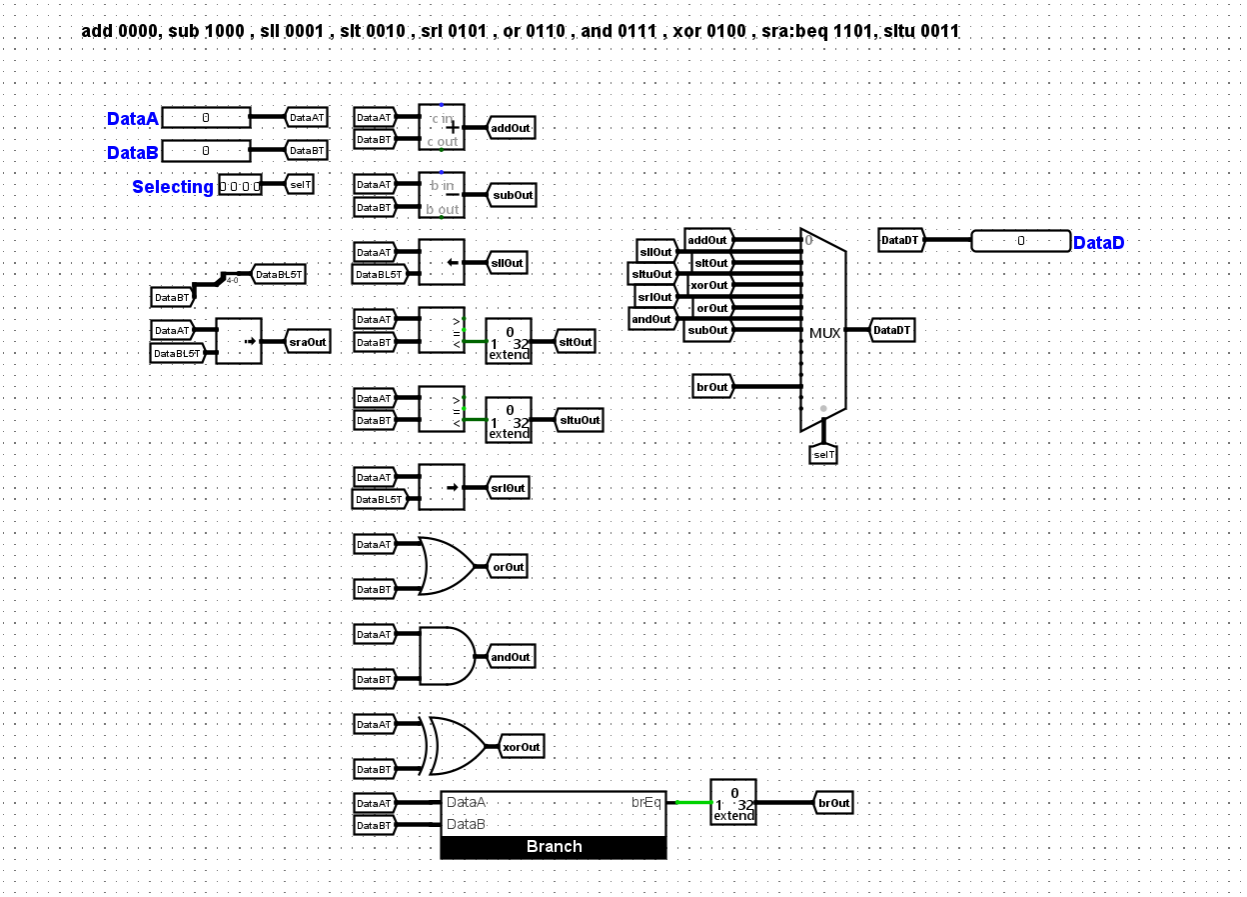

Figure 3-1

مراحل ساخت :

مرحله اول : تعریف ورودی‌ها و خروجی‌ها

در این طراحی، سه ورودی اصلی در نظر گرفته شده است:

- DataA (ورودی ۳۲ بیتی)
- DataB (ورودی ۳۲ بیتی)
- sel (چهار بیت جهت انتخاب نوع عملیات)
- خروجی نهایی با نام DataD، یک سیگنال ۳۲ بیتی است که نتیجه عملیات را ارائه می‌دهد.

مرحله دوم : ایجاد واحدهای عملیاتی

برای هر عملیات اختصاصی، یک واحد جدا ساخته می‌شود :

- جمع (Adder)
- تفریق (Subtractor)
- انواع شیفت (SLL، SRL، SRA)
- عملگرهای منطقی (AND، OR، XOR)
- مقایسه‌گرهای SLT و SLTU
- ورودی هر واحد مستقیماً به DataA و DataB متصل است و خروجی‌های آنها به مالتی‌پلکسر داده می‌شوند.

مرحله سوم : مقایسه و شاخه

بخش مقایسه (Branch) برای تشخیص برابری دو عدد (BEQ) استفاده می‌شود. در صورت برابری، خروجی این واحد به کمک یک "واحد بسط ۱ به ۳۲ بیت" به شکل مناسب برای استفاده در مالتی‌پلکسر آماده می‌شود.

مرحله چهارم: استفاده از مالتی‌پلکسر

تمام خروجی‌های عملیات مختلف با یک مالتی‌پلکسر واحد جمع می‌شوند. سیگنال sel تعیین می‌کند کدام خروجی به عنوان خروجی اصلی (DataD) انتخاب گردد.

جدول کدهای عملیاتی Sel :

Operand	sel
ADD	**0000**
SUB	**1000**
SLL	**0001**
SLT	**0010**
SLTU	**0011**
SRL	**0101**
OR	**0110**
XOR	**0100**
AND	**0111**

SRA	**1101**

Table 3-1

4. Register File

معرفی :

Register File یک ماژول حیاتی در اکثر پردازنده‌هاست که وظیفه ذخیره‌سازی موقت داده‌ها برای انجام سریع عملیات‌های پردازشی را بر عهده دارد. دیاگرام فوق معماری داخلی یک Register File با ۳۲ رجیستر را نمایش می‌دهد که امکان خواندن و نوشتن همزمان را فراهم می‌کند.

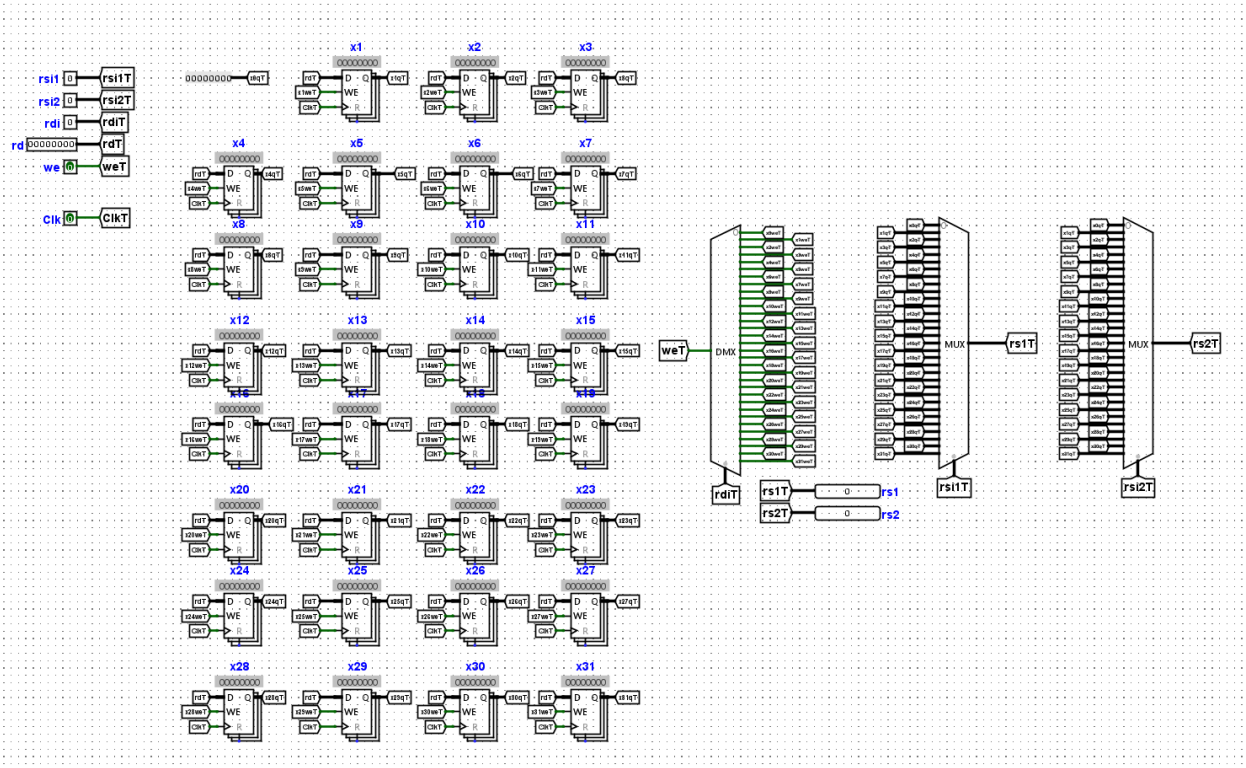

Figure 4-1

مراحل ساخت :

مرحله اول : تعریف ورودی‌ها و خروجی‌ها

ورودی‌ها و خروجی‌های این Register File عبارت‌اند از:

- rsi1 : آدرس رجیستر اول برای خواندن (rs1T)
- rsi2 : آدرس رجیستر دوم برای خواندن (rs2T)
- rd : آدرس رجیستری که مقدار باید در آن نوشته شود (Register Data)

- we : سیگنال فعال‌سازی نوشتن (Write Enable)
- wData : داده‌ای که باید نوشته شود
- clk : سیگنال ساعت سیستم (Clock)
- rs1T, rs2T : داده خروجی رجیستر اول و دوم به عنوان خروجی Register File

مرحله دوم : معماری داخلی

- ۳۲ رجیستر (x0 تا x31): هر رجیستر قابلیت خواندن و نوشتن را دارد.
- نوشتن : با فعال شدن سیگنال weT (Write Enable) و تطابق آدرس rdT در ورودی دی مالتی پلکسر (DMX)، دقیقا همان رجیستر مورد نظر مقدار جدید را (در لبه‌ی ساعت) دریافت می‌کند.
- خواندن : هر دو ورودی آدرس خواندن (rs1 و rs2) به دو مالتی‌پلکسر (MUX) متصل هستند که داده مورد نظر از هر رجیستر را به صورت همزمان و موازی روی دو خروجی ارائه می‌دهند.

مرحله سوم : واحد دی مالتی پلکسر (DMX)

- سیگنال نوشتن (weT) به ورودی دی مالتی پلکسر داده می‌شود.
- دی مالتی پلکسر آدرس، فقط اجازه نوشتن روی یک رجیستر خاص را صادر می‌کند و سایر رجیسترها در همان سیکل بدون تغییر باقی می‌مانند.

مرحله چهارم: مالتی‌پلکسرهای خواندن

- دو مالتی‌پلکسر بزرگ برای انتخاب خروجی‌های خواندن (rsi1T و rsi2T) وجود دارد.
- انتخاب خروجی هر مالتی‌پلکسر توسط آدرس ورودی مربوطه (rs1T و rs2T) انجام می‌شود و مقدار ذخیره‌شده رجیستر مورد نظر را به خروجی می‌فرستد.

مرحله پنجم: تست و صحت‌سنجی

- با نوشتن مقدار در یک رجیستر خاص (فعال کردن weT و تنظیم آدرس rdT)، و سپس خواندن همان آدرس توسط یکی از ورودی‌های خواندن (rs1T یا rs2T)، باید مقدار خوانده‌شده دقیقا با مقدار نوشته‌شده برابر باشد.

---

5. Immediate Generator (IMM)

معرفی:

Immediate Generator مداری است که بسته به نوع دستورالعمل، مقدار immediate (ثابت) را به‌درستی استخراج و سیگنال مورد نیاز را به قسمت‌های دیگر پردازنده ارسال می‌کند.

مراحل ساخت:

مرحله اول : ورودی

- دریافت سیگنال دستورالعمل (InsT) و سیگنال انتخاب نوع (SelectingT).

مرحله دوم : تفکیک دستورالعمل

- چندین ماژول اسپلیتر (Isplitor، Ssplitor، Bsplitor، Jsplitor، Usplitor) با توجه به نوع دستورالعمل، فیلدهای مختلف مربوط به immediate را جدا می‌کنند.

مرحله سوم : گسترش علامت (Sign Extend)

- مقدار immediate استخراج شده توسط هر ماژول (مثلاً SOutT، BOutT، JOutT، UOutT) به سایز ۳۲ بیت گسترش داده می‌شود.

مرحله چهارم : مالتی‌پلکسر

- یک مالتی‌پلکسر با توجه به سیگنال SelectingT یکی از immediateهای خروجی را روی خروجی اصلی (imm31_0T) قرار می‌دهد.

جمع‌بندی:

این مدار توانایی استخراج و گسترش انواع مختلف فیلدهای immediate از دستورالعمل‌های مختلف (مانند I-type، S-type، B-type و ...) را داراست و نقش کلیدی در اجرای صحیح دستورات پردازنده دارد.

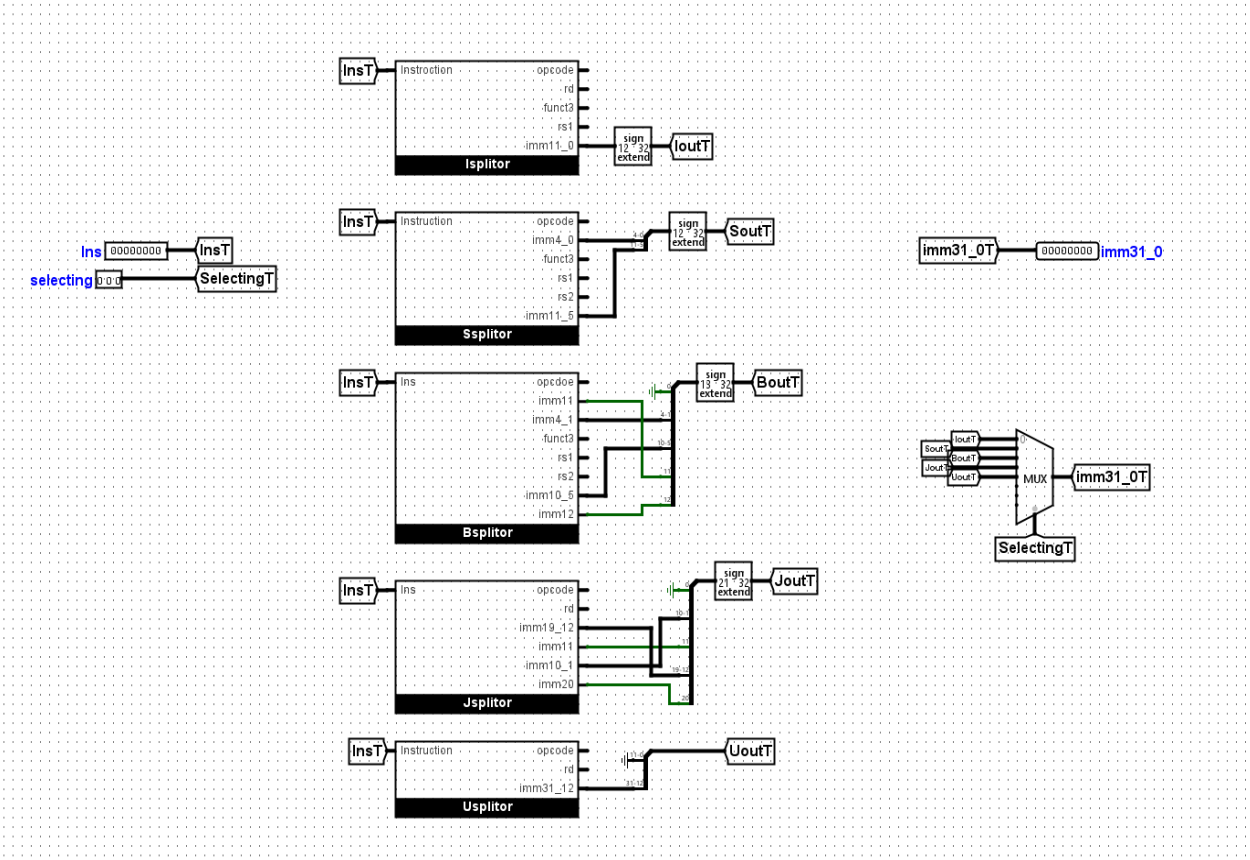

Figure 5-1

6. Load Control

معرفی :

در معماری پردازنده، واحد LoadCtrl مسئول مدیریت خواندن داده‌های مختلف (مانند بایت، نیم‌کلمه، کلمه کامل) از حافظه (RAM) و تبدیل آن‌ها برحسب نیاز دستورالعمل است. این ماژول قابلیت انتخاب نوع داده بارگذاری‌شونده (امضادار/بدون‌امضا) را نیز بر عهده دارد.

ماژول LoadCtrl به عنوان واسط اصلی پردازنده و حافظه عملیاتی، نقش عمده‌ای در صحیح و انعطاف‌پذیر بارگذاری داده‌ها در سطح سخت‌افزار دارد. این مدار با ترکیب مالتی‌پلکسرهای انتخاب نوع داده، اسپلیترها و اتصال مستقیم به RAM، کارآیی و دقت لازم برای بارگذاری انواع مختلف داده را تامین می‌کند.

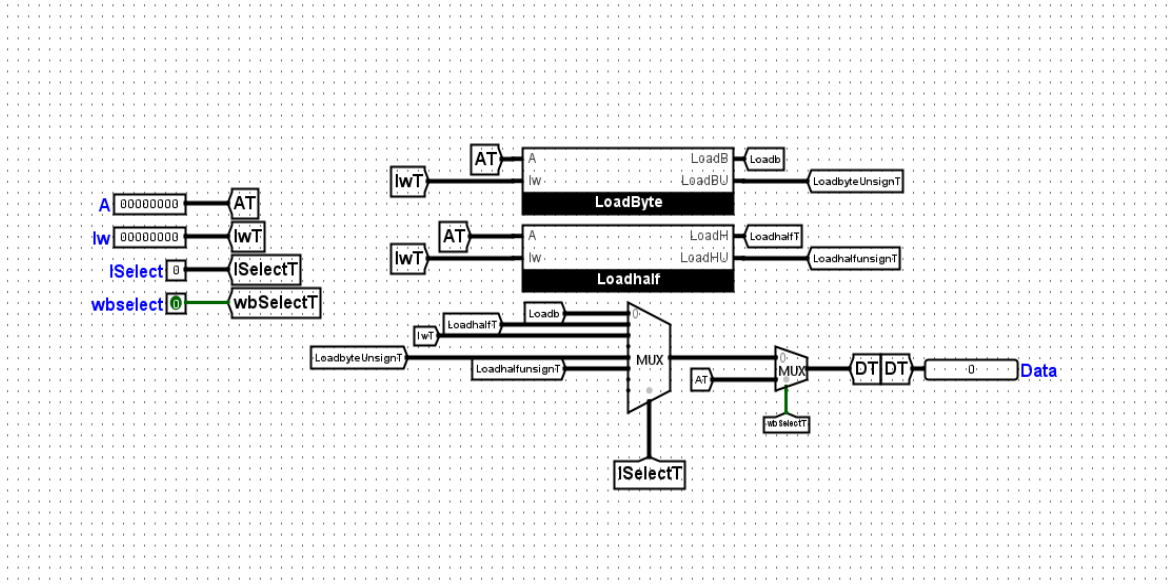

Figure 6-1

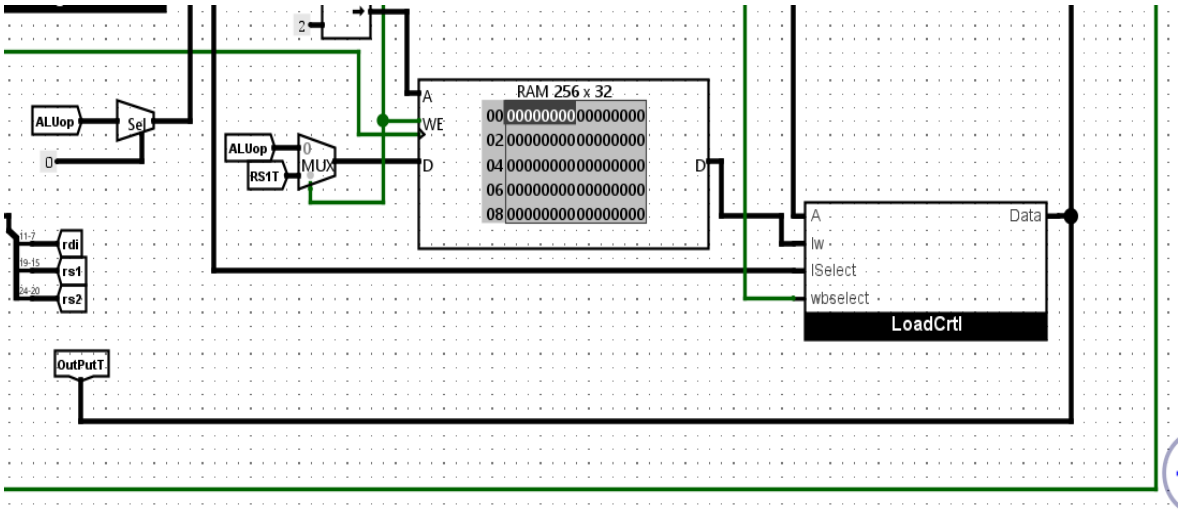

Figure 6-2

مراحل ساخت :

مرحله اول : تعریف ورودی و خروجی‌ها

ورودی‌ها:

- A (آدرس داده)
- Iw (داده ورودی/یا مقادیری برای انتخاب)
- ISelect (انتخاب نوع داده دریافتی: بایت، نیم‌کلمه و ...)
- wbSelect (انتخاب مسیر داده در مرحله بعد)

خروجی:

- Data (داده نهایی پردازش‌شده که آماده استفاده است)

مرحله دوم : ساختار داخلی ماژول LoadCtrl

ورودی‌ها به چند بلوک تقسیم‌کننده داده (Splitter) ارسال می‌شوند:

- LoadByte: استخراج و گسترش بایت (signed/unsigned)
- Loadhalf: استخراج و گسترش نیم‌کلمه (signed/unsigned)
- LoadB/LoadH: هرکدام برحسب بخش مورد نیاز داده را از کلمه ۳۲بیتی جدا می‌کنند

هر بخش **خروجی** مرتبط خود (مثلاً LoadbyteUnsignT) را به مالتی‌پلکسر مرکزی می‌دهد.

مرحله سوم : مالتی‌پلکسر مرکزی

- مالتی‌پلکسر (MUX) تمامی خروجی‌های واحدها را دریافت و بر اساس سیگنال ISelectT، یکی را به عنوان داده نهایی انتخاب می‌کند.
- مرحله بعدی، یک مالتی‌پلکسر دیگر (بر اساس wbSelectT)، داده انتخاب‌شده را به مسیر خروجی یا مسیرهای پردازشی دیگر هدایت می‌کند.

مرحله چهارم : اتصال به حافظه RAM

- آدرس‌دهی و سیگنال‌های مربوط به خواندن/نوشتن (A, WE, D) مستقیماً با ماژول RAM مرتبط هستند.
- در RAM با ظرفیت 256×32، بسته به سیگنال‌های ورودی، داده از آدرس مناسب خوانده یا در آن نوشته می‌شود.
- داده خروجی رام وارد ماژول LoadCtrl شده و پس از پردازش به واحدهای دیگر منتقل می‌شود.

مرحله پنجم : کنترل و تست

- عملکرد LoadCtrl بر اساس دستورالعمل‌های کنترلی و سیگنال‌های ISelect / wbSelect تنظیم می‌شود.
- با تست دستورالعمل‌های مختلف بارگذاری (LB, LBU, LH, LHU, LW)، صحت استخراج و ترکیب داده‌ها ارزیابی می‌شود.

---

7. Control Unit (ROM)

معرفی :

واحد کنترل (Control Unit) قلب فرمان‌دهی پردازنده است که بر اساس کد دستورالعمل ورودی، سیگنال‌های کنترلی مورد نیاز برای ماژول‌های مختلف (ALU، رجیستر فایل، حافظه و ...) را تولید می‌کند. در این معماری، پیاده‌سازی Control Unit با ROM و رمزگشایی بیت‌های دستورالعمل انجام شده است.

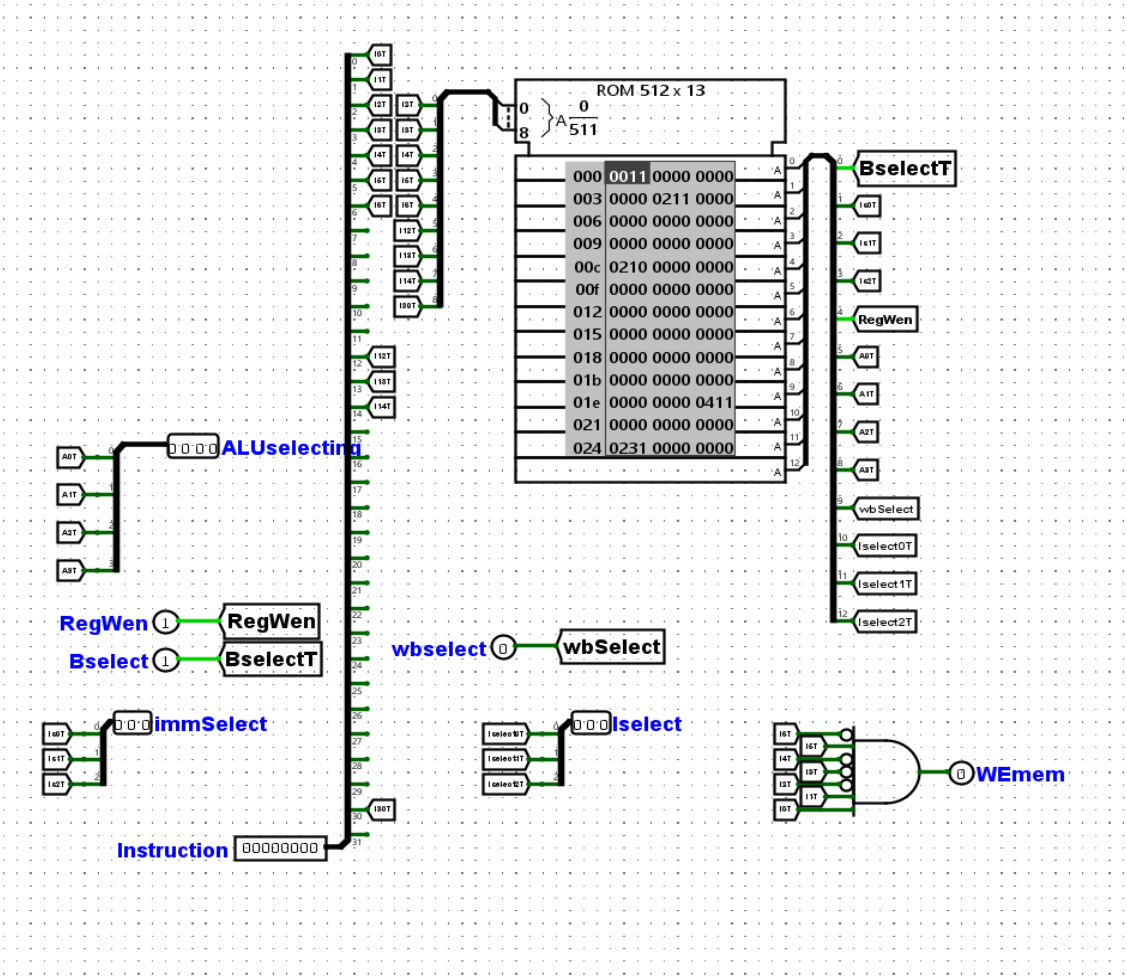

Figure 7-1

مراحل ساخت :

مرحله اول : تعریف ورودی‌ها و خروجی‌ها

ورودی:

- Instruction (دستورالعمل 32 بیتی که از حافظه برنامه خوانده می‌شود)

خروجی‌ها:

سیگنال‌های کنترلی شامل:

- RegWen (فعال‌سازی/غیرفعالسازی نوشتن رجیستر)
- Bselect، BselectT (انتخاب باس‌ها یا مسیر داده خاص)

- wbSelect، wbSelectT (انتخاب داده برای نوشتن در رجیسترها یا مرحله پایانی)
- Iselect, IselectT (انتخاب نوع بارگذاری/عملیات)
- WEmem (فعال‌سازی نوشتن در حافظه داده)
- ALUselecting (انتخاب عملکرد ALU)
- immSelect (سیگنال انتخاب immediate)

مرحله دوم : ساختار داخلی واحد کنترل

رمزگشایی آدرس :

- بیت‌های خاصی از ورودی دستورالعمل (مثلاً بیت‌های فانکشن و اپ‌کد) به عنوان آدرس به ROM داده می‌شوند.

ROM کنترل :

- این ROM با ظرفیت 512×13، در هر آدرس خود، مجموعه‌ای از سیگنال‌های کنترلی ورود به ماژول‌های دیگر را ذخیره کرده است.
- هر خط از ROM حاوی بیت‌هایی برای فعال/غیرفعال‌سازی و انتخاب عملکرد بخش‌های مختلف است.

خروجی‌های ROM:

- مستقیماً به خطوط کنترلی (مانند RegWen, BselectT, wbSelectT و ...) متصل‌اند که هرکدام به بخش مرتبط خود در پردازنده (ALU، حافظه، رجیستر فایل و ...) میروند.

مرحله سوم : نحوه عملکرد کلی

پس از دریافت یک دستورالعمل:

- بیت‌های کلیدی دستورالعمل به ورودی‌های آدرس‌دهی ROM متصل می‌شود.
- خروجی ROM در هر سیکل (طبق مقدار بیت‌های آدرس)، ترکیب مناسبی از سیگنال‌های کنترلی را فعال می‌کند.
- انتخاب سیگنال‌ها باعث انتخاب نوع عملیات ALU، نوع داده خواندن/نوشتن، نوع داده immediate و فعال/غیرفعال شدن نوشتن در حافظه یا رجیستر می‌شود.
- تمام خطوط کنترلی به صورت موازی به بخش‌های دیگر ارسال می‌شوند تا اجرای دقیق دستورالعمل مطابق معماری RISC-V تضمین شود.

مرحله چهارم : تست و ارزیابی

- تست این واحد با اجرای مجموعه‌ای از دستورالعمل‌ها با کدهای مختلف صورت می‌گیرد تا اطمینان حاصل شود سیگنال‌های کنترلی مرتبط برای هر opcode و func به درستی تولید می‌شوند.

واحد کنترل طراحی‌شده به کمک ROM و آدرس‌دهی مستقیم توسط بیت‌های کلیدی دستورالعمل، سیگنال‌های کنترلی مناسب برای عملکرد صحیح کل پردازنده RISC-V را تأمین می‌کند. این روش باعث انعطاف‌پذیری بالا و ساده‌سازی توسعه و تغییر کنترل در معماری می‌شود.

8. Datapath (ROM)

معرفی :

پردازنده‌های مبتنی بر معماری RISC-V با طراحی ماژولار، اجزای متعددی مانند واحد دستورالعمل، رجیستر فایل، ALU، ماژول حافظه (RAM)، واحد کنترل (Control Unit)، تولیدکننده Immediate و سیستم مسیریابی داده (Data Path) را شامل می‌شوند. در این پروژه، یک مسیر داده کامل طراحی شده است که واحد کنترل رام (ControlLogicROM) وظیفه‌ی رمزگشایی دستورالعمل و تولید سیگنال‌های کنترلی را برعهده دارد.

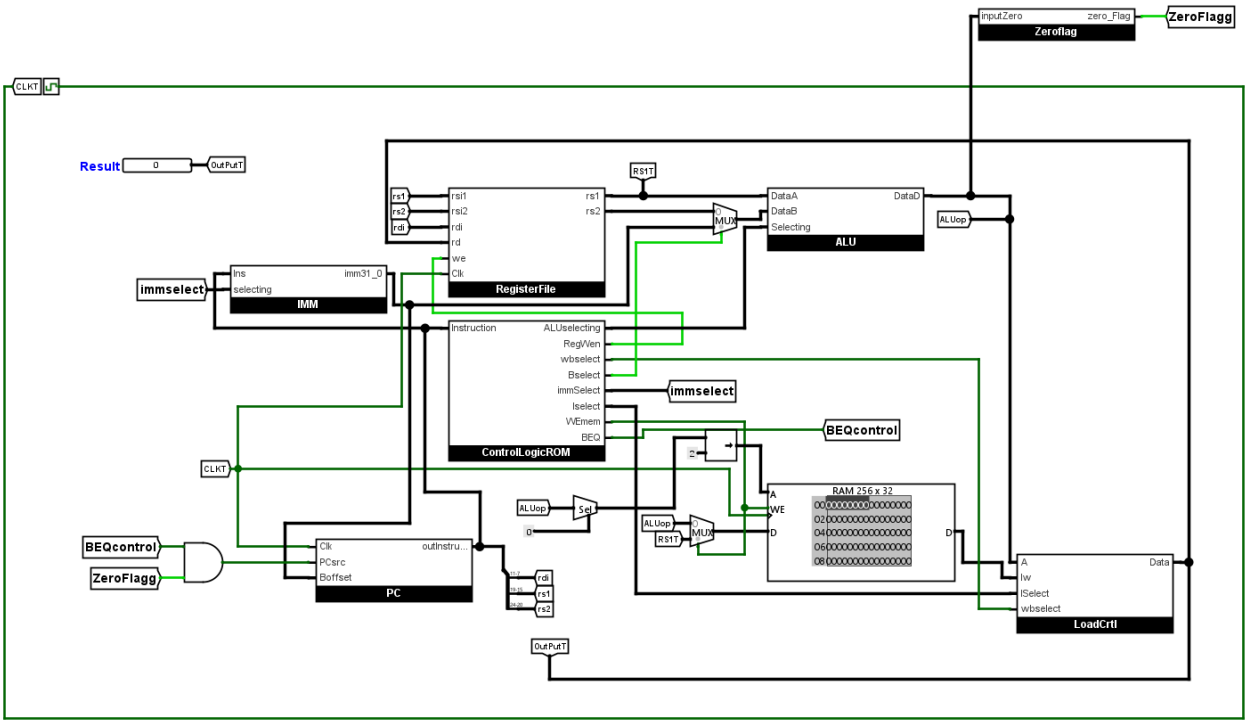

Figure 8-8-1

8.1. Main Components :

۱. Register File

- وظیفه: ذخیره و تأمین داده‌های ورودی/خروجی رجیسترها
- ورودی/خروجی: آدرس رجیسترهای خواندنی (rs1, rs2) و آدرس رجیستر مقصد (rd) به همراه سیگنال نوشتن (WE)
- توضیح: داده‌های ورودی جهت انجام عملگرها به ALU ارسال می‌شوند.

۲. ALU (Arithmetic Logic Unit)

- وظیفه: انجام عملیات محاسباتی/منطقی مانند جمع، تفریق، AND، OR و ...

- ورودی/خروجی: داده‌های دو ورودی (DataA, DataB)، سیگنال انتخاب عملکرد (Selecting)، خروجی داده نهایی
- توضیح: انتخاب ورودی دوم توسط مالتی‌پلکسر (rs2 یا مقدار Immediate)

۳. IMM (Immediate Generator)

- وظیفه: استخراج و تشکیل مقدار Immediate از دستورالعمل
- ورودی/خروجی: دستورالعمل (Instruction)، سیگنال انتخاب نوع Immediate (immselect)
- توضیح: برای دستورالعمل‌هایی مانند load/store و نوع immediate

۴. ControlLogicROM

- وظیفه: رمزگشایی بیت‌های دستورالعمل و تولید سیگنال‌های کنترلی برای کل مسیر داده

سیگنال‌های خروجی:

- ALUselecting: انتخاب عملکرد ALU
- RegWen: فعال‌سازی نوشتن در رجیستر
- Bselect: انتخاب ورودی دوم ALU
- iSelect: انتخاب Memory یا ALU برای نوشتن در رجیستر
- Immsel: انتخاب Immediate مورد استفاده
- WEmem: فعال‌سازی نوشتن در RAM

۵. RAM (Memory)

- وظیفه: حافظه داده برای بارگذاری (Load) یا ذخیره سازی (Store)
- ورودی/خروجی: آدرس، داده ورودی/خروجی، سیگنال WE برای فعال‌سازی نوشتن

۶. مسیر بازگشت داده(Write Back)

نتیجه نهایی عملیات) چه از حافظه و چه از (ALU از طریق MUX به فایل ثبات‌ها ارسال می‌شود. سیگنال‌های wbselectو Iselectتعیین می‌کنند که چه داده‌ای به ثبات مقصد (rd) نوشته شود.

۷. LoadCtrl

- وظیفه: کنترل دسترسی به حافظه بر اساس سیگنال و ماسک (برای Loadهای مختلف: byte, halfword, word)
- ورودی/خروجی: آدرس و داده RAM، سیگنال‌های انتخاب

۸. حافظه داده(Load/Store)

در صورت اجرای دستوراتی مانند lw یا sw ، واحد کنترل LoadCtrl مشخص می‌کند که داده‌ای از حافظه خوانده یا به آن نوشته شود. آدرس حافظه از ALU و داده از ثبات‌ها یا Immediate به حافظه منتقل می‌شود.

۹. PC (Program Counter)

- وظیفه : نگهداری و افزایش آدرس فعلی دستورالعمل
- ورودی/خروجی : مقدار فعلی PC و مقدار بعدی خروجی به حافظه دستورالعمل (در اینجا ورودی جدا تعریف نشده)

۱۰. کنترل پرش(Branch)

در صورت وجود دستور پرشی مانند BEQ، خروجی ALU با صفر مقایسه شده و سیگنال پرچم صفر (ZeroFlagg) فعال می‌شود. در این حالت، ترکیب سیگنال BEQcontrol و پرچم صفر تعیین می‌کند که آیا PC باید به آدرس جدید بپرد یا خیر.

Figure 8-2

۱۰. مالتی‌پلکسرها و سیگنال‌های کنترلی

- مالتی‌پلکسر انتخاب ورودی ALU: تعیین می‌کند مقدار دریافتی از رجیستر فایل یا Immediate به ورودی دوم ALU داده شود (سیگنال کنترل: Bselect)
- مالتی‌پلکسر انتخاب داده خروجی نهایی: با توجه به نوع دستور، مقدار نهایی از ALU یا حافظه RAM به رجیستر فایل ارسال می‌شود (کنترل با سیگنال iSelect)

نحوه عملکرد کلی مسیر داده:

- آغاز سیکل: ورودی دستورالعمل، همراه با مقدار فعلی PC، وارد مسیر داده می‌شود.
- رمزگشایی دستورالعمل: بیت‌های کلیدی توسط واحد Splitter جداسازی و به ControlLogicROM ارسال می‌شود.

- تولید سیگنال‌های کنترل: ROM با توجه به نوع دستورالعمل، سیگنال‌هایی مانند ALUselecting، RegWen، Bselect و غیره را تولید می‌کند.

فعالسازی داده‌ها با توجه به سیگنال‌های ROM:

- داده‌ها از رجیستر فایل یا Immediate به مالتی‌پلکسر ALU می‌روند.
- عملیات مورد نظر در ALU روی داده‌ها انجام می‌شود.
- در صورت دستور حافظه، انتقال داده بین RAM و رجیستر فایل توسط LoadCtrl و مالتی‌پلکسر خروجی صورت می‌گیرد.
- به‌روزرسانی PC : مقدار جدید PC با توجه به دستورالعمل بعدی به ماژول PC داده می‌شود.

سیگنال‌های کلیدی کنترلی :

- Bselect : ورودی دوم ALU (rs2 یا Immediate)
- ALUselecting : نوع عملکرد ALU (جمع، تفریق و ...)
- RegWen : اجازه نوشتن در رجیستر فایل
- iSelect : انتخاب مسیر داده خروجی (از حافظه یا از ALU)
- WEmem : فعال‌سازی نوشتن در RAM در دستور store
- immSelect : تعیین نوع Immediate استخراج‌شده از دستور

---

9. IM (Instruction Memmory)

در این بخش از طراحی پردازنده، یک واحد ROM با ظرفیت $64K \times 32$ بیت به‌عنوان حافظه دستورالعمل مورد استفاده قرار گرفته است. این حافظه وظیفه نگهداری و ارائه دستورالعمل‌ها به واحد کنترل را بر عهده دارد.

مشخصات کلی حافظه ROM :

- ظرفیت آدرس‌دهی 64K :مکان (2^{16} = 65536 آدرس)
- طول هر کلمه ۳۲ بیت
- نوع حافظه : فقط‌خواندنی (Read-Only Memory)

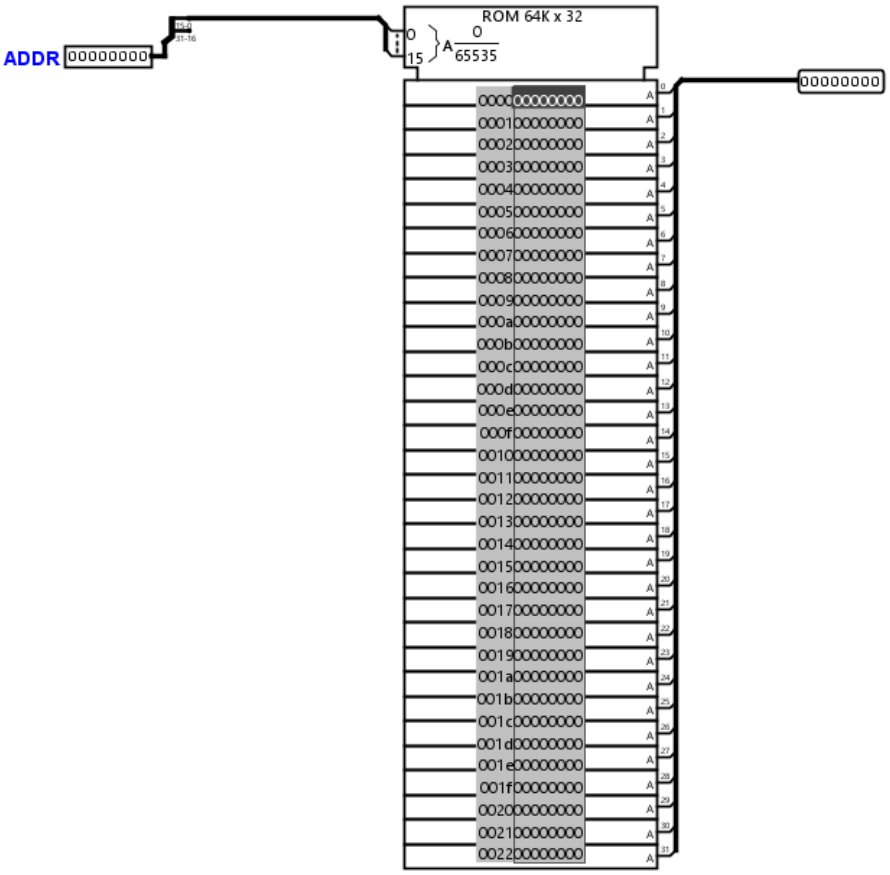

Figure 8-2

اجزای اصلی دیاگرام :

۱. ورودی آدرس (ADDR) :

ورودی آدرس 16 بیتی در سمت چپ با عنوان ADDR نشان داده شده است. این مقدار معمولاً از رجیستر برنامه‌شمار (PC) تأمین می‌شود و مشخص می‌کند که کدام مکان از حافظه باید خوانده شود.

۲. گذرگاه آدرس(A[15:0]) :

سیم‌کشی 16 بیتی از ورودی ADDR به ورودی آدرس حافظه ROM متصل شده است. این گذرگاه آدرس مشخص می‌کند که داده از کدام موقعیت در حافظه ROM خوانده شود.

۳. حافظه ROM:

ماژول ROM در مرکز شکل با ظرفیت 64K × 32 نمایش داده شده و مقادیر آن از پیش تعیین شده‌اند. هر سطر، نشان‌دهنده یک مکان حافظه است که شامل یک مقدار ۳۲ بیتی می‌باشد.

۴ . خروجی داده(A) :

در سمت راست حافظه، خروجی ۳۲ بیتی داده قرار دارد که در پاسخ به آدرس ورودی، مقدار متناظر با آن آدرس را در خروجی قرار می‌دهد. این مقدار معمولاً به واحد کنترل داده می‌شود تا رمزگشایی و اجرا شود.

نحوه عملکرد:

۱. مقدار PC (یا هر منبع دیگر) به‌عنوان ورودی ADDR داده می‌شود.

۲. مقدار ADDR از طریق باس آدرس ۱۶ بیتی به حافظه ROM داده می‌شود.

۳. حافظه ROM مقدار ۳۲ بیتی ذخیره‌شده در آن آدرس را به خروجی منتقل می‌کند.

۴. این مقدار به‌عنوان **دستورالعمل فعلی** به سایر بخش‌های پردازنده داده می‌شود تا پردازش شود.

ماژول ROM نمایش داده‌شده، حافظه‌ای فقط‌خواندنی با ظرفیت بالا برای ذخیره‌سازی دستورالعمل‌ها در پردازنده‌های ساده می‌باشد. این حافظه نقش کلیدی در فرآیند واکشی دستورالعمل (Instruction Fetch) دارد و توسط سیگنال آدرس ۱۶ بیتی هدایت می‌شود.

---

10. Last talk :

برای طراحی این مدار روز های زیادی وقت گذاشته شد و باعث شد به مهارت های ارزشمندی دست پیدا کنیم . در صورتی که دستور العمل پرش رو نداره اما بقیه اجزا و دستور العمل ها به درستی هر چه تمام تر کار میکنن این طراحی طبق استاندارد های خود معماری RISC-V طراحی شده است .

در طراحی این مدار هر چه که پیش روی میکردیم ایده های بیشتری به فکر میرسید به طور مثال تغییر control logic به control logic rom که در طراحی ان از یک حافظه ROM به جای مدارات و گیت هست با این حال با باگ های خود برنامه مجبور به استفاده یک گیت AND شدیم که برای فعال های RAM استفاده می شود .

در مرحله طراحی ALU ماژولی برای طراحی یک Full Adder قرار دادیم در طراحی این ماژول به این نتیجه رسیدیم که اگر این ماژول رو داخل مدار قرار بدیم سرعت بشدت پایین اومده و زمان اجرا خیلی زیاد می شد بنابراین از ماژول های اماده برنامه استفاده کردیم .

در مسیر طراحی این پردازنده با معماری RISC-V به باگ های زیادی برخورد کردیم از جمله باگ هایی که بشدت وقت گیر شد برای ما عمل نکردن دستور ذخیر سازی و پیاده سازی بود که مشکل از ادرس دهی RAM برای ذخیره سازی بود باگ های دیگه نحوی بودند یعنی اسم های تانل ها اشتباه بود .

مدار طراحی شده باعث شد تا برای تست ان یک اسمبلر کوچیک برای هر دو معماری RISC-V و MIPS طراحی کنیم تا با استفاده از اون کد های دستوری رو به کد Hexadecimal تبدیل و برای تست اماده کنیم در این برنامه مثال هایی هم هست برای رابط کاربری بهتر این برنامه تاجایی که شد به فاصله سه روز تا ارائه پروژه طراحی و بهینه سازی شد .

11. References