Abstract:

مسیر داده پردازنده با تغییر در مسیر داده ارائه شده در کلاس، طراحی شده است. دستورات جدید اضافه شده به پردازنده با رنگ‌های متفاوت نشان داده شده‌اند. این طراحی در ادامه به نمایش گذاشته شده است.
قالب دستورات و جدول مربوط به واحد کنترل نیز در ادامه آورده شده‌اند. برای تست پردازنده مذکور برنامه‌ای به زبان اسمبلی نوشته شده است که مقدار و اندیس بزرگ‌ترین عضو یک آرایه 20 عنصری را پیدا می‌کند و آن‌ها را در مموری ذخیره می‌کند. همچنین برای تبدیل برنامه اسمبلی ذکر شده به زبان ماشین، یک برنامه اسمبلر به زبان C++ نوشته شده که در فولدر Utils قرار داده شده است.
برای قرار دادن مقادیر آرایه در مموری از یک فایل به نام ArrayData.txt استفاده می‌شود که در فولدر Memory قرار دارد. در این فایل مقادیر به صورت Decimal و در 20 خط متوالی نوشته می‌شوند. این مقادیر در ادامه به کمک یک اسکریپت پایتون به مقادیر باینری 32 بیتی تبدیل می‌شوند که هر کدام از این مقادیر 32 بیتی به 4 عدد 8 بیتی جهت قرار گرفتن در مموری تقسیم می‌شوند. فایل FindMax.asm که کد اسمبلی برنامه تست پردازنده است که بالاتر ذکر شده بود، نیز در فولدر Memory قرار می‌گیرد. در نهایت با اجرای اسکریپت GenerateMemFiles.bat که در همین فولدر Memory قرار دارد، کد اسمبلی به اسمبلر ذکر شده داده می‌شود و خروجی آن که فایل instructions.txt است، در فولدر Memory ذخیره می‌شود. این فایل حاوی تعدادی دستور 32 بیتی به زبان ماشین است که هر کدام در یک خط قرار گرفته‌اند. در نهایت هر دو فایل instructions.txt و ArrayData.txt به اسکریپت‌های پایتون موجود در فولدر Utils داده می‌شوند که به اعداد 8 بیتی جهت قرار گرفتن در حافظه تقسیم شوند. فایل‌های نهایی که با نام‌های instructions_divided.mem و data.mem ایجاد شده‌اند، به وسیله اسکریپت اجرا شده در فولدر Verilog\Sim که محل ایجاد پروژه ModelSim است قرار می‌گیرند تا با اجرای شبیه‌سازی، توسط مموری خوانده شوند.
در ادامه به کمک شبیه‌سازی، درستی عملکرد پردازنده نشان داده می‌شود.

MIPS Instructions:

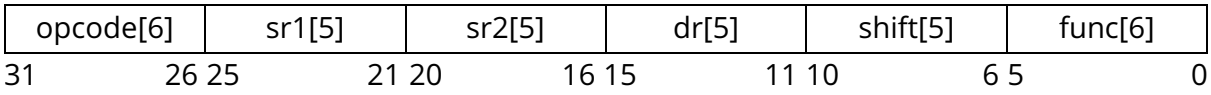
R-Type:

The opcode for all R-Type instructions is *000000* func:

add R1, R2, R3	R1 = R2 + R3	100000
sub R1, R2, R3	R1 = R2 - R3	100010
slt R1, R2, R3	R1 = (R2 < R3) ? 1 : 0	101010
and R1, R2, R3	R1 = R2 & R3	100100
or R1, R2, R3	R1 = R2 | R3	100101

Assembly: *inst dr, sr1, sr2*

Machine:



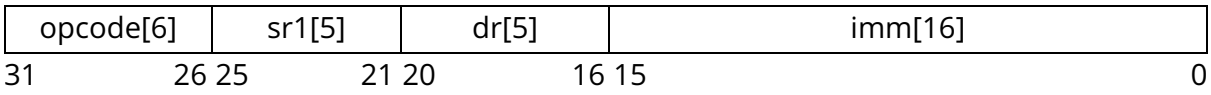
I-Type:

Num is an immediate value. opcode:

addi R1, R2, Num	R1 = R2 + Num	001000
slti R1, R2, Num	R1 = (R2 < Num) ? 1 : 0	001010

Assembly: *inst dr, sr1, imm*

Machine:



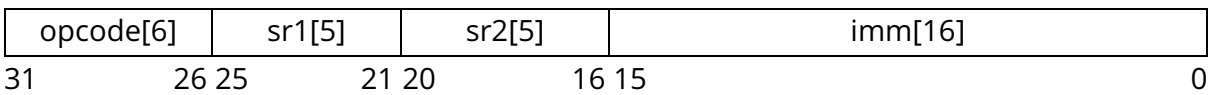
Mem-Type:

Num is an immediate value. opcode:

lw R1, Num(R2)	R1 = Mem[R2 + Num]	100011
sw R1, Num(R2)	Mem[R2 + Num] = R1	101011

Assembly: *inst sr2, imm(sr1)*

Machine:



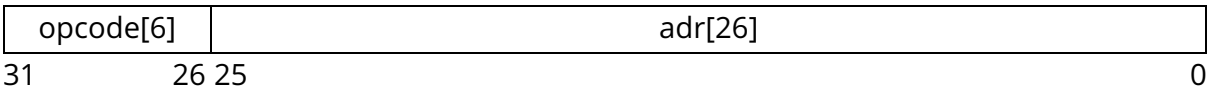
Jump1:

Adr is a label which turns into an address to jump to. opcode:

j Adr	PC = {(PC+4)[31:28], Adr << 2}	000010
jal Adr	j and R31 = PC+4	000011

Assembly: *inst adr* (label)

Machine:



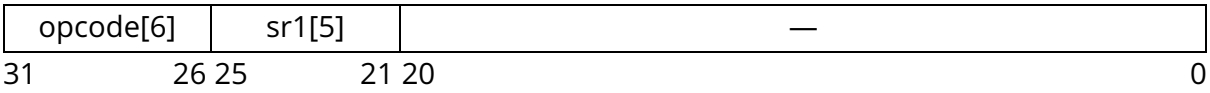
Jump2:

Jumps to the address stored in R1. opcode:

jr R1	PC = R1	111111

Assembly: *inst sr1*

Machine:



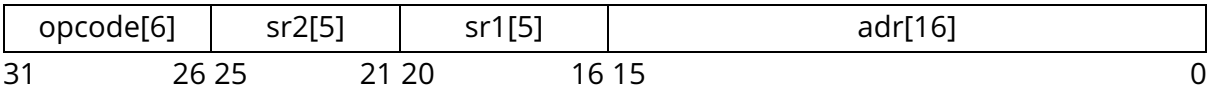
Branch:

Adr is how much the PC will change. (can be a label) opcode:

beq R1, R2, Adr	if R1==R2: PC = PC+4+Adr<<2	000100

Assembly: *inst sr1, sr2, adr* (address or label)

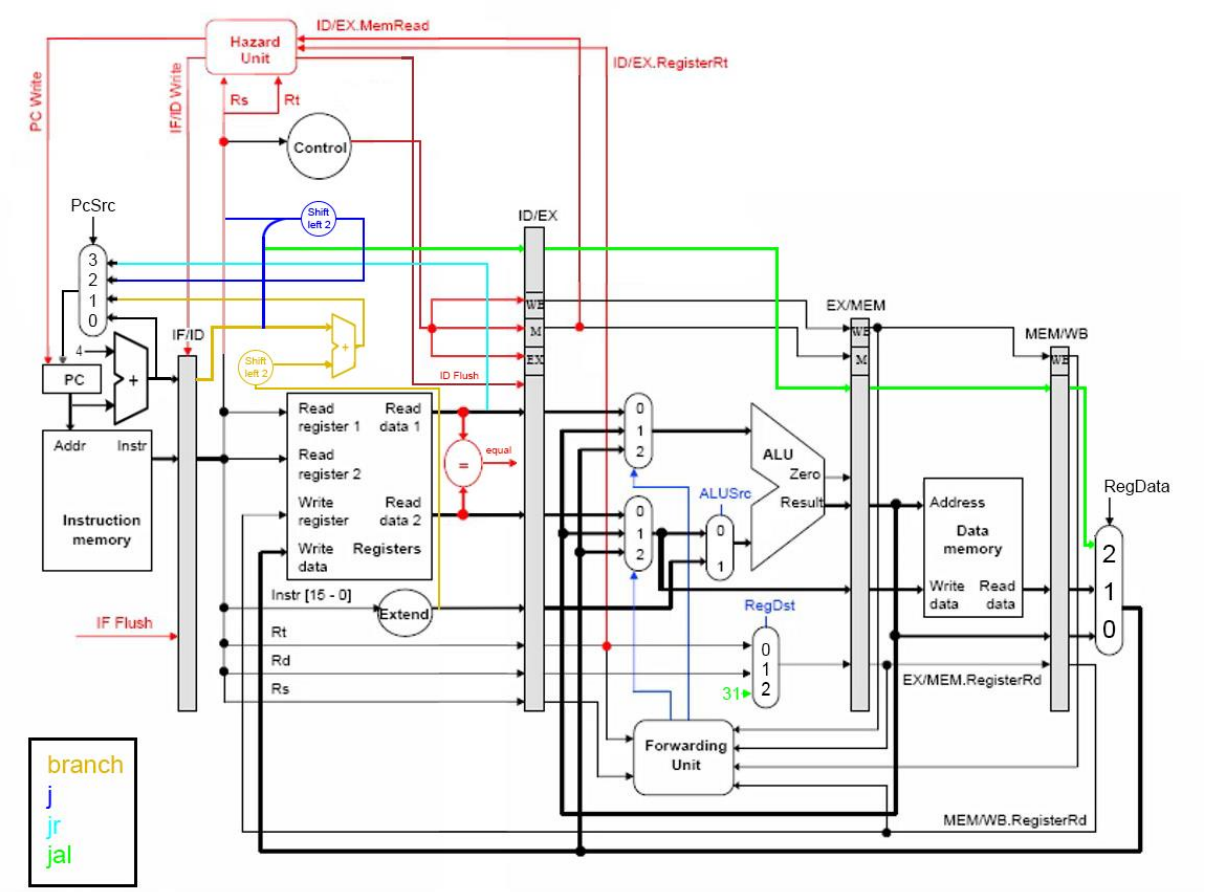
Machine:



CPU:

Datapath:

مسیر داده طراحی شده در کلاس برای اجرای دستورات جدید تغییر یافت و مسیرهای جدید با رنگ‌های متفاوتی مشخص شده‌اند:



Controller:

سیگنال های مسیر داده:

	PCSrc	IFID_Flush	ALUSrc	RegDst	MemRead	MemWrite	RegData	RegWrite
add	00	0	0	01	0	0	00	1
sub	00	0	0	01	0	0	00	1
and	00	0	0	01	0	0	00	1
or	00	0	0	01	0	0	00	1
slt	00	0	0	01	0	0	00	1
addi	00	0	1	00	0	0	00	1
slti	00	0	1	00	0	0	00	1
lw	00	0	1	00	1	0	01	1
sw	00	0	1	—	0	1	—	0
j	10	1	—	—	0	0	—	0
jal	10	1	—	10	0	0	10	1
jr	11	1	—	—	0	0	—	0
beq	01	ReadData1 == ReadData2	—	—	0	0	—	0
	IF		EX		MEM		WB	

جدول طراحی ALU:

ALU Operations	Opcode
+	010
-	110
AND	000
OR	001
slt	111

جدول خروجی کنترلر ALU (به ALU):

ALU Controller Function	ALUopc
100000	010 (+)
100010	110 (-)
100100	000 (&)
100101	001 (|)
101010	111 (slt)

جدول ورودی کنترلر ALU (از کنترلر اصلی):

CPU To ALU Controller	ALUop
+	00
-	01
slt	10
Function	11

Test Program:

Assembly:

کد اسمبلی برای پیدا کردن بزرگ‌ترین عضو آرایه‌ای 20 عنصری با آدرس شروع 1000:

```
j Main

FindMax:
    addi R1, R0, 1000          # i = 1000
    lw   R2, 0(R1)             # maxElement = mem[1000]
    addi R3, R0, 0             # maxIndex = 0
    addi R4, R0, 0             # idx = 0

    Loop:
        addi R1, R1, 4         # i += 4
        addi R4, R4, 1         # idx += 1
        slti R5, R4, 20        # check if 20 elements are traversed

        nop                    # branch-equal data hazard pipeline manual stall
        nop
        nop

        beq  R5, R0, EndLoop   # if 20 elements are traversed, jump to EndLoop
        lw   R6, 0(R1)         # element = mem[i]
        slt  R5, R2, R6        # check if element is greater than maxElement

        nop                    # branch-equal data hazard pipeline manual stall
        nop
        nop

        beq  R5, R0, Loop      # if element is not greater than maxElement, jump to Loop
        add  R2, R0, R6        # maxElement = element
        add  R3, R0, R4        # maxIndex = idx
        j    Loop              # jump to Loop

    EndLoop:
        sw R2, 2000(R0)        # mem[2000] = maxElement
        sw R3, 2004(R0)        # mem[2004] = maxIndex
        jr R31                 # return


Main:
    jal FindMax                # call FindMax
```

(nop یک pseudo-instruction است که به دستوری R-Type با آدرس رجیستر 0 (که نمی‌شود به آن write کرد) ترجمه می‌شود. مثلا اینجا در کد ماشین تبدیل به add R0, R0, R0 می‌شود)

Machine Code:

با توجه به معادل‌های اسمبلی و کدهای 32 بیتی قابل اجرا در بخش Instructions، یک برنامه assembler نوشته شد که خروجی آن برای کد بالا در تصویر زیر مشخص شده است:

```
1   00001000000000000000000000011000
2   00100000000000010000001111101000
3   10001100001000100000000000000000
4   00100000000000110000000000000000
5   00100000000001000000000000000000
6   00100000010000100000000000000100
7   00100000100001000000000000000001
8   00101000100001010000000000010100
9   00000000000000000000000000100000
10  00000000000000000000000000100000
11  00000000000000000000000000100000
12  00010000000001010000000000001001
13  10001100001001100000000000000000
14  00000000010001100010100000101010
15  00000000000000000000000000100000
16  00000000000000000000000000100000
17  00000000000000000000000000100000
18  00010000000001011111111111110011
19  00000000000001100001000000100000
20  00000000000001000001100000100000
21  00001000000000000000000000000101
22  10101100000000100000011111010000
23  10101100000000110000011111010100
24  11111111111000000000000000000000
25  00001100000000000000000000000001
```

Simulation Results:

Memory > ArrayData.txt

1	-34
2	-10
3	37
4	46
5	98
6	2
7	131
8	-982
9	143
10	8
11	56
12	36
13	-28
14	98
15	17
16	-7
17	0
18	2
19	5
20	91

Memory > data.mem

1	@3e8
2	11011110
3	11111111
4	11111111
5	11111111
6	@3ec
7	11110110
8	11111111
9	11111111
10	11111111
11	@3f0
12	00100101
13	00000000
14	00000000
15	00000000
16	@3f4
17	00101110
18	00000000
19	00000000
20	00000000

عناصر آرایه به صورت روبه‌رو انتخاب شده‌اند.
بزرگ‌ترین عضو آرایه 143 است که در اندیس شماره 8 قرار دارد.

این عناصر خانه 1000 تا 1079 حافظه را اشغال می‌کنند.
برای خواندن این مقادیر از فرمت mem. استفاده شده است.

نتیجه شبیه‌سازی:

MaxElement نتیجه combine شدن خانه‌های 2000 تا 2003 مموری است که مقدار 143 را نشان می‌دهد.
همچنین MaxIndex نتیجه combine شدن خانه‌های 2004 تا 2007 مموری است که مقدار 8 را نشان می‌دهد. با توجه به مقادیر Decimal آرایه که بالاتر نشان داده شد، عملکرد پردازنده صحیح است.