

## 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:

opcode[6]	sr1[5]	sr2[5]	dr[5]	shift[5]	func[6]
31	26 25	21 20	16 15	11 10	6 5
					0

### 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:

opcode[6]	sr1[5]	dr[5]	imm[16]
31	26 25	21 20	16 15
			0

### 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:

opcode[6]	sr1[5]	sr2[5]	imm[16]
31	26 25	21 20	16 15
			0

### 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:

opcode[6]	adr[26]
31	26 25 0

### Jump2:

Jumps to the address stored in R1. opcode:

jr	R1	PC = R1	111111
----	----	---------	--------

Assembly: *inst sr1*

Machine:

opcode[6]	sr1[5]	—
31	26 25	21 20 0

### 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)

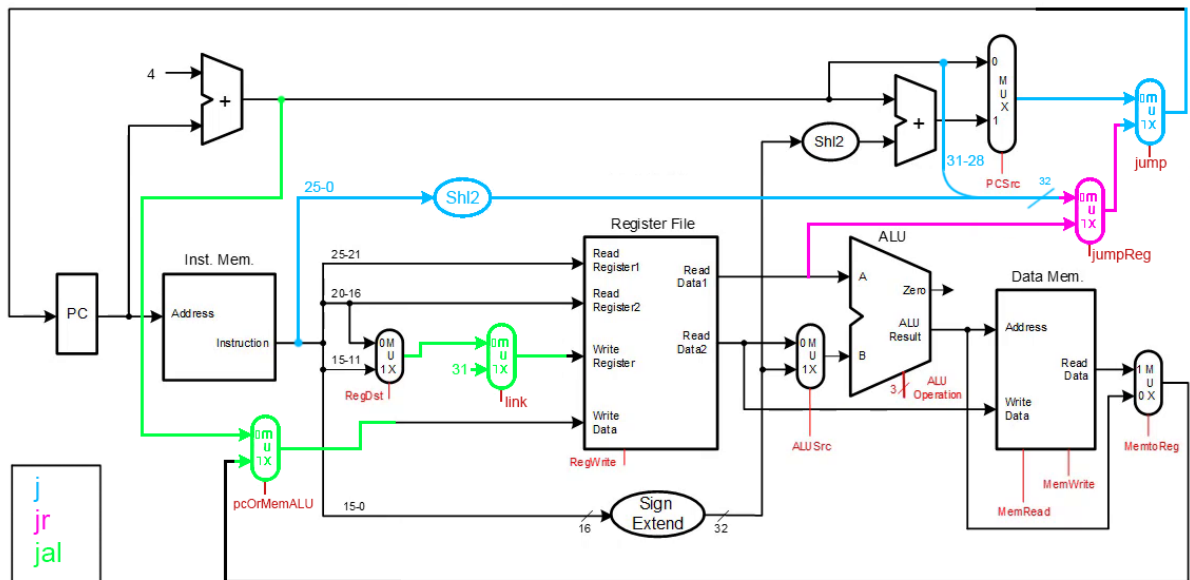
Machine:

opcode[6]	sr2[5]	sr1[5]	adr[16]
31	26 25	21 20	16 15 0

CPU:

### Datapath:

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



### Controller:

سیگنال‌های مسیر داده توسط کنترلر با توجه به opcode دستور فعلی، func در حالت r-type، و خروجی zero واحد ALU تعیین می‌شوند:

	regDst	regWrite	ALUSrc	ALUOp	memRead	memWrite	memToReg	PCSrc	jump	jumpReg	pcOrMemAlu	link
add	1	1	0	010 (+)	0	0	0	0	0	—	1	0
sub	1	1	0	110 (-)	0	0	0	0	0	—	1	0
slt	1	1	0	110 (-)	0	0	0	0	0	—	1	0
and	1	1	0	000 (&)	0	0	0	0	0	—	1	0
or	1	1	0	001 ( )	0	0	0	0	0	—	1	0
addi	0	1	1	010 (+)	0	0	0	0	0	—	1	0
slti	0	1	1	110 (-)	0	0	0	0	0	—	1	0
lw	0	1	1	010 (+)	1	0	1	0	0	—	1	0
sw	—	0	1	010 (+)	0	1	—	0	0	—	—	—
j	—	0	—	—	0	0	—	0	1	0	—	—
jal	—	1	—	—	0	0	—	0	1	0	0	1
jr	—	0	—	—	0	0	—	—	1	1	—	—
beq	—	0	0	110 (-)	0	0	—	zero	0	—	—	—

## Test Program:

### Assembly:

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

```
Memory > FindMax.asm
1  j Main
2
3  FindMax:
4      addi R1, R0, 1000          # i = 1000
5      lw R2, 0(R1)              # maxElement = mem[1000]
6      addi R3, R0, 0            # maxIndex = 0
7      addi R4, R0, 0            # idx = 0
8
9      Loop:
10     addi R1, R1, 4             # i += 4
11     addi R4, R4, 1             # idx += 1
12     slti R5, R4, 20            # check if 20 elements are traversed
13     beq R5, R0, EndLoop        # if 20 elements are traversed, jump to EndLoop
14     lw R6, 0(R1)              # element = mem[i]
15     slt R5, R2, R6             # check if element is greater than maxElement
16     beq R5, R0, Loop           # if element is not greater than maxElement, jump to Loop
17     add R2, R0, R6             # maxElement = element
18     add R3, R0, R4             # maxIndex = idx
19     j Loop                     # jump to Loop
20
21     EndLoop:
22     sw R2, 2000(R0)            # mem[2000] = maxElement
23     sw R3, 2004(R0)            # mem[2004] = maxIndex
24     jr R31                     # return
25
26
27 Main:
28     jal FindMax                # call FindMax
```

### 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:

```
Memory > instructions.mem
1  0000100000000000000000000000000010010
2  001000000000000010000000111101000
3  10001100001000100000000000000000
4  00100000000000110000000000000000
5  00100000000010000000000000000000
6  001000000010000100000000000000100
7  00100000100001000000000000000001
8  001010001000010100000000000010100
9  000100000000010100000000000000110
10 10001100001001100000000000000000
11 00000000010001100010100000101010
12 0001000000000101111111111111001
13 0000000000001100001000000100000
14 0000000000001000001100000100000
15 000010000000000000000000000000101
16 1010110000000010000001111101000
17 10101100000000110000011111010100
18 1111111111100000000000000000000
19 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 MaxIdx 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.

