

Abstract:

مسیر داده و واحد کنترل کننده در امتحان میان ترم دوم طراحی شده اند و در ابتدا قالب دستورات پردازنده آورده شده است.

برای تست پردازنده مذکور برنامه ای به زبان اسمبلی نوشته شده است که مقدار و اندیس بزرگ ترین عضو یک آرایه 20 عنصری را پیدا می کند و آن ها را در مموری ذخیره می کند. 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 می شود که در فولدر Memory قرار دارد. در این فایل مقادیر به صورت Decimal و در 20 خط متوالی نوشته می شوند. این مقادیر در ادامه به کمک یک اسکریپت پایتون به مقادیر باینری 16 بیتی تبدیل می شوند.

فایل FindMax.asm که کد اسمبلی برنامه تست پردازنده است توسط اسکریپت AssemblyGenerator.py که در فولدر Memory قرار دارد تولید می شود. (چون همانطور که جلوتر در بخش **Test Program** توضیح داده میشود، باید به ازای همه اعضای آرایه کد اسمبلی کپی شود)

در نهایت با اجرای اسکریپت GenerateMemFiles.bat که در همین فولدر Memory قرار دارد، کد اسمبلی به اسمبلر ذکر شده داده می شود و خروجی آن در فایل FindMax.machine ذخیره می شود. این فایل حاوی تعدادی دستور 16 بیتی به زبان ماشین است که هر کدام در یک خط قرار گرفته اند.

در نهایت هر دو فایل FindMax.machine و ArrayData.txt به اسکریپت های پایتون موجود در فولدر Utils داده می شوند که در فرمت mem. قرار گیرند. فایل نهایی data.mem ایجاد شده به وسیله اسکریپت اجرا شده در فولدر Verilog\Sim که محل ایجاد پروژه ModelSim است قرار می گیرد تا با اجرای شبیه سازی، توسط مموری خوانده شوند.

در ادامه به کمک شبیه سازی، درستی عملکرد پردازنده نشان داده می شود.

CPU Instructions:

A-Type:

Adr is an immediate value.

opcode:

load <i>Adr</i>	$R0 = \text{Mem}[\text{Adr}]$	0000
store <i>Adr</i>	$\text{Mem}[\text{Adr}] = R0$	0001
jump <i>Adr</i>	$\text{PC} = \text{Adr}$	0010

Assembly: *inst adr*

Machine:

opcode[4]	adr[12]
15	12 11 0

B-Type:

Adr is an immediate value.

opcode:

branch <i>Ri</i> , <i>Adr</i>	if $Ri == R0$: $\text{PC}[8:0] = \text{Adr}$	0100
-------------------------------	---	------

Assembly: *inst adr*

Machine:

opcode[4]	i[3]	adr[9]
15	12 11	9 8 0

C-Type:

The opcode for all C-Type instructions is 1000

func:

add <i>Ri</i>	$R0 = R0 + Ri$	000000100
sub <i>Ri</i>	$R0 = R0 - Ri$	000001000
and <i>Ri</i>	$R0 = R0 \& Ri$	000010000
or <i>Ri</i>	$R0 = R0 Ri$	000100000
not <i>Ri</i>	$R0 = \sim R0$	001000000
nop	No Operation	010000000
mvto <i>Ri</i>	$R0 = Ri$	000000001
mvfrom <i>Ri</i>	$Ri = R0$	000000010

Assembly: *inst Ri*

Machine:

opcode[4]	i[3]	func[9]
15	12 11	9 8 0

Num is an immediate value.

opcode:

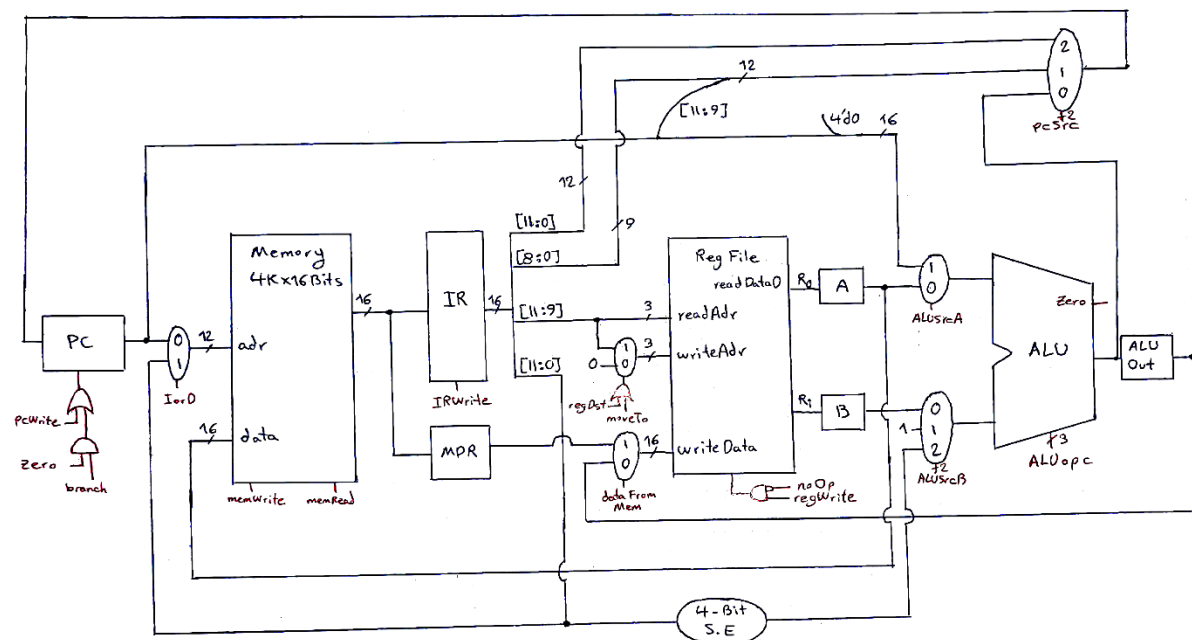
Assembly: *inst imm*

Machine:

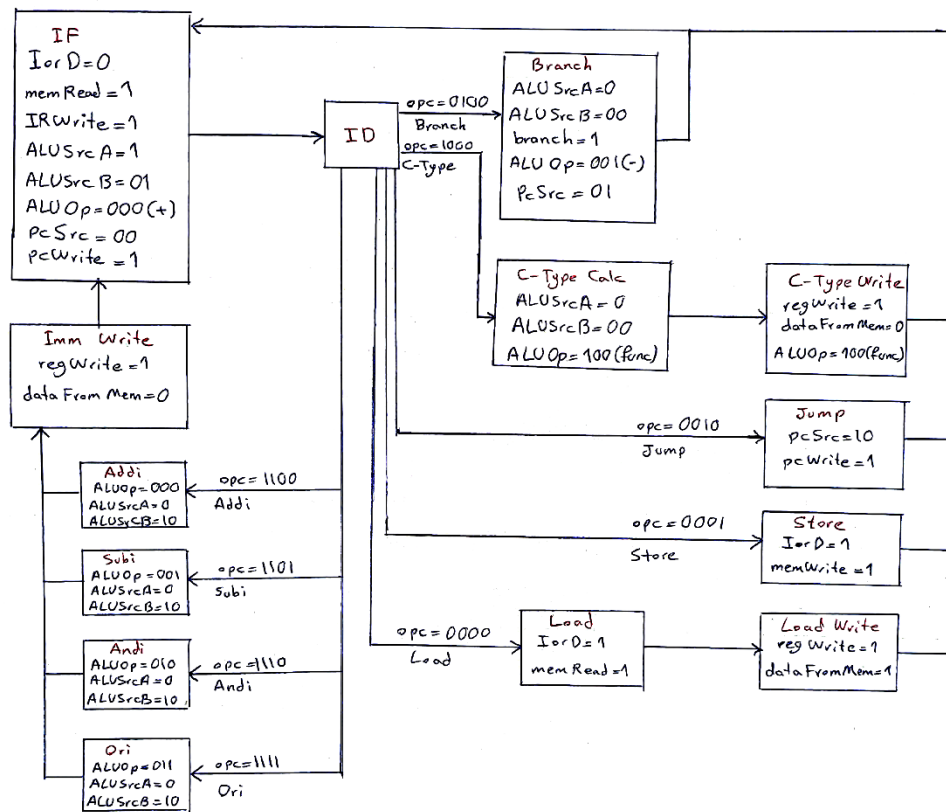
opcode[4]	imm[12]
15 12	11 0

CPU:

Datapath:



Controller:



جدول زیر مقادیر سیگنال های خروجی واحد کنترل کننده را در هر state نشان می دهد:

	memWrite	memRead	lorD	IRWrite	regDst	dataFromMem	regWrite	ALUSrcA	ALUSrcB	ALUOp	PcSrc	PcWrite	branch
Instruction Fetch (IF)	0	1	0	1	—	—	0	1 (PC)	01 ('d1)	000 (+)	00 (ALU)	1	0
Instruction Decode (ID)	0	0	—	0	—	—	0	—	—	—	—	0	0
Branch	0	0	—	0	—	—	0	0 (RO)	00 (Ri)	001 (-)	01 (Branch)	—	1
C-Type Calc	0	0	—	0	—	—	0	0 (RO)	00 (Ri)	100 (func)	—	0	0
C-Type Write	0	0	—	0	0 (RO)	0	1	—	—	100 (func)	—	0	0
Jump	0	0	—	0	—	—	0	—	—	—	10 (Jump)	1	0
Store	1	0	1	0	—	—	0	—	—	—	—	0	0
Load	0	1	1	0	—	—	0	—	—	—	—	0	0
Load Write	0	0	—	0	0 (RO)	1	1	—	—	—	—	0	0
Addi	0	0	—	0	—	—	0	0 (RO)	10 (imm)	000 (+)	—	0	0
Subi	0	0	—	0	—	—	0	0 (RO)	10 (imm)	001 (-)	—	0	0
Andi	0	0	—	0	—	—	0	0 (RO)	10 (imm)	010 (&)	—	0	0
Ori	0	0	—	0	—	—	0	0 (RO)	10 (imm)	011 (l)	—	0	0
Imm Write	0	0	—	0	0 (RO)	0	1	—	—	—	—	0	0

جدول طراحی ALU:

ALU Operations	Opcode	Output
+	000	A + B
-	001	A - B
AND	010	A & B
OR	011	A B
NOT	100	~B
PassInput1	101	A
PassInput2	110	B

جدول خروجی کنترلر ALU (به ALU):

ALU Controller Function	ALUopc	noOp	moveTo
000000100	000 (+)	0	0
000001000	001 (-)	0	0
000010000	010 (&)	0	0
000100000	011 ()	0	0
001000000	100 (~)	0	0
000000001	101 (Pass1 R0)	0	1
000000010	110 (Pass2 Ri)	0	0
010000000	101 (Pass)	1	0

جدول ورودی کنترلر ALU (از کنترلر اصلی):

CPU To ALU Controller	ALUop
+	000
-	001
AND	010
OR	011
Function	100

Test Program:

Assembly:

از آنجا که این CPU دستوری برای دسترسی به حافظه با مقدار رجیستر ندارد و فقط مقادیر ثابت را قبول می‌کند، نمیتوانیم حلقه for بنویسیم و باید کد اسمبلی برای چرخه را 20 بار (با تغییر آدرس حافظه) تکرار کنیم. این کار تعداد دستورها را افزایش می‌دهد و از 200 دستور بیشتر خواهد شد. پس شروع عناصر آرایه را به خانه 300 و مقدار بزرگترین عنصر و اندیس آن را به ترتیب در خانه‌های 400 و 404 حافظه قرار می‌دهیم. کد اسمبلی برای پیدا کردن بزرگ‌ترین عضو آرایه‌ای 20 عنصری:

```
1  andi 0          # R0 = 0
2  mvto R7         # R7 = 0 (const)
3  addi 1024       # R0 = 1024
4  add R0          # R0 = 2048
5  add R0          # R0 = 4096
6  add R0          # R0 = 8192
7  add R0          # R0 = 16384
8  add R0          # R0 = 32768
9  mvto R6         # R6 = 0x8000 (const)
10
11 load 300        # R0 = M[300]
12 mvto R1         # R1 = M[300] (max)
13 andi 0          # R0 = 0
14 mvto R2         # R2 = 0 (maxIdx)
15
16 # start loop
17
18 load 301        # R0 = M[301]
19 mvto R4         # R4 = M[301] (temp)
20 sub R1          # R0 = temp - max
21 and R6          # R0 & 0x8000 (keep MSB)
22 branch R7, gt01 # R0 == 0
23 jump end01
24
25 gt01:
26   mvfrom R4     # R0 = temp
27   mvto R1       # max = temp
28   andi 0        # R0 = 0
29   addi 01       # R0 = 1
30   mvto R2       # R2 = 1 (maxIdx)
31
32 end01:
33
34 # end loop
35
36 ... # repeat (start-end) 19 times
37
38
39 mvfrom R1       # R0 = max
40 store 400       # M[400] = max
41 mvfrom R2       # R0 = maxIdx
42 store 404       # M[404] = maxIdx
```

توضیح مقایسه کردن دو مقدار در اسمبلی:

برای اینکه دو عنصر را مقایسه کنیم، از آنجا که دستوری برای این کار نداریم، ابتدا دو عنصر را از هم کم می‌کنیم، و سپس از MSB برای تشخیص منفی بودن حاصل تفریق استفاده می‌کنیم. از اینجا متوجه می‌شویم که عدد بزرگ‌تر است یا خیر.

در ابتدای اسمبلی عدد 32768 که در باینری معادل 1000000000000000 است (1 با 15 تا 0) را ذخیره می‌کنیم. از این عدد برای mask کردن MSB استفاده می‌کنیم.

برای مقایسه، ابتدا دو عدد را از هم کم می‌کنیم (a-b)، سپس با عدد بالا and می‌کنیم، و در آخر با عدد 0 مقایسه می‌کنیم (این کار با دستور branch ممکن است). اگر عدد 0 بود، یعنی a-b مثبت بوده و در نتیجه $a > b$ است.

Machine Code:

با توجه به معادل‌های اسمبلی و کدهای 16 بیتی قابل اجرا در بخش **Instructions**، یک برنامه assembler نوشته شد که بخشی از خروجی آن برای کد بالا در تصویر زیر مشخص شده است:

```
1 1110000000000000
2 1000111000000001
3 1100010000000000
4 1000000000000100
5 1000000000000100
6 1000000000000100
7 1000000000000100
8 1000000000000100
9 1000110000000001
10 0000000100101100
11 1000001000000001
12 1110000000000000
13 1000010000000001
14 0000000100101101
15 1000100000000001
16 1000001000001000
17 1000110000010000
18 0100111000010011
19 0010000000011000
20 1000100000000010
21 1000001000000001
22 1110000000000000
23 1100000000000001
24 1000010000000001
25 0000000100101110
26 1000100000000001
27 1000001000001000
28 1000110000010000
```

Simulation Results:

Memory > ArrayData.txt	Memory > data.mem
1 -34	1 // Array Data
2 -10	2
3 37	3 @12c
4 46	4 1111111111011110
5 98	5 1111111111101110
6 2	6 00000000100101
7 131	7 00000000101110
8 -982	8 00000000110010
9 143	9 00000000000010
10 8	10 00000010000011
11 56	11 11111000101010
12 36	12 00000001000111
13 -28	13 00000000001000
14 98	14 00000000011000
15 17	15 00000000100100
16 -7	16 11111111100100
17 0	17 00000000110010
18 2	18 00000000010001
19 5	19 11111111111001
20 91	20 00000000000000
	21 00000000000010
	22 00000000000010
	23 00000000101101
	24
	25
	26 // Instructions
	27
	28 0000
	29 1110000000000000
	30 1000111000000001
	31 1100010000000000
	32 1000000000000000

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

این عناصر خانه 300 تا 319 حافظه را اشغال می‌کنند.
برای خواندن این مقادیر از فرمت mem. استفاده شده است.
ابتدا مقادیر آرایه از خانه 300 نوشته شده‌اند و سپس دستورها از خانه 0 شروع می‌شوند.

نتیجه شبیه‌سازی:

Max Element خانه 400 مموری است که مقدار 143 را نشان می‌دهد.
همچنین Max Index خانه 404 مموری است که مقدار 8 را نشان می‌دهد.
با توجه به مقادیر Decimal آرایه که بالاتر نشان داده شد، عملکرد پردازنده صحیح است.
در طی محاسبات، R1 عنصر بیشینه و R2 هم اندیس آن را نگه می‌دارد.

