

گزارش پروژه۲ درس سیستم های دیجیتال

اعضای گروه: سینا طبسی (۸۱۰۱۹۹۵۵۴) سیدحامد میرامیرخانی (۸۱۰۱۹۹۵۰۰)

دستورات جدید اضافه شده

دستور j :

دستور	کاری که انجام میدهد	آپکود
J adr	PC = {(PC+4)[31:28], adr<<2}	000010

از ۳۲ بیتی که بعنوان دستور دریافت میکنیم بیت های ۲۵ تا ۰ مربوط به آدرس پرش ما میشود.

در دیتاپث این آدرس باید در سایکل بعد بعنوان مقدار PC باشد. اما آدرس باید ۳۲ بیت باشد! پس باید این آدرس را ابتدا ۲بیت به چپ و درنهایت ۴بیت PC سابق را در سمت چپ آن کانکت کنیم تا درنهایت یک آدرس ۲۴+۲+۴+۳۳ بیتی در PC داشته باشیم. همانطور که در شکل دیتاپث نهایی که در ادامه مشاهده میکنید این مسیر با رنگ بنفش مشخص شده است.

در کنترلر هم باید سلکتور های mux هایی که سرراه این آدرس است را طبق مقادیر آن انتخاب کنیم پس jsel = 0 شود.

دستور jal:

دستور	کاری که انجام میدهد	آپکود
Jal adr	J and R31 = PC+4	000011

ساختار دستور مثل j است. یعنی ۶ بیت سمت چپ دستور را مشخص میکند و بقیه بیت ها مختص آدرس خواهد بود.

تغییرات در دیتاپث با رنگ قهوه ای مشخص شده اند. آدرس اکستند شده بعد از دوبار شیفت خوردن و بعد از یک سایکل در PC قرار میگیرد. مقدار بعدی PC در همین مرحله باید مشخص شود تا دوبار به آدرس بازگردد. برای همین باید در R31 مقدار PC+4 نوشته شود.

در كنترلر هم بايد regWrite=1 و pcSel=0 و pcSel=1 شود.

دستور jr:

دستور	کاری که انجام میدهد	آپکود
Jr Ri	PC = Ri	000110

ع بیت سمت چپ دستور را مشخص میکند، $\overline{\alpha}$ بیت بعدی یعنی R25 تا R21 مقدار رجیستر را مشخص میکند بقیه بیت ها هم برای ما فاقد اهمیت است.

تغییرات در دیتایث با رنگ زرد مشخص شده اند. مشخص است چه اتفاقی می افتد، Ri در PC ذخیره میشود.

در کنترلر هم باید pcSel=1 و Jsel=1 شود.

دستور slti :

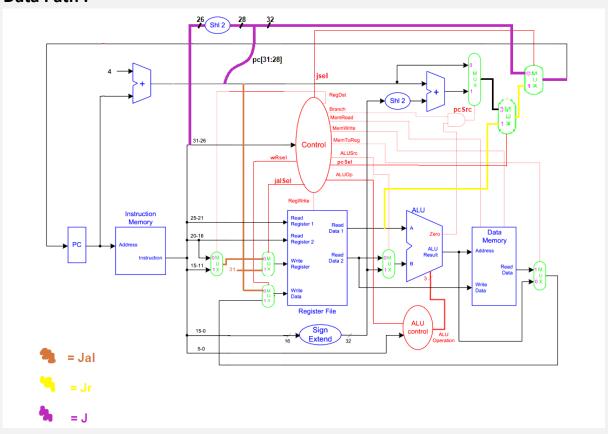
دستور	کاری که انجام میدهد	آپکود
Jr R1	Ri = (Rj < Num) ? 1 : 0	001010

۶ بیت سمت چپ دستور را مشخص میکند، ۵ بیت بعدی یعنی R25 تا R21 مقدار رجیستر اول و ۵ بیت یعدی یعنی R20 تا R16 رجیستر دوم، عدد ثابت هم در ۱۶ بیت بعدی است.

در دیتاپث نیازی نیست تغییر ی ایجاد کنیم.

در كنترلر هم بايد Jsel=1 و Jsel=1 و wRsel=1 و egWrite =1 شود و بقيه سيگنال ها O.

Data Path:



Controler:

aluOp	pcSel	aluSrc	memToReg	memWrite	memRead	pcSrc	jalSel	wRsel	jsel	RegWrite	RegDst	
010(+)	0	0	0	0	0	0	0	1	1	1	1	add
110(-)	0	0	0	0	0	0	0	1	1	1	1	Sub
111(<)	0	0	0	0	0	0	0	1	1	1	1	Slt
000(&)	0	0	0	0	0	0	0	1	1	1	1	And
001()	0	0	0	0	0	0	0	1	1	1	1	Or
010(+)	0	1	0	0	0	0	0	1	1	1	0	Addi
110(-)	0	0	-	0	0	1	-	-	1	0	1	Beq
010(+)	0	1	1	0	1	0	0	1	1	1	0	Lw
101(+)	0	1	-	1	0	0	-	-	1	0	-	Sw
-	-	-	-	0	0	0	-	-	0	0	-	j
-	-	-	-	0	0	0	1	0	0	1	-	Jal
-	1	-	-	0	0	0	-	-	1	0	-	jr
111(<)	0	1	0	0	0	0	0	1	1	1	0	Slti

خواندن دستورات و تست کردن صحت تمام دستورات

برای بررسی برنامه زیر اجرا میشود که هدف آن این است که یک آرایه ۲۰ تایی از عناصر را بگیرد (مکان این عناصر از خانه ۱۰۰۰ حافظه شروع میشود) و در نهایت کوچکترین عنصر آنها را بازگرداند.

```
i main
     mipsFunc:
         addi R1, R0, 1000
         lw R2, 0(R1)
         addi R3, R0, 0
         addi R4, R0, 0
9
         Loop:
10
             addi R1, R1, 4
11
             addi R4, R4, 1
12
             slti R5, R4, 20
13
             beq R5, R0, EndLoop
14
             lw, R6, 0(R1)
15
             slt R5, R6, R2
16
             beq R5, R0, Loop
17
             add R2, R0, R6
18
             add R3, R0, R4
19
             add R3, R0, R4
20
             j Loop
21
22
         EndLoop:
23
             sw R2, 2000(R0)
24
             sw R3, 2004(R0)
25
             jr R31
26
27
     main:
28
         jal mipsFunc
              کد اسمیلی
```

```
case (opcode)
 // RType instructions
 6'b000000 : {reg dst, reg write, alu op
 // Load Word (lw) instruction
 6'b100011 : {alu_src, mem_to_reg, reg_wr
 // Store Word (sw) instruction
 6'b101011 : {alu src, mem write, Jsel} =
 // Branch on equal (beq) instruction
 6'b000100 : {branch, alu op, Jsel} = 4'b1
 // Add immediate (addi) instruction
 6'b001001: {reg write, alu src, Jsel, wDse
 //Jump
 6'b000010: ;
 //Jump And Link
 6'b000011: {reg_write, JalSel} = 2'b11;
 //Jump Register
 6'b000110: {Jsel,pcSel} = 2'b11;
 // Set on less than immediate (slti)
  6'b001010: {reg_write, Jsel, wDsel, alu_src
     آیکود های مربوط به هر دستور
```

پس در واقع لیست دستورات ما (برای فهم بهتر سمت راست هر خط دستور، شکل اسمبلی آن نیز نوشته شده) به این صورت خواهد بود:

_	15,02		- 11
ð	. 18	main	
0 - 0000 loke 2000 0000 0000			
alli Ro Ri	loss	(mipsFunc:)	_
- an la oles		addi Ri Ro	- f ana-
P I W RI R2	0	1 8 /6	
2 [1	**** **** ****	Lw R2, 0(F	1)
addi Ro R3		11: 0 0	
3 - [00 10	**** **** ****	addi Ra, Ro,	
addi Ro R4	•	addi Ry, Ro,	
	**** **** ****	The second secon	
loop; alli R. R.	4	[Loop:] addi R, R	-4
20010 0100 0010 0001		addi Nj.	19
addi R4 R4		addi Ry, Rs	
Sala alle light also are a	*** *** ***	10000 194 714	17
Slti R4 R5	20	slti R5, R4	-20
beq Ro R5	+6	beg R5, Ro	Enollo
		3,0	, .
9 1 He R6		IW R6,0(R.)
	sk slt		
Rtype R6 R2 R5	rec sele lele	SIT R5, R6	,R2
beg Ro R5	-6		
- 1 11		beg R5, Ro.	Loop
	sh add		.
Rotype Ro Ro Kz		add R2, Ro	,R6
	sh add	8	4
Rtype Ro R4 R3		add R3, Ro	, R4
<i>ā</i> 5			
9 100 100 100 200 200 200		J loop	
Sw Ro R2 2	1000	Endloop	
-late He and pole tore all		Sw R2, 2000	(Ro)
SW R. R3	2004		
flata 11	Hel elee	5w R3, 200	4(Ro)
Jr 31 -			
(1.)		Jr R31	
1760	14		
Jal 1		main	
Jal I		main de mips Fun	c(1)

حالا به حافظه دقت کنید و دیتاهایی که در آن وجود دارد، همانطور که گفته شد از خانه ۲۰۰۰ شروع میکنیم (3e8 معادل هگزادسیمال ۲۰۰۰ است).

1	@3e8
2	01000101
3	00000000
4	00000000
5	00000000
6	@3ec
7	01100101
8	00000000
9	00000000
10	00000000
11	@3 f 0
12	10101100
13	11111111
14	11111111
15	11111111
16	@3f4
17	01100110
18	00000000
19	00000000
20	00000000
21	@3f8
22	00000011
23	00000000
24	00000000
25	00000000
26	@3fc
27	00000000
28	00000000
29	00000000
30	00000000

31	@400
32	11111100
33	11111111
34	11111111
35	11111111
36	@404
37	00001111
38	00000000
39	00000000
40	00000000
41	@408
42	01010101
43	00000000
44	0000000
45	0000000
46	@40c
47	11100010
48	11111111
49	11111111
50	11111111
51	@410
52	00001101
53	00000000
54	0000000
55	00000000
56	@414
57	01111011
58	00000000
59	99999999

61	@418
62	10100001
63	11111111
64	11111111
65	11111111
66	@41c
67	10001111
68	11111111
69	11111111
70	11111111
71	@420
72	11110001
73	11111111
74	11111111
75	11111111
76	@424
77	00010111
78	00000000
79	00000000
80	00000000

81	@428
82	01001100
83	00000000
84	00000000
85	00000000
86	@42c
87	01001100
88	00000000
89	00000000
90	00000000
91	@430
92	10111011
93	11111111
94	11111111
95	11111111
96	@434
97	00010101
98	00000000
99	00000000
100	00000000

در فایل dataMem.mem این موارد نوشته شده است.

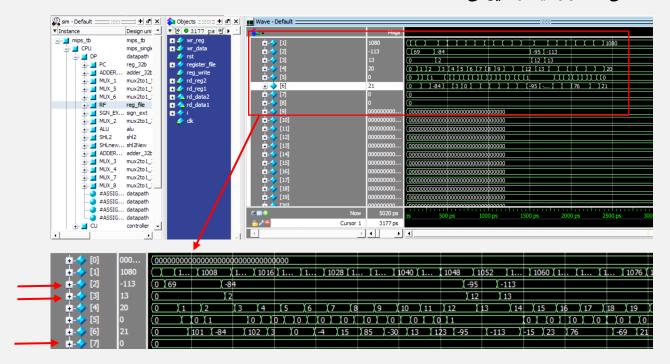
```
1 69
2 101
3 -84
4 102
5 3
6 0
7 -4
8 15
9 85
10 -30
```

```
13
11
    123
12
    -95
13
14
    -113
    -15
15
16
     23
     76
17
     76
18
19
     -69
20
     21
```

00000000

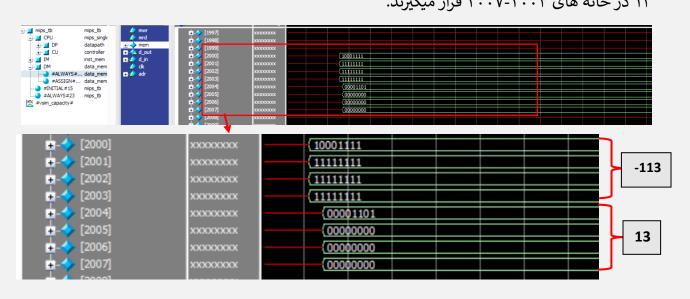
محتویات این ۲۰ عنصر حافظه هم مطابق رو به روست. در فایل arrayData.txt نوشته شده است.

اتفاقاتی که در رجیستر فایل می افتد:

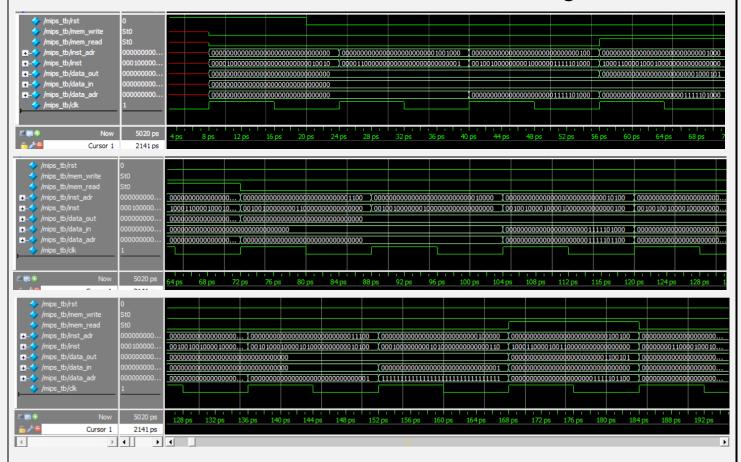


همانطور که مشاهده میکنید عناصر آرایه پیمایش میشوند و هربار چک میشود آیا مقدار R6 از R2 کوچکتر است یا خیر، اگر بود مقدار R2 آپدیت میشود و اندیس متناظر با مقدار جدید در R3 ثبت میشود.

در نهایت کوچکترین عنصر آرایه در خانه ۲۰۰۰ حافظه ذخیره میشود و مقدار اندیس آن هم در خانه ۲۰۰۴ حافظه. یعنی در این مثال عدد 113- در خانه های ۲۰۰۳-۲۰۰۳ حافظه و اندیس آن یعنی ۱۳ در خانه های ۲۰۰۴-۲۰۰۳ قرار میگیرند.



خروجي تست بنچ بصورت کلي:



بهمین صورت تا انتها دستورات مختلف (برای درک بهتر به کد اسمبلی که پیشتر گفته شد دقت کنید اجرا میشوند.

نمای کلی خروجی مادلسیم بصورت زیر خواهد بود:

