گزارش پروژه۲ درس سیستم های دیجیتال

اعضای گروه:

سینا طبسی (۸۱۰۱۹۹۵۵۴)

سیدحامد میرامیرخانی (۸۱۰۱۹۹۵۰۰)

دستورات جدید اضافه شده

دستور j :

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

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 ۲۶+۲+۴=۳۲ 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 :

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

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 کنترلر هم باید regWrite=1 و pcSrc=0 و pcSel=0 و Jsel=1 شود.

دستور jr :

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

۶ بیت سمت چپ دستور را مشخص میکند، ۵ بیت بعدی یعنی R25 تا R21 مقدار رجیستر را مشخص میکند بقیه بیت ها هم برای ما فاقد اهمیت است.

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

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

دستور slti :

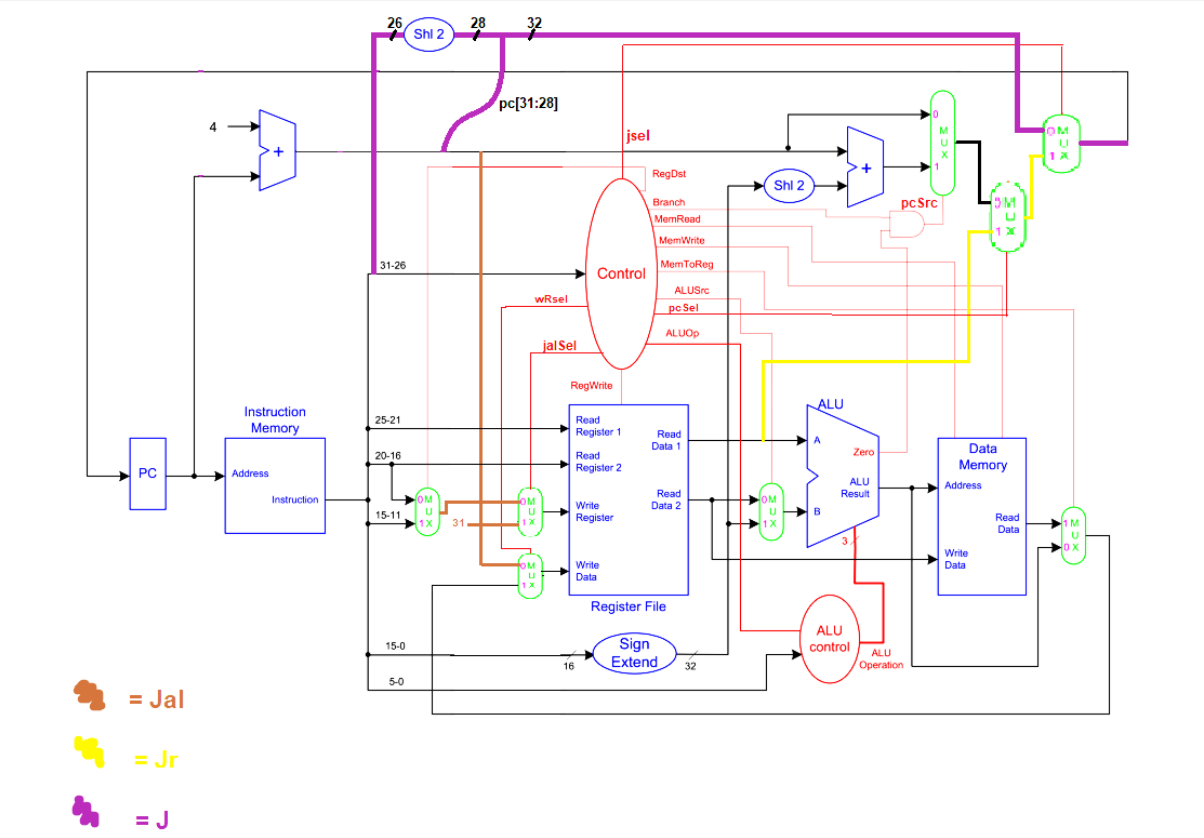
آپکود	کاری که انجام میدهد	دستور
001010	**Ri = (Rj < Num) ? 1 : 0**	**Jr R1**

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

در دیتاپث نیازی نیست تغییر ی ایجاد کنیم.

در کنترلر هم باید aluSrc=1 و Jsel=1 و wRsel=1 و regWrite =1 شود و بقیه سیگنال ها 0.

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.

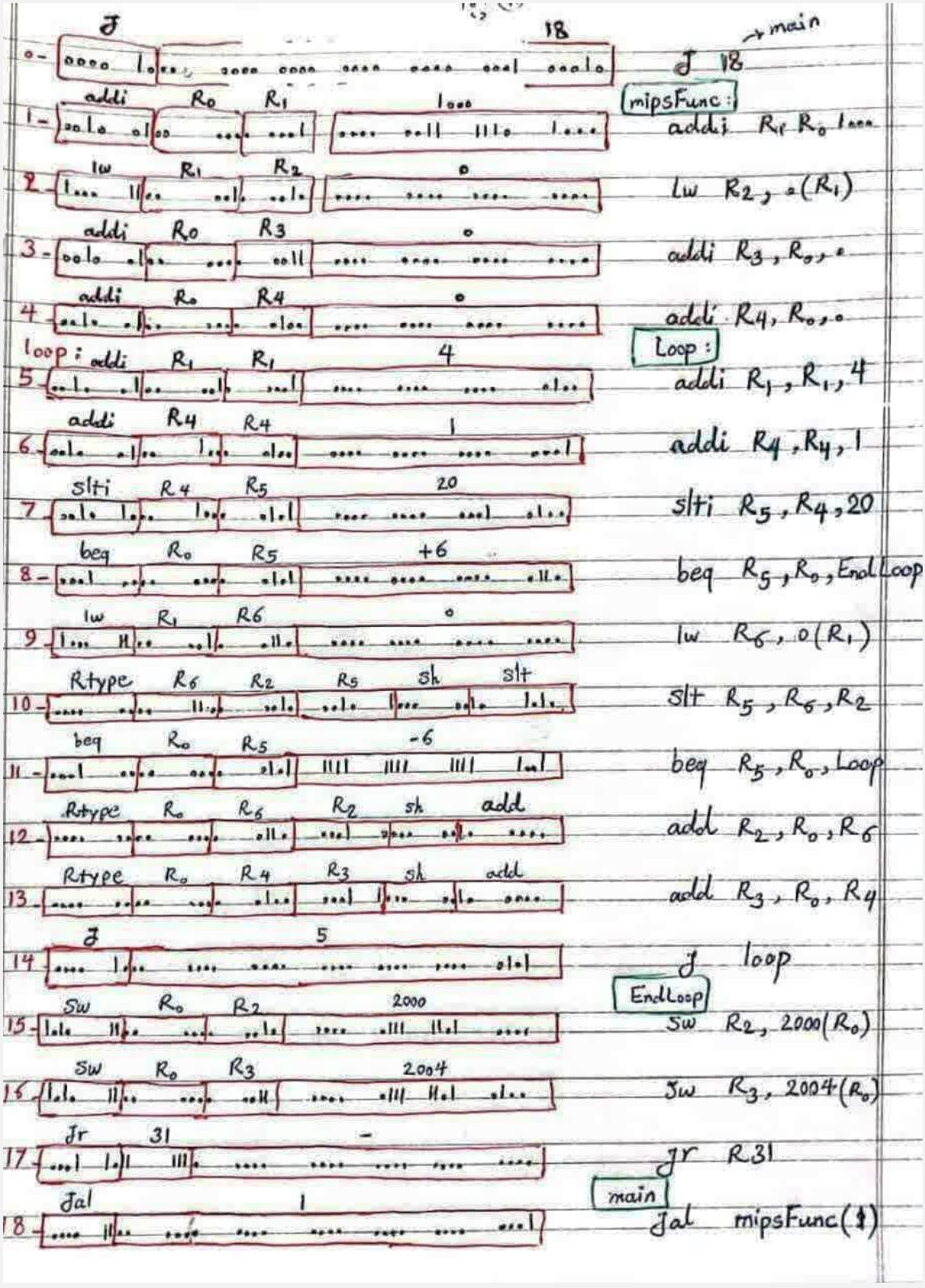
```
j main

mipsFunc:
    addi R1, R0, 1000
    lw R2, 0(R1)
    addi R3, R0, 0
    addi R4, R0, 0

    Loop:
        addi R1, R1, 4
        addi R4, R4, 1
        slti R5, R4, 20
        beq R5, R0, EndLoop
        lw, R6, 0(R1)
        slt R5, R6, R2
        beq R5, R0, Loop
        add R2, R0, R6
        add R3, R0, R4
        add R3, R0, R4
        j Loop

    EndLoop:
        sw R2, 2000(R0)
        sw R3, 2004(R0)
        jr R31

main:
    jal mipsFunc
```

کد اسمبلی

```
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
```

آپکود 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 :



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).

```
@3e8
01000101
00000000
00000000
00000000
@3ec
01100101
00000000
00000000
00000000
@3f0
10101100
11111111
11111111
11111111
@3f4
01100110
00000000
00000000
00000000
@3f8
00000011
00000000
00000000
00000000
@3fc
00000000
00000000
00000000
00000000
```

```
@400
11111100
11111111
11111111
11111111
@404
00001111
00000000
00000000
00000000
@408
01010101
00000000
00000000
00000000
@40c
11100010
11111111
11111111
11111111
@410
00001101
00000000
00000000
00000000
@414
01111011
00000000
00000000
00000000
```

```
@418
10100001
11111111
11111111
11111111
@41c
10001111
11111111
11111111
11111111
@420
11110001
11111111
11111111
11111111
@424
00010111
00000000
00000000
00000000
```

```
@428
01001100
00000000
00000000
00000000
@42c
01001100
00000000
00000000
00000000
@430
10111011
11111111
11111111
11111111
@434
00010101
00000000
00000000
00000000
```

در فایل
dataMem.mem
این موارد نوشته شده است.

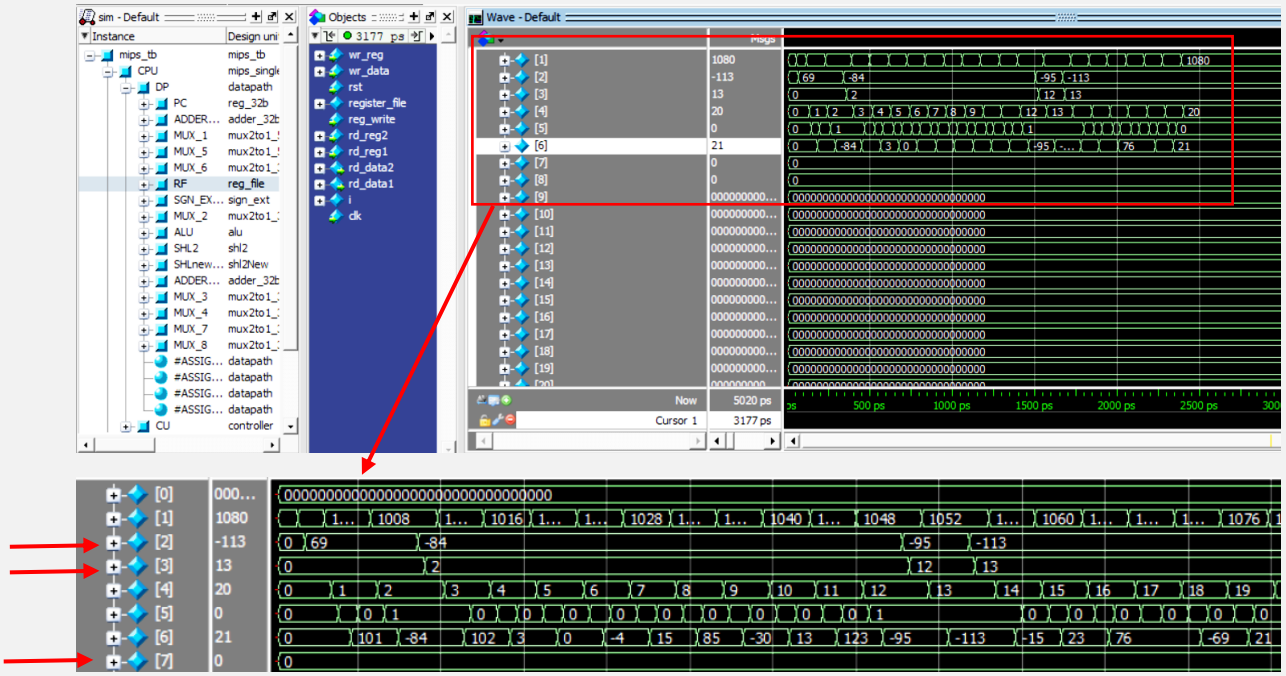
```
69
101
-84
102
3
0
-4
15
85
-30
```

```
13
123
-95
-113
-15
23
76
76
-69
21
```

محتویات این ۲۰ عنصر حافظه هم مطابق رو به روست.

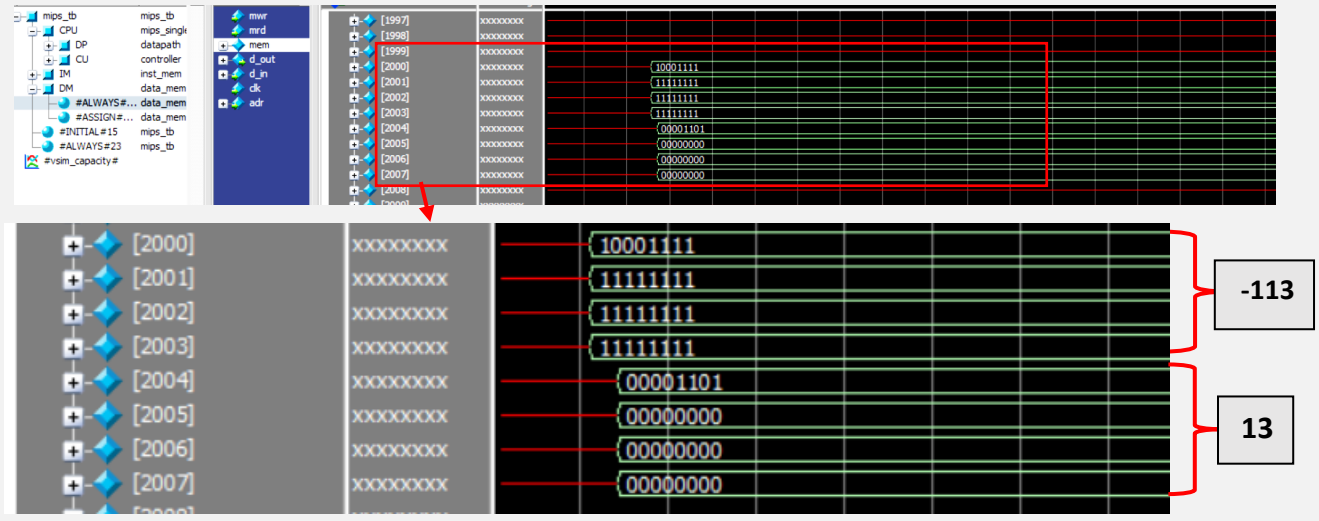
در فایل **arrayData.txt** نوشته شده است.

اتفاقاتی که در رجیستر فایل می افتد :

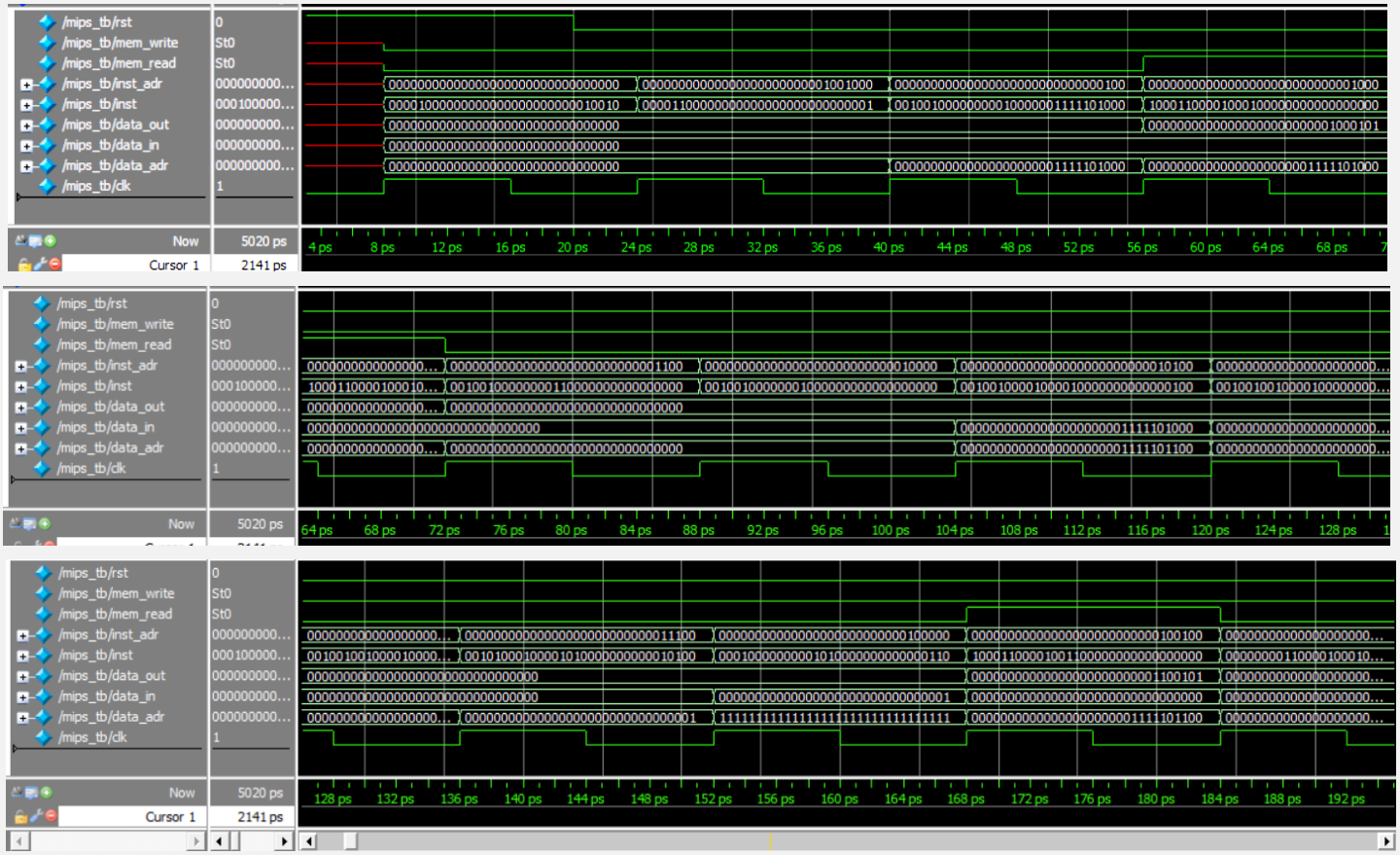


همانطور که مشاهده میکنید عناصر آرایه پیمایش میشوند و هربار چک میشود آیا مقدار R6 از R2 کوچکتر است یا خیر، اگر بود مقدار R2 آپدیت میشود و اندیس متناظر با مقدار جدید در R3 ثبت میشود.

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 ۲۰۰۰-۲۰۰۳ حافظه و اندیس آن یعنی ۱۳ در خانه های ۲۰۰۴-۲۰۰۷ قرار میگیرند.



خروجی تست بنچ بصورت کلی :



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

نمای کلی خروجی مادلسیم بصورت زیر خواهد بود :