

2019

# گزارش پروژه معماری کامپیووتر

علیرضا علیدوستی ۹۶۱۴۳۰۱۴۷

محمد مهدی محققولی ۹۶۱۴۳۰۶۱

## بسمه تعالی

### مقدمه

با مطالعه صورت پروژه و منبع استالینگز دریافتیم که باید یک پردازنده multicycle طراحی کنیم؛ با این تفاوت که واحد کنترل آن به جای استفاده از State Machine از یک Rom استفاده می کند و دستورات کنترلی را از آن دریافت می کند. ابتدا جدول ۱ در صورت پروژه را تکمیل کردیم تا شهودی از صورت پروژه و سیگنال های کنترلی برایمان ایجاد شود.

Micro-instruction (Name)	NextPc	Branch	MemW	RegW	IRWrite	AdrSrc	ResultSrc	ALUSrcA	ALUSrcB	ALUOP	(Partial) FSM Control Word
0 Fetch	1	0	0	0	1	0	10	01	10	0	0x114C
1 Decode	0	0	0	0	0	0	10	01	10	0	0x004C
2 MemAdr	0	0	0	0	0	0	00	00	01	0	0x0002
3 MemRead	0	0	0	0	0	1	00	00	00	0	0x0080
4 MemWB	0	0	0	1	0	0	01	00	00	0	0x0220
5 MemWrite	0	0	1	0	0	1	00	00	00	0	0x0480
6 ExecuteR	0	0	0	0	0	0	0	00	00	1	0x0001
7 Executel	0	0	0	0	0	0	0	00	01	1	0x0003
8 ALUWB	0	0	0	1	0	0	00	00	00	0	0x0200
9 Branch	0	1	0	0	0	0	10	00	01	0	0x0842

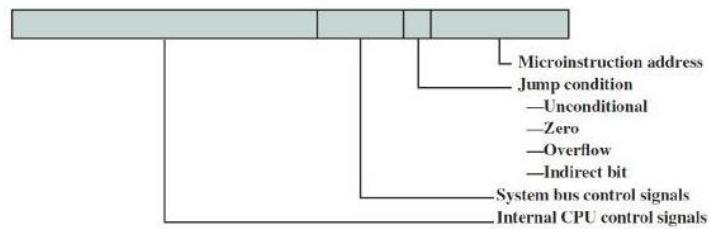
جدول ۱- جدول سیگنال های خروجی

در ادامه و با مطالعه کتاب دریافتیم که واحد کنترلی باید به صورت زیر طراحی شود:  
به همان صورت قبلی از واحد های decode و condLogic Control unit تشکیل می شود.  
واحد condLogic درست مانند قبل طراحی شده و تنها بخشی که باید تغییر کند همان بخش است که در مدل قبلی در کتاب Harris برای ساخت آن از FSM استفاده شده و حال باید تغییراتی لحاظ شود تا سیگنال های خروجی توسط یک واحد حافظه فقط خواندنی (Rom) پیاده سازی شود.

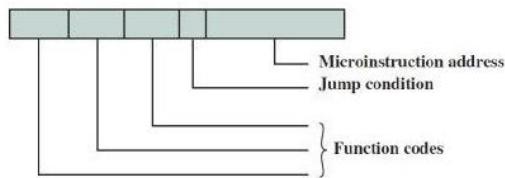
پیاده سازی با Rom:

Rom قرار است کلمه هایی که به اصطلاح به آن micro Instruction گفته می شود را نگهداری کند. در حقیقت مجموعه ای از bit هاست که با یک قالب کلی

سیگنال های خروجی واحد کنترل را در خود نگه می دارد. Microinstruction ها به دو صورت vertical و horizontal قابل پیاده سازیست که طبق صورت پژوهش ما آن را horizontal و vertical سازی کردیم. در حالت horizontal تمام سیگنال های کنترلی به صورت مستقیم در microinstruction مشخص می شود



(a) Horizontal microinstruction



(b) Vertical microinstruction

**Figure 21.1** Typical Microinstruction Formats

شکل ۱- تقاویت ریزدستور العمل و vertical و horizontal

Control Signals												Branch Condition			Branch Target	
FlagW	PCS	NextPC	RegW	MemW	IRWrite	AdrSrc	ResultSrc [1:0]	ALUSrcA [1:0]	ALUSrcB [1:0]	ImmSrc [1:0]	RegSrc [1:0]	ALUControl [1:0]	OP	Funct0	Funct5	Micro-instruction Address [4:0]

شکل ۲- قالب کلی ریزدستور العمل horizontal در صورت پژوهش

در داخل rom هر کدام از مراحلی که در FSM داشتیم را تبدیل به یک microinstruction می کنیم. به این صورت که ما در هر State خروجی های مشخصی داشتیم، حال در هر microinstruction همان سیگنال های خروجی را مطابق قالب کلی تعیین می کنیم. حال باید فرآیندی را طراحی کنیم که مطابق با آن در هر cycle یک microinstruction از rom خوانده شود، سیگنال ها فعل شوند، و در cycle بعدی به microinstruction rom برویم و به همین ترتیب.

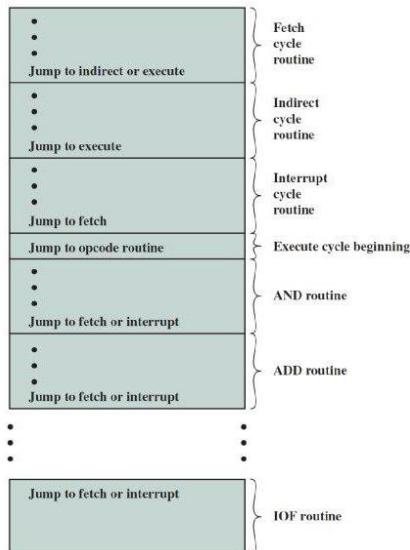
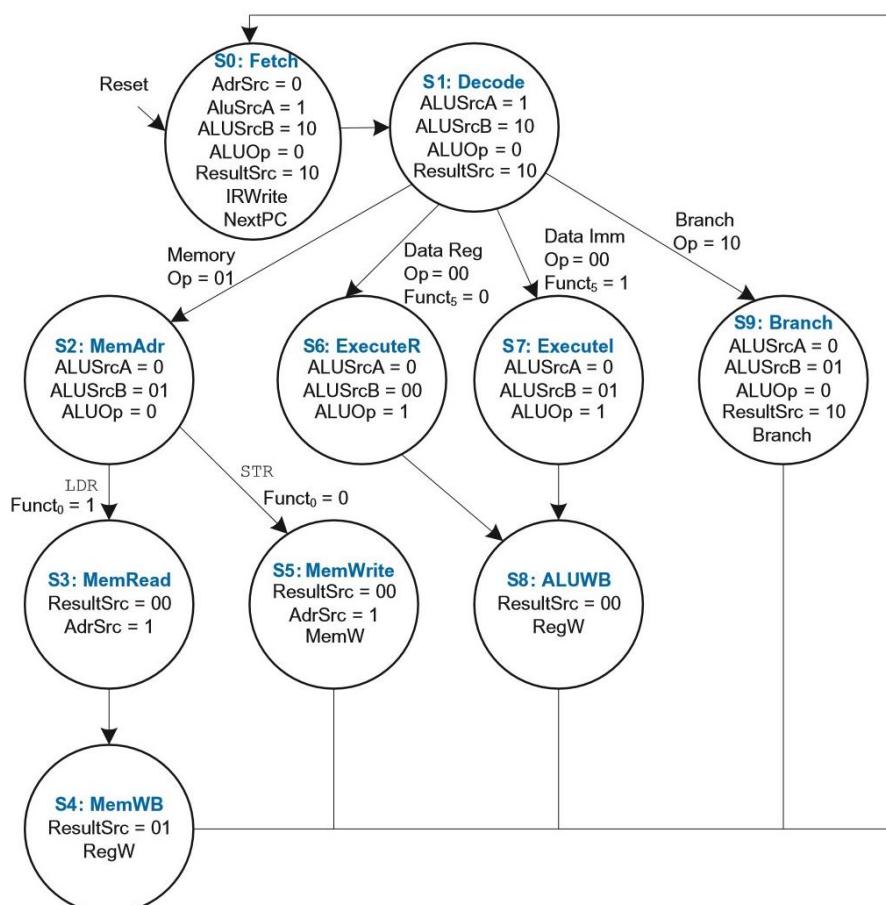


Figure 21.2 Organization of Control Memory

### شکل ۳ – ریز دستور العمل ها و روتین های داخل مموری

اولین چالش ما این است که microinstruction بعدی باید چگونه انتخاب شود. در قالب کلی صورت پروژه ۵ bit سمت راست هر microinstruction، آدرس microinstruction بعدی است. می دانیم در اجرای هر instruction ابتدا ریز دستور fetch و سپس ریز دستور decode انجام می شود. بعد از انجام ریز دستور decode متوجه می شویم که ادامه فرآیند، باید در کدام دنبال شود (هر کدام از شاخه های بعد از routine یک decode routine)

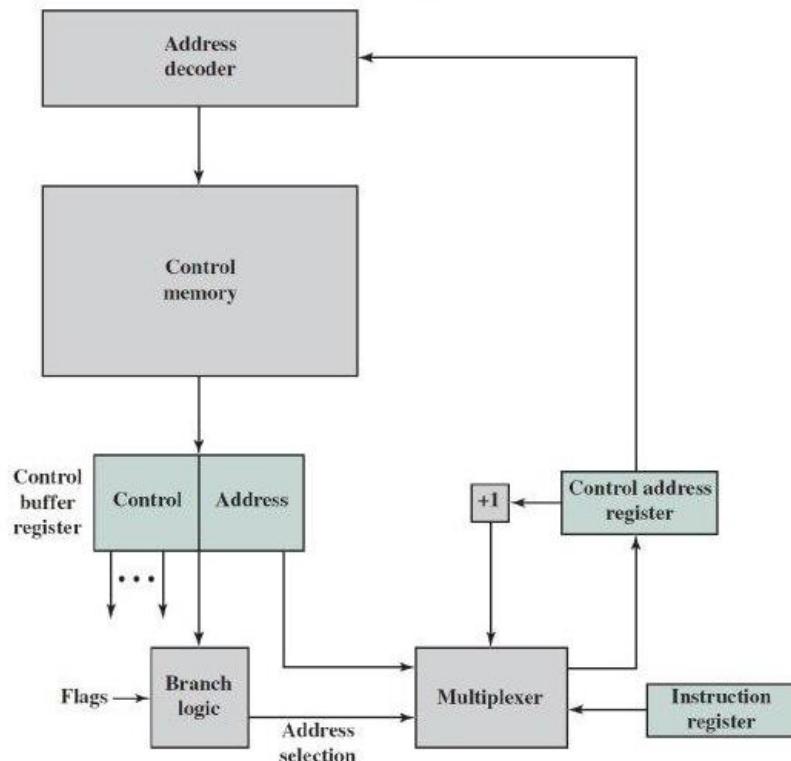


شکل ۴ – شکل ۷,۴۱ کتاب هریس – FSM state های

در ادامه در هر روتین نیز باید ریز دستورات به ترتیب انجام شوند تا instruction پایان یابد و دوباره به ریز دستورالعمل fetch برگردیم.

چالش اصلی اینجاست که چطور بعد از مرحله decode روتین بعدی مشخص می شود. طبق متن کتاب و صورت پروژه در قالب کلی microinstruction (شکل ۱) ۵ بیت سمت راست مربوط به آدرس ریز دستورالعمل بعدی است. همین طور ۴ بیت بعد از آن نیز برای branch condition در نظر گرفته شده که مربوط به سیگنال های funct0، funct5 و op می باشد. مطابق شکل ۲ مشخص است که با استفاده از سیگنال های op و funct5 میتوانیم تشخیص دهیم که بعد از decode باید به کدام microinstruction برویم؛ همین طور با سیگنال funct0 تشخیص می دهیم بعد از memAdr باید کدام روتین را ادامه دهیم.

روش بیان شده در کتاب این است که به طور معمول بعد از اجرای هر ریزدستورالعمل باید ریز دستورالعمل بعدی در rom (آدرس بعدی) اجرا شود. مگر آنکه branch لازم باشد (مثل ریز دستورالعمل branch و decode) در صورتی که branch لازم باشد (که از روی next address تشخیص می دهیم) به سراغ بیت های condition می رویم و به microinstruction بعدی را مشخص می کند و آن ریز دستورالعمل را اجرا می کنیم. برای اجرا کردن این منطق مطابق شکل زیر خارج از memory ، باید logic را ایجاد کنیم تا با استفاده از سیگنال های branch condition تشخیص دهد که branch داریم یا نه؛ اگر داشتیم به next address برود، در غیر این صورت آدرس حاضر را بعلاوه ۱ کند و ریزدستورالعمل بعدی را اجرا کند.



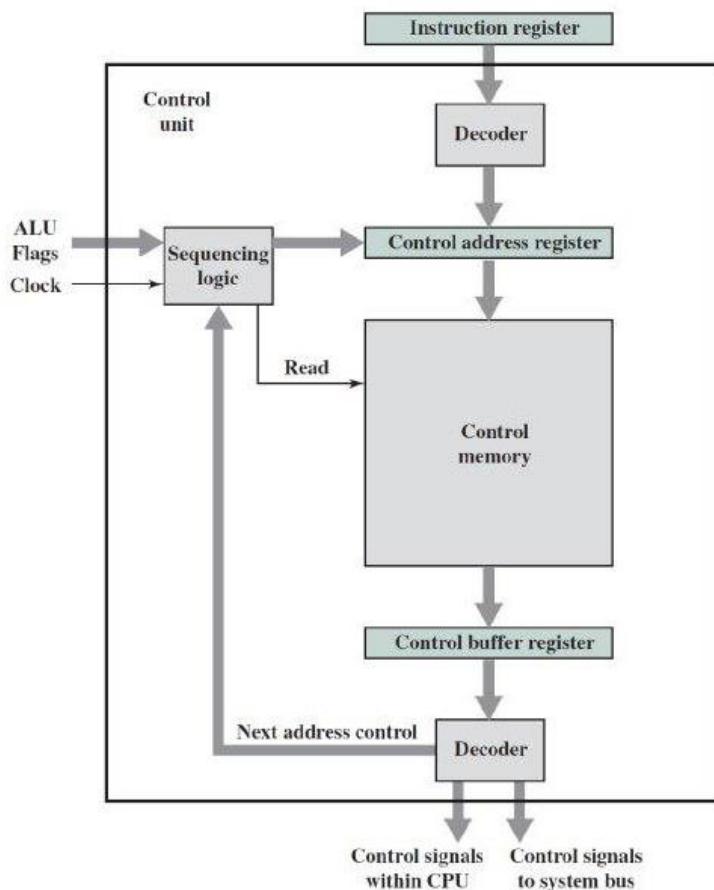
**Figure 21.7** Branch Control Logic: Single Address Field

شکل ۵ – منطق انتخاب آدرس بعدی

مشکل اصلی پیاده سازی این شیوه این بود که conditional logic ها چگونه باید داخل rom تعیین شوند؟

برای حل این مشکل بخش conditional logic را از Rom خارج کردیم. همین طور به تمام ریز دستورالعمل ها آدرس خانه بعدی آن را اختصاص دادیم یعنی دیگر لازم نیست آدرس را بعلاوه یک کنیم و در صورتی که باید ریزدستورالعمل بعدی اجرا شود مستقیماً به آدرس موجود در ریزدستورالعمل حاضر مراجعه می کند. همین طور برای بخش هایی که branch لازم است آدرس های temporal decode (به طور مثال ۱۱۱۱ برای ۱۱۱۰) را در نظر گرفتیم. در صورتی که سیگنال های آدرس بعدی یکی از این آدرس های رزرو باشند، با استفاده از branch condition instruction می گیریم مشخص می کنیم که آدرس بعدی چه باشد.

برای پیاده سازی این راه حل بخشی را در خارج از rom ایجاد کردیم به نام Sequential logic. این بخش فقط سیگنالهای branch Target را از microinstruction و funct5 و funct0 و Op را از instruction دریافت می کند. حال مطابق روش بالا و استفاده از چندین mux آدرس بعدی را انتخاب می کند و به memory میدهد تا دستور بعدی را اجرا کند. سیگنال های کنترلی هم مستقیماً از conditional logic و Datapath به control buffer register می رود تا ریز دستور العمل اجرا شود.



شکل ۶ - پیاده سازی microprogrammed controller

چالش بعدی در پیاده سازی آن بود که سیگنال های خروجی در قالب کلی microinstruction شامل سیگنال هایی بود که هر کدام برای تعیین شدن به شرط هایی احتیاج دارند که نمی توان آن را در rom بررسی کرد. مثل Pcs (سیگنال branch و RegW فعال باشند یا  $rd=15$  باشد)، .RegSrc و ImmSrc و AluControl و FlagW

برای حل این موضوع، مجبور شدیم که این بخش ها را از microinstruction خارج کنیم و مانند قبل بخش های pc Logic و instrDecoder و AluDecoder را درست مانند گذشته پیاده سازی کنیم.

از طرفی دیدیم که می توان executeR و executeI را از مموری خارج نکنیم به این شرط که برای هر کدام از حالت ها (register و immediate) حالت های ADD,SUB,AND,OR را جدا کنیم

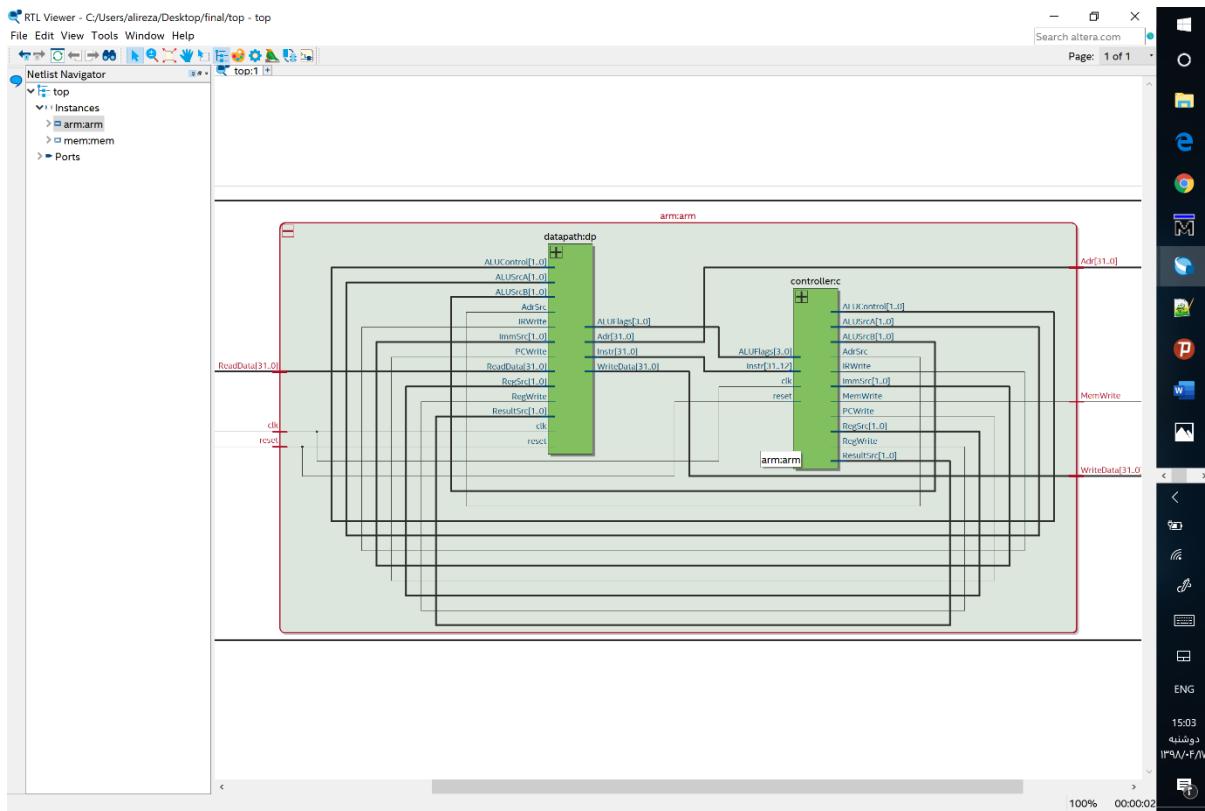
```
//Aluop_B_NextPc_RegW_MemW_IrWrite_AdrSrc_Resultsrc[2]_AlusrcA[2]_AlusrcB[2]_Alucontrol[2]_Nextadr[5]
5'b00000: dout = 20'b001001001100000001; //Fetch
5'b00001: dout = 20'b00000000100110001111; //Decode
5'b00010: dout = 20'b000000000000010011110; //MemAdr
5'b00011: dout = 20'b00000010000000000100; //memread
5'b00100: dout = 20'b00010000100000000000; //memwb
5'b00101: dout = 20'b00001010000000000000; //memwrite
5'b01000: dout = 20'b00010000000000000000; //AluwB
5'b01001: dout = 20'b01000001000010000000; //Branch

//execute R
5'b00110: dout = 20'b10000000000000001000; //ERADD
5'b01010: dout = 20'b1000000000000000101000; //ERSUB
5'b01011: dout = 20'b10000000000000001001000; //ERAND
5'b01100: dout = 20'b10000000000000001101000; //EROR

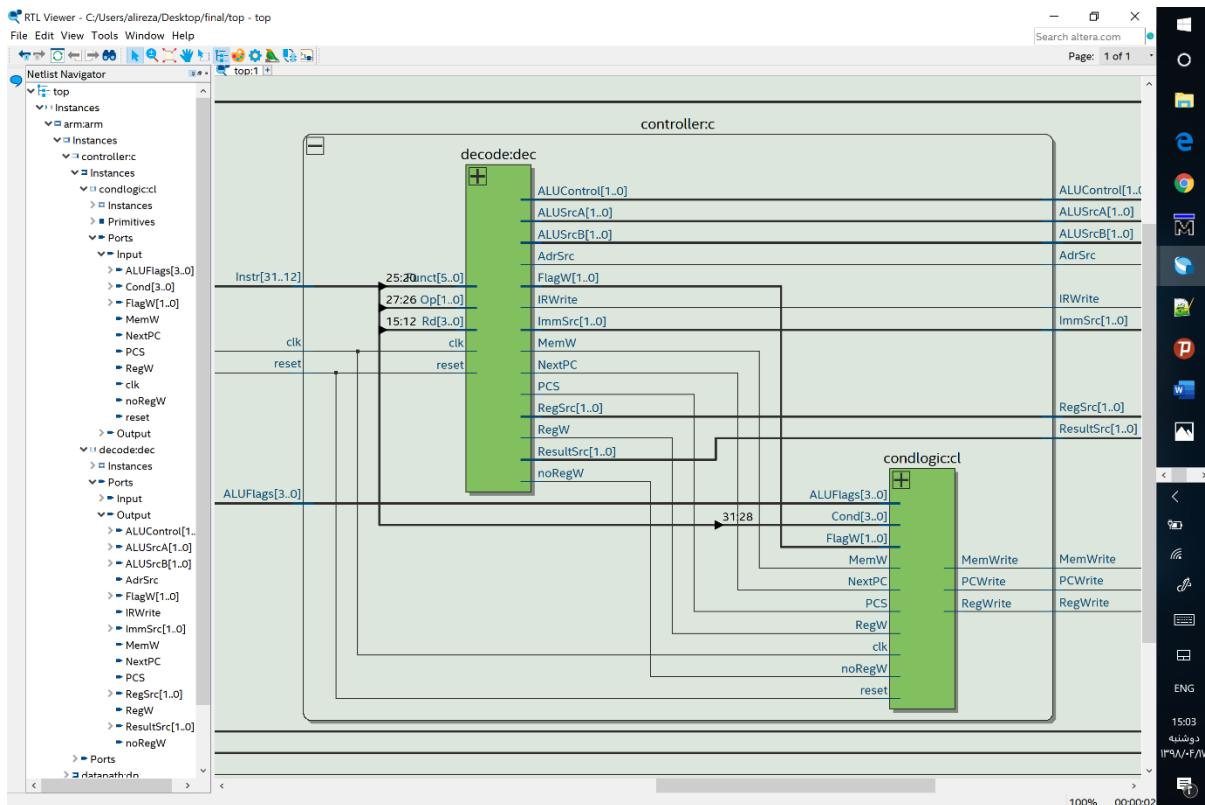
//execute I
5'b00111: dout = 20'b100000000000010001000; //EIADD
5'b01101: dout = 20'b100000000000010101000; //EISUB
5'b01110: dout = 20'b100000000000011001000; //EIAND
5'b01111: dout = 20'b100000000000011101000; //EIOR

default : dout = 20'b00000000000000000001; // undefined
```

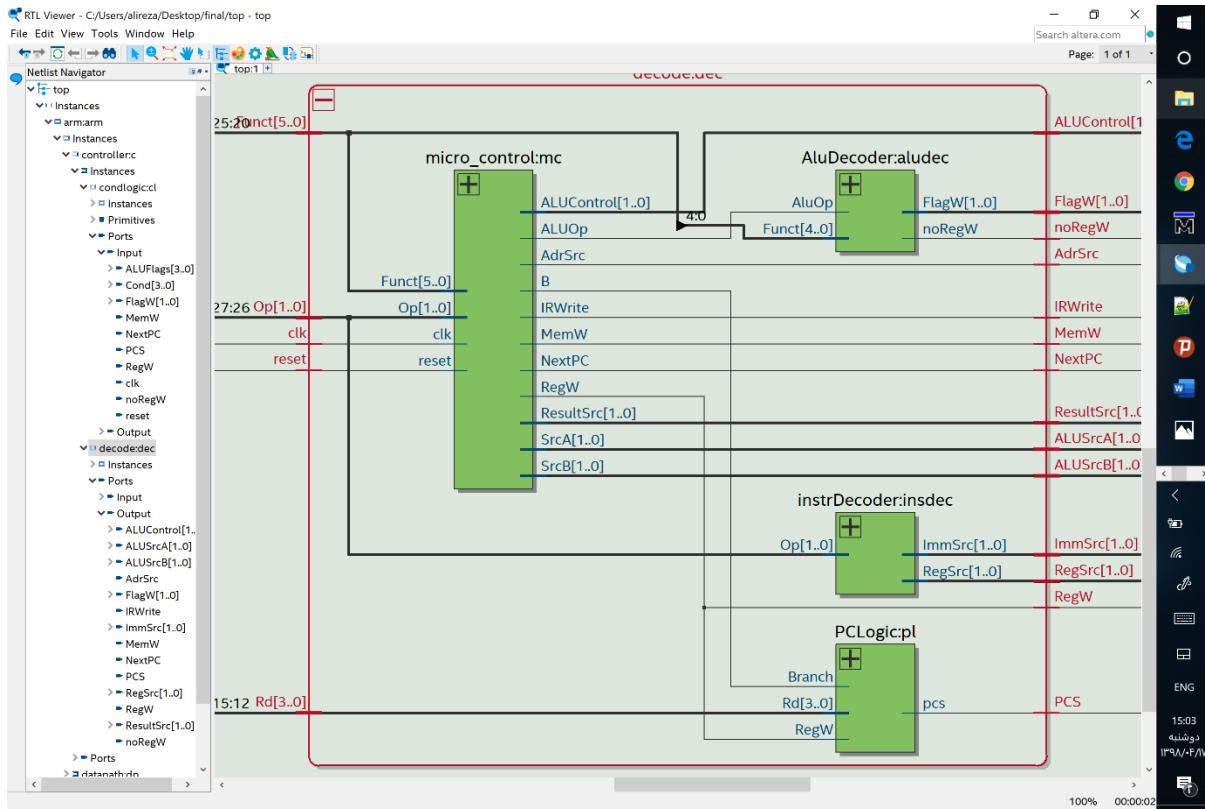
کد ۱ – ریزدستور العمل ها و پیاده سازی execute ها به صورت جداگانه



شکل ۷- پردازنده پیاده سازی شده



شکل ۸- کنترلر پیاده سازی شده



شکل ۹-بخش و اجزاء آن decode

مطابق شکل ۷ و کد ۱ بخش های instrdecoder، aludecoder و pclogic دوباره طراحی شده و alucontrol نیز از Rom پیاده سازی شده.

بعد از اتمام طراحی واحد کنترل، datapath نیز دقیقا مانند قبل پیاده سازی شد، سپس با نوشتن کد مربوط به testbench و کد نمونه پردازنده را تست کردیم نتایج را با تکمیل شده جدول ۲ در صورت پروژه مقایسه کردیم.

Cycle	Reset	Pc	Instr	FSM state	SrcA	SrcB	AluResult
1	1	00	0	Fetch	0	4	4
2	0	04	sub E04F00FF	Decode	4	4	8
3	0	04		ERsub	8	8	0
4	0	04		AluWb	X	X	X
5	0	04		Fetch	4	4	8
6	0	08	Add E2802005	Decode	8	4	C
7	0	08		EIAdd	0	5	5
8	0	08		AluWb	X	X	X
9	0	08		Fetch	8	4	C
10	0	0C	Add E280300C	Decode	C	4	10
11	0	0C		EIAdd	0	C	C
12	0	0C		AluWb	X	X	X
13	0	0C		Fetch	C	4	10
14	0	10	Sub E2437009	Decode	10	4	14

15	0	10		EISub	C	9	3
16	0	10		AluWb	X	X	X
17	0	10		Fetch	10	4	14
18	0	14	Or E1874002	Decode	14	4	18
19	0	14		ERor	3	5	7
20	0	14		AluWb	X	X	X
21	0	14		Fetch	14	4	18
22	0	18	AND E0035004	Decode	18	4	1C
23	0	18		ERAnd	C	7	4
24	0	18		AluWb	X	X	X
25	0	18		Fetch	18	4	1C
26	0	1c	Add E0855004	Decode	1C	4	20
27	0	1c		ERAdd	4	7	B
28	0	1c		AluWb	X	X	X
29	0	1c		Fetch	1C	4	20
30	0	20	Sub E0558007	Decode	20	4	24
31	0	20		ERSub	B	3	8
32	0	20		AluWb	X	X	X
33	0	20		Fetch	20	4	24
34	0	24	B 0A00000C	Decode	24	4	28
35	0	24		Branch	28	30	58
36	0	24		Fetch	24	4	28
37	0	28	Sub E0538004	Decode	28	4	2C
38	0	28		ERSub	C	7	5
39	0	28		AluWb	X	X	X
40	0	28		Fetch	28	4	2C
41	0	2c	B AA000000	Decode	2C	4	30
42	0	2c		Branch	30	0	30
43	0	34		Fetch	30	4	34
44	0	34	Sub E0578002	Decode	34	4	38
45	0	34		ERSub	3	5	FFFFFFFE
46	0	34		AluWb	X	X	X
47	0	34		Fetch	34	4	38
48	0	38	Add B2857001	Decode	38	4	3C
49	0	38		EIAdd	B	1	C
50	0	38		AluWb	X	X	X
51	0	38		Fetch	38	4	3C
52	0	3c	Sub E0477002	Decode	3C	4	40
53	0	3c		ERSub	C	5	7
54	0	3c		AluWb	X	X	X
55	0	3c		Fetch	3C	4	40
56	0	40	STR E5837054	Decode	40	4	44
57	0	40		MemAdr	C	54	60
58	0	40		MemWrite	X	X	X

59	0	40		Fetch	40	4	44
60	0	44	LDR E5902060	Decode	44	4	48
61	0	44		MemAdr	0	60	60
62	0	44		MemRead	X	X	X
63	0	44		MemWb	X	X	X
64	0	44		fetch	44	4	48
65	0	48	Add E08FF000	Decode	48	4	4C
66	0	48		ERAdd	4C	0	4C
67	0	48		AluWb	X	X	X
68	0	50		fetch	4C	4	50
69	0	50	B EA000001	Decode	50	4	54
70	0	50		Branch	54	4	58
71	0	60		fetch	58	4	5C
72	0	60	STR E5802064	Decode	5C	4	60
73	0	60		MemAdr	0	64	64
74	0	60		MemWrite	X	X	X

جدول ۲- جدول ۲ صورت پروژه تکمیل شده

در انتها بعد از اتمام پروژه دستورات shift و CMP را به مجموعه دستورات قابل پردازش اضافه کردیم.

دستور CMP: در aludecoder با تولید سیگنال noregw بوسیله  $[1:4]$  funct از اعمال تغییرات بر روی register جلوگیری میشود و دستور sub برای مقایسه اجرا میشود.

دستور shift و rotate: قبل از mux-srcb مازول shifter قرار میدهیم تا در صورت لزوم rm را shift یا rotate دهیم.

Quartus Prime Lite Edition - C:/Users/alireza/Desktop/final/top - top

File Edit View Project Assignments Processing Tools Window Help

Project Navigator Hierarchy Entity:Instance Cyclone IV E: EP4CE115F29C top

Table of Contents Flow Summary

Flow Status Successful - Mon Jul 08 11:25:52 2019  
 Revision Name 17.1.0 Build 590 10/25/2017 SJ Lite Edition  
 Top-level Entity Name top  
 Family Cyclone IV E  
 Device EP4CE115F29C7  
 Timing Models Final  
 Total logic elements 4,226 / 114,480 ( 4 %)  
 Total registers 2762  
 Total pins 67 / 529 ( 2762 )  
 Total virtual pins 0  
 Total memory bits 0 / 3,981,312 ( 0 %)  
 Embedded Multiplier 9-bit elements 0 / 532 ( 0 %)  
 Total PLLs 0 / 4 ( 0 %)

IP Catalog Installed IP Project Directory No Selection Available Library Basic Functions DSP Interface Protocols Memory Interfaces and Controllers Processors and Peripherals University Program Search for Partner IP

Tasks Compilation Task TimeQuest Timing Analysis Edit Settings View Report TimeQuest Timing Analyzer EDA Netlist Writer Edit Settings Program Device (Open Programmer)

All Find... Find Next

Messages 332140 No Recovery paths to report 332140 No Removal paths to report 332146 Worst-case minimum pulse width slack is -3.000 Analyzing Fast 1200mV OC Model 332123 Deriving Clock Uncertainty. Please refer to report\_sdc in TimeQuest to see clock uncertainties. 332148 Timing requirements not met 332146 Worst-case setup slack is -6.951 332146 Worst-case hold slack is 0.181 332140 No Removal paths to report 332140 No Removal paths to report 332146 Worst-case minimum pulse width slack is -3.000 332102 Design is not fully constrained for setup requirements 332102 Design is not fully constrained for hold requirements Quartus Prime TimeQuest Timing Analyzer was successful. 0 errors, 5 warnings 293000 Quartus Prime Full Compilation was successful. 0 errors, 23 warnings

System (1) Processing (147)

TimeQuest Timing Analyzer - C:/Users/alireza/Desktop/final/top - top

File View Netlist Constraints Reports Script Tools Window Help Set Operating Conditions Slow 1200mV 85C Model

Slow 1200mV 85C Model Slow 1200mV OC Model Fast 1200mV OC Model

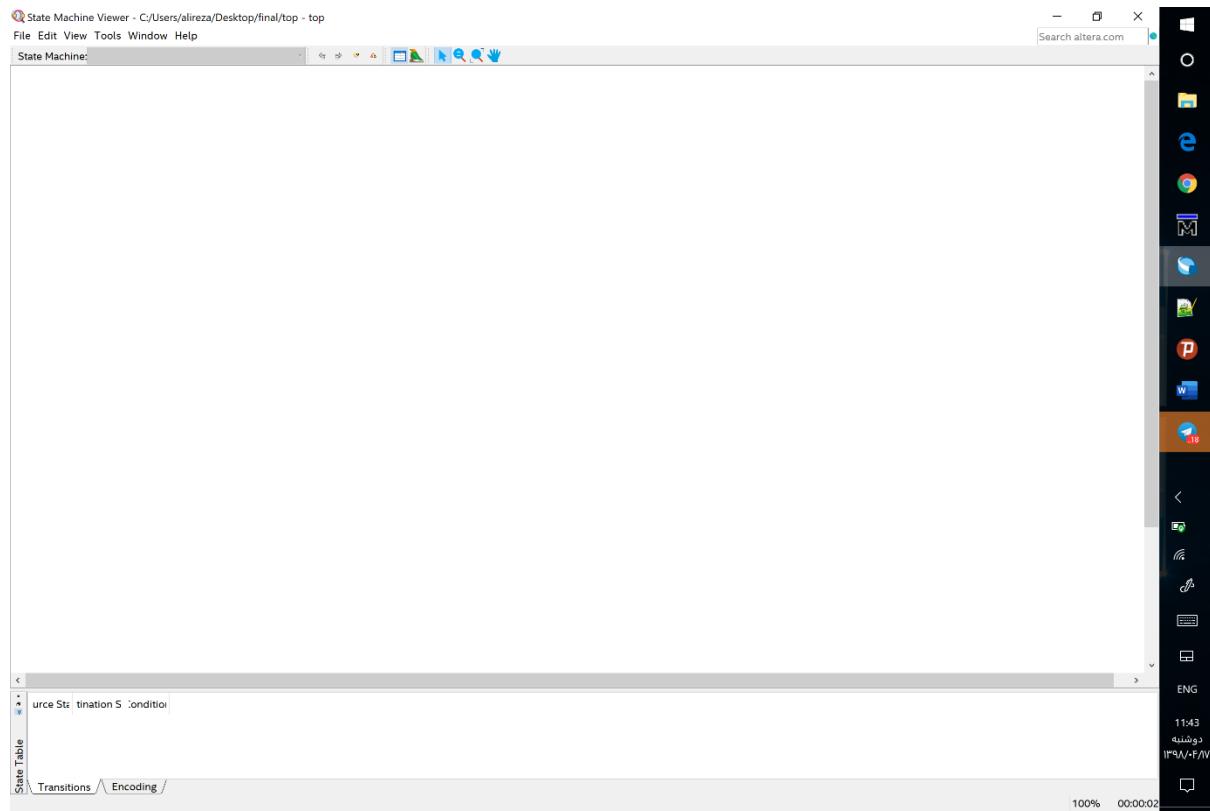
Report TimeQuest Timing Analyzer Advanced I/O Timing Report Path Slow 1200mV 85C Model Slow 1200mV OC Model Fast 1200mV OC Model

Path #1: Delay is 19.040 Path #1: Delay is 19.040

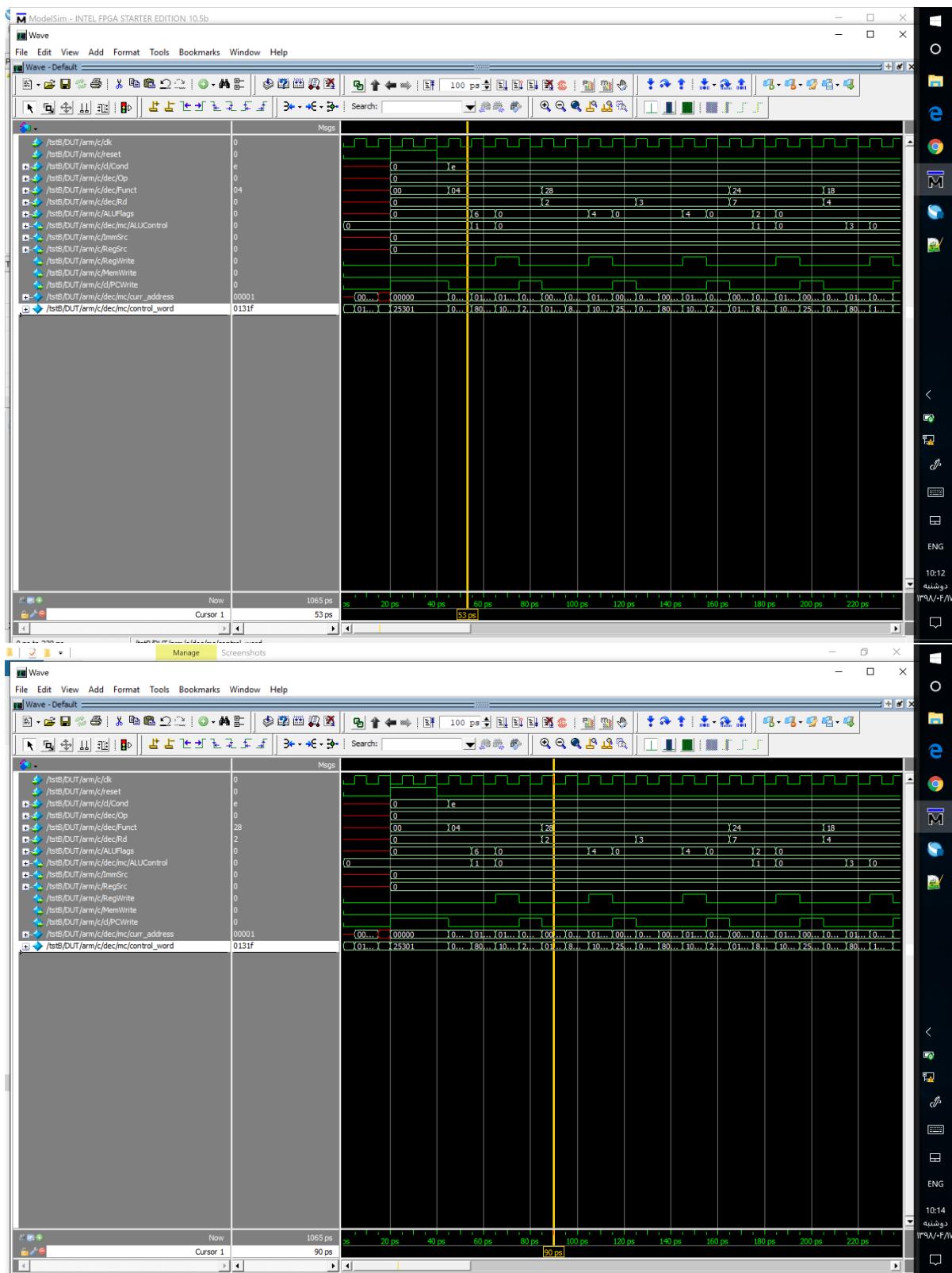
Total	Incr	RF	Type	Fanout	Location	Ele	Total	Incr	RF	Type	Fanout	Location	Ele
19.040	19.040			1	FF_X86_Y37_N9	data path	19.040	19.040			1	FF_X86_Y37_N9	data path
0.000	0.000			1	FF_X86_Y37_N9	arm dp RegDataReg2[q 18] Adr[31]	0.000	0.000			1	FF_X86_Y37_N9	arm dp RegDataReg2[
0.844	0.844	FF	IC	1	LCCOMB_X86_Y38_N0	arm dp WriteDsh Shift	0.844	0.844	FF	IC	1	LCCOMB_X86_Y38_N0	arm dp WriteDsh Shift
1.269	0.425	FF	CELL	5	LCCOMB_X86_Y38_N0	arm dp WriteDsh Shift	1.269	0.425	FF	CELL	5	LCCOMB_X86_Y38_N0	arm dp WriteDsh Shift
1.587	0.318	FF	IC	1	LCCOMB_X86_Y38_N28	arm dp WriteDsh Shift	1.587	0.318	FF	IC	1	LCCOMB_X86_Y38_N28	arm dp WriteDsh Shift
2.012	0.425	FF	CELL	2	LCCOMB_X86_Y38_N28	arm dp WriteDsh Shift	2.012	0.425	FF	CELL	2	LCCOMB_X86_Y38_N28	arm dp WriteDsh Shift
3.284	1.272	FF	IC	1	LCCOMB_X86_Y36_N28	arm dp WriteDsh Shift	3.284	1.272	FF	IC	1	LCCOMB_X86_Y36_N28	arm dp WriteDsh Shift
3.564	0.280	FF	CELL	2	LCCOMB_X86_Y36_N28	arm dp WriteDsh Shift	3.564	0.280	FF	CELL	2	LCCOMB_X86_Y36_N28	arm dp WriteDsh Shift
3.856	0.292	FF	IC	1	LCCOMB_X86_Y36_N10	arm dp WriteDsh Out	3.856	0.292	FF	IC	1	LCCOMB_X86_Y36_N10	arm dp WriteDsh Out
4.260	0.404	FF	CELL	1	LCCOMB_X86_Y36_N10	arm dp WriteDsh Out	4.260	0.404	FF	CELL	1	LCCOMB_X86_Y36_N10	arm dp WriteDsh Out
4.487	0.227	FF	IC	1	LCCOMB_X86_Y36_N10	arm dp WriteDsh Out	4.487	0.227	FF	IC	1	LCCOMB_X86_Y36_N10	arm dp WriteDsh Out
4.612	0.125	FF	CELL	1	LCCOMB_X86_Y36_N4	arm dp WriteDsh Out	4.612	0.125	FF	CELL	1	LCCOMB_X86_Y36_N4	arm dp WriteDsh Out
6.359	1.747	FF	IC	1	LCCOMB_X91_Y30_N24	arm dp WriteDsh Out	6.359	1.747	FF	IC	1	LCCOMB_X91_Y30_N24	arm dp WriteDsh Out
6.640	0.281	FF	CELL	1	LCCOMB_X91_Y30_N24	arm dp WriteDsh Out	6.640	0.281	FF	CELL	1	LCCOMB_X91_Y30_N24	arm dp WriteDsh Out
6.868	0.228	FF	IC	1	LCCOMB_X91_Y30_N12	arm dp ALUSrcBmux	6.868	0.228	FF	IC	1	LCCOMB_X91_Y30_N12	arm dp ALUSrcBmux
6.993	0.125	FF	CELL	1	LCCOMB_X91_Y30_N12	arm dp ALUSrcBmux	6.993	0.125	FF	CELL	1	LCCOMB_X91_Y30_N12	arm dp ALUSrcBmux
7.220	0.227	FF	IC	1	LCCOMB_X91_Y30_N18	arm dp ALUSrcBmux	7.220	0.227	FF	IC	1	LCCOMB_X91_Y30_N18	arm dp ALUSrcBmux
7.345	0.125	FF	CELL	1	LCCOMB_X91_Y30_N18	arm dp ALUSrcBmux	7.345	0.125	FF	CELL	1	LCCOMB_X91_Y30_N18	arm dp ALUSrcBmux

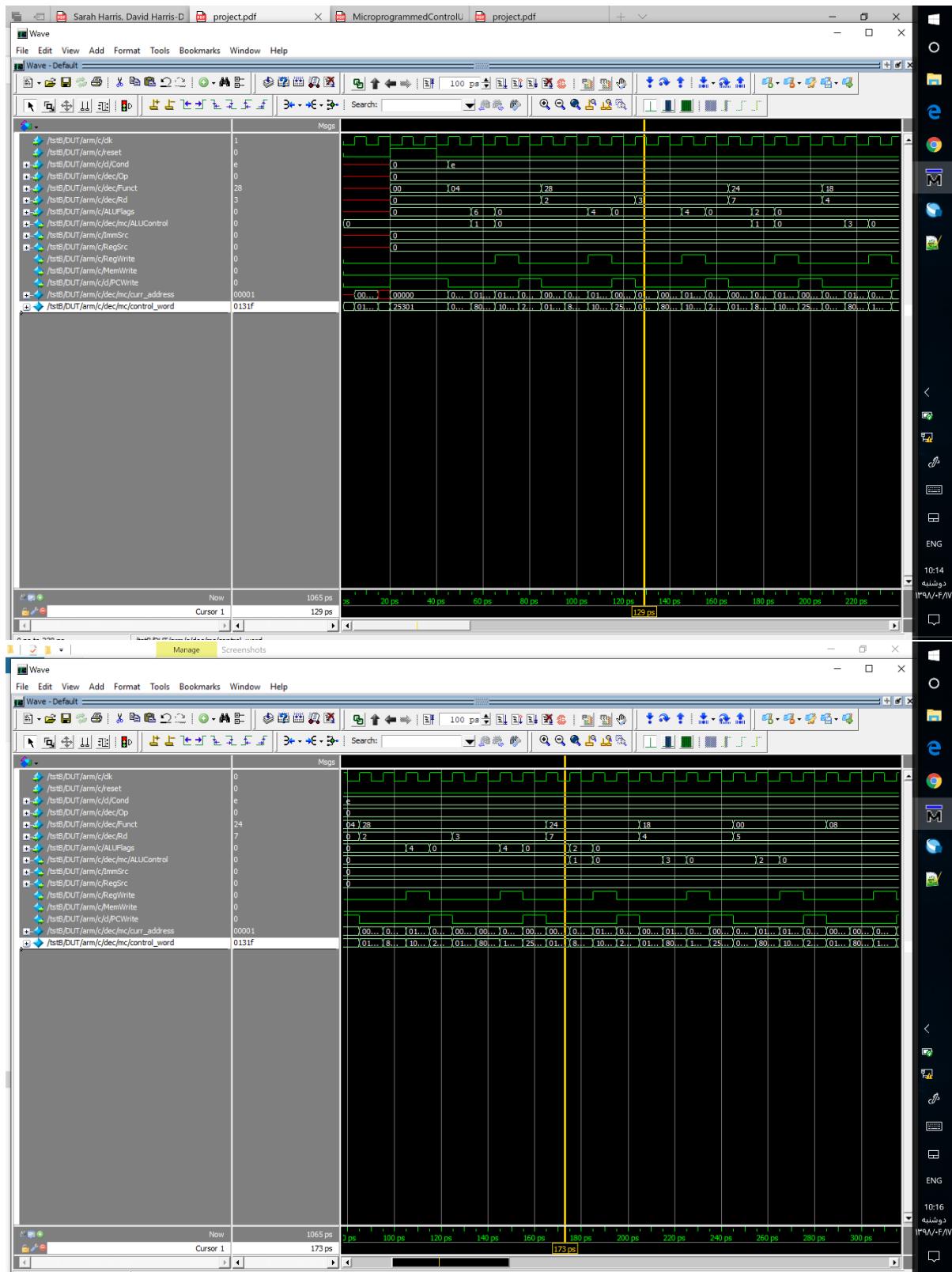
Console read\_sdc A Synopsys Design Constraints File file not found: 'top.sdc'. A Synopsys Design Constraints File is required by the TimeQuest Timing Analyzer to get proper timing analysis results. update\_timing\_netslist No user constrained base clocks found in the design. Calling "derive\_clocks -period 1.0"

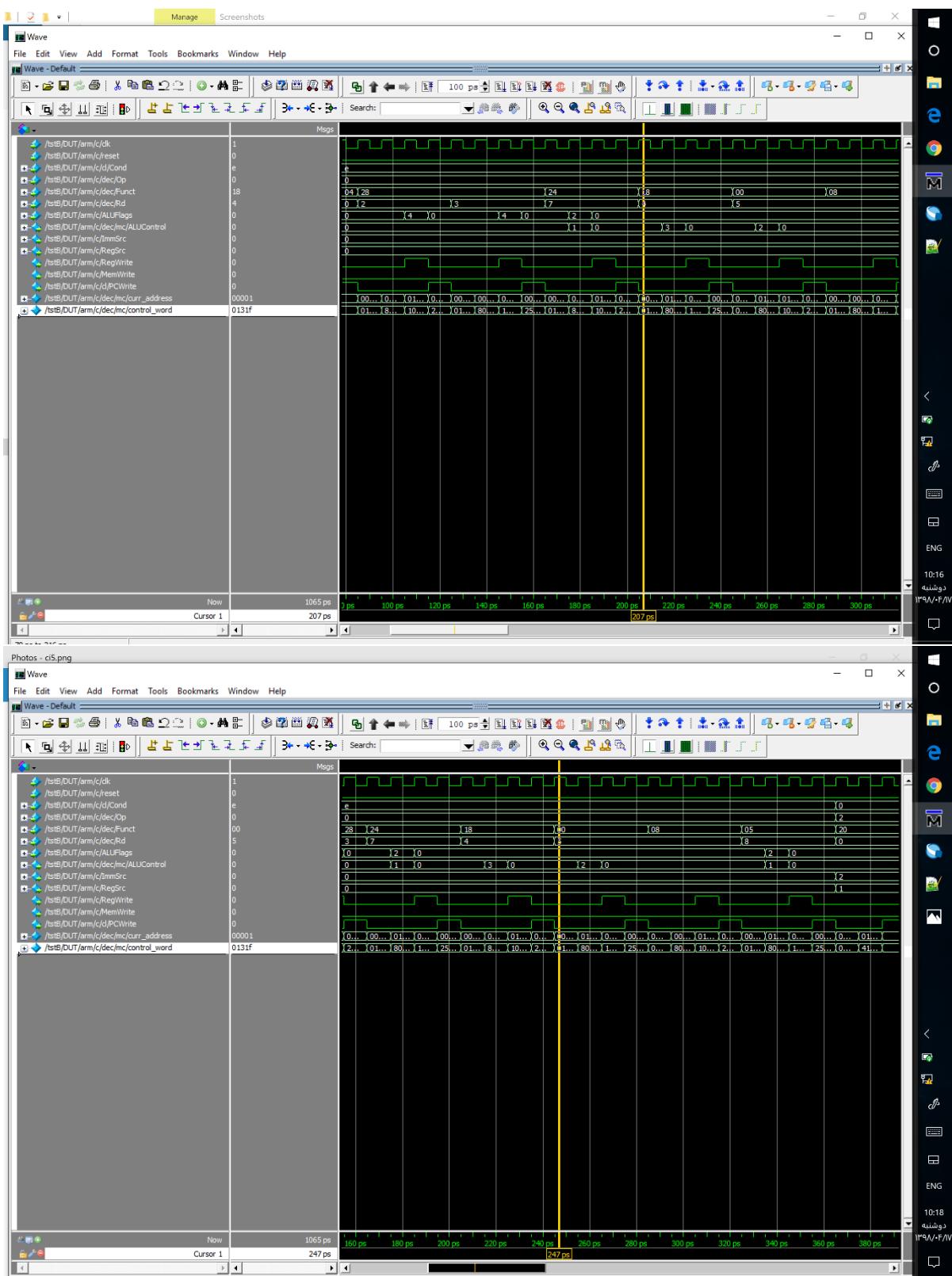
Search altera.com 11:28 ١٢:٣٠ م ٢٠١٧/٧/٨ ENG 11:28 ١٢:٣٠ م ٢٠١٧/٧/٨ ENG 0% 00:00:00 Ready

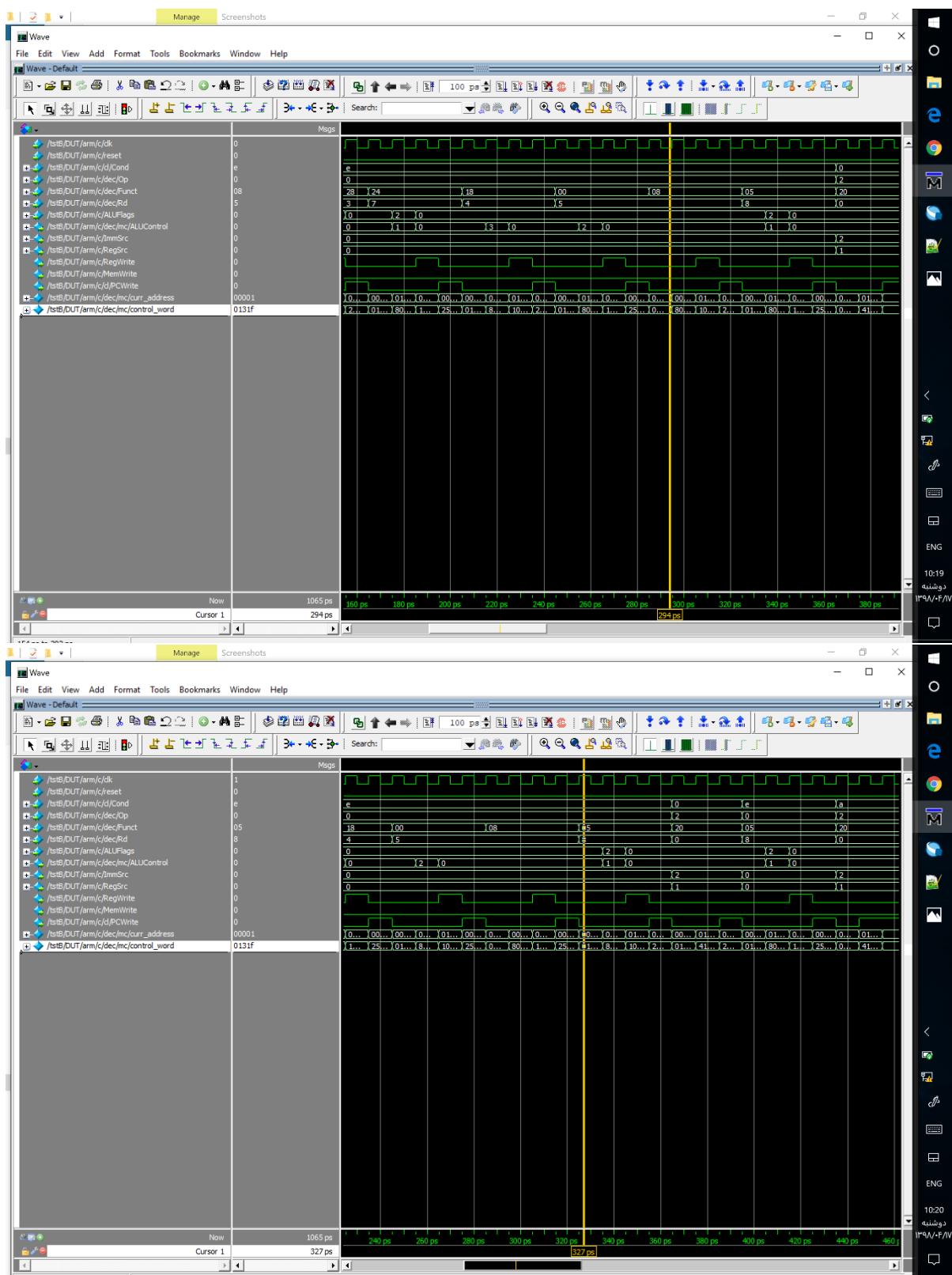


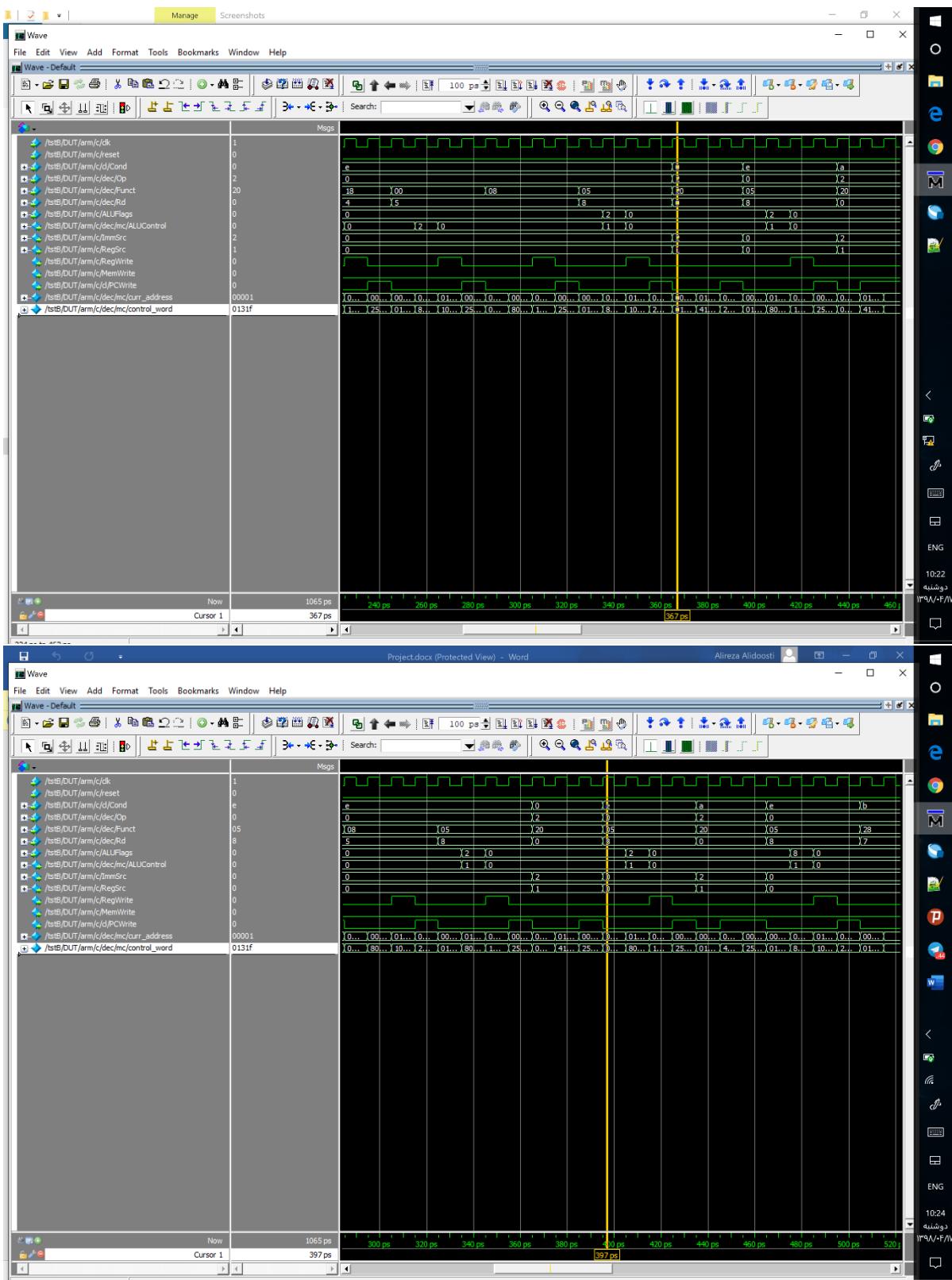
شکل موج های سیگنال های خواسته شده واحد کنترل برای هر کدام از instruction ها (در کد نمونه داده شده) در ریزدستور العمل decode (به ترتیب):

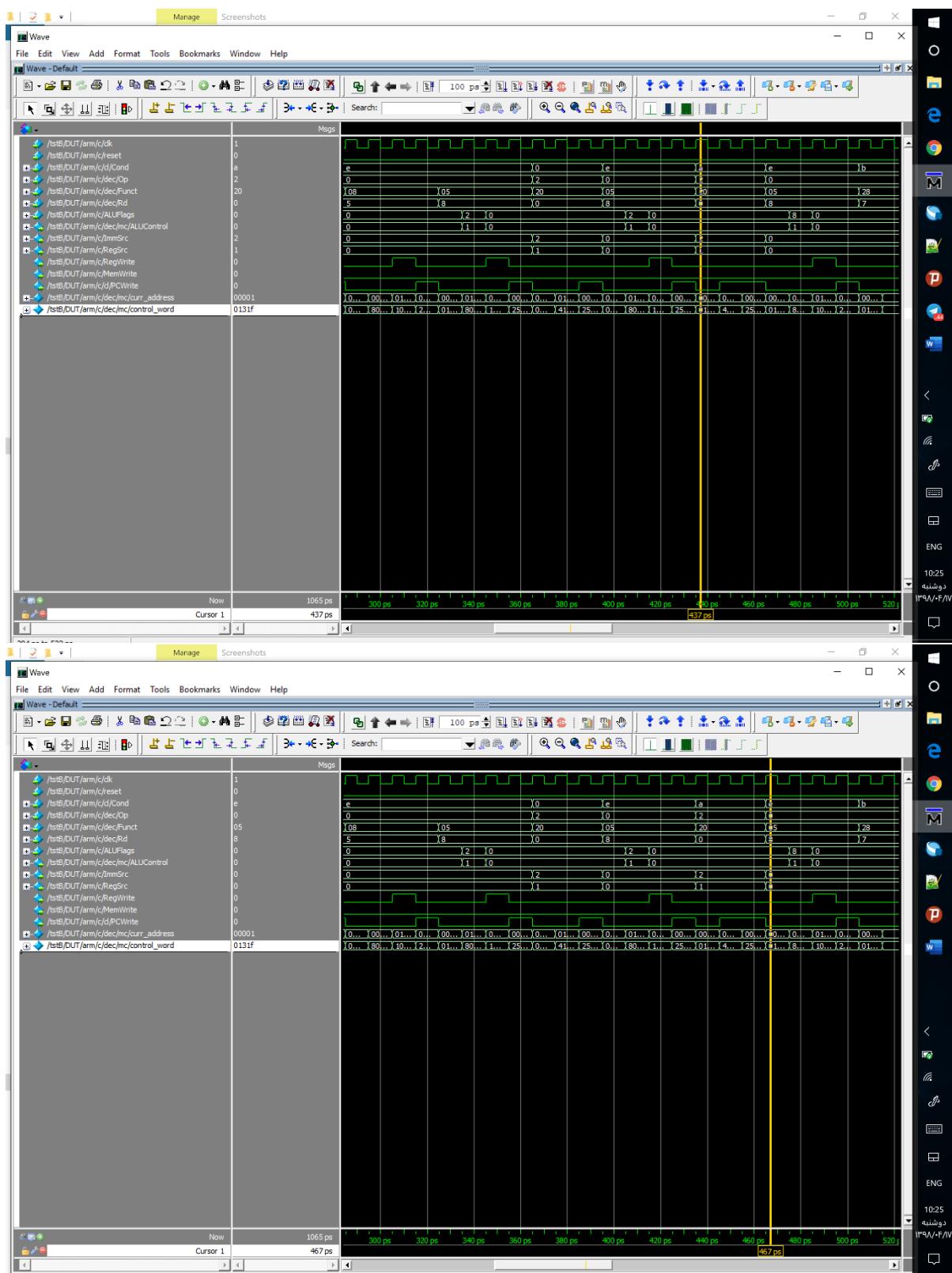


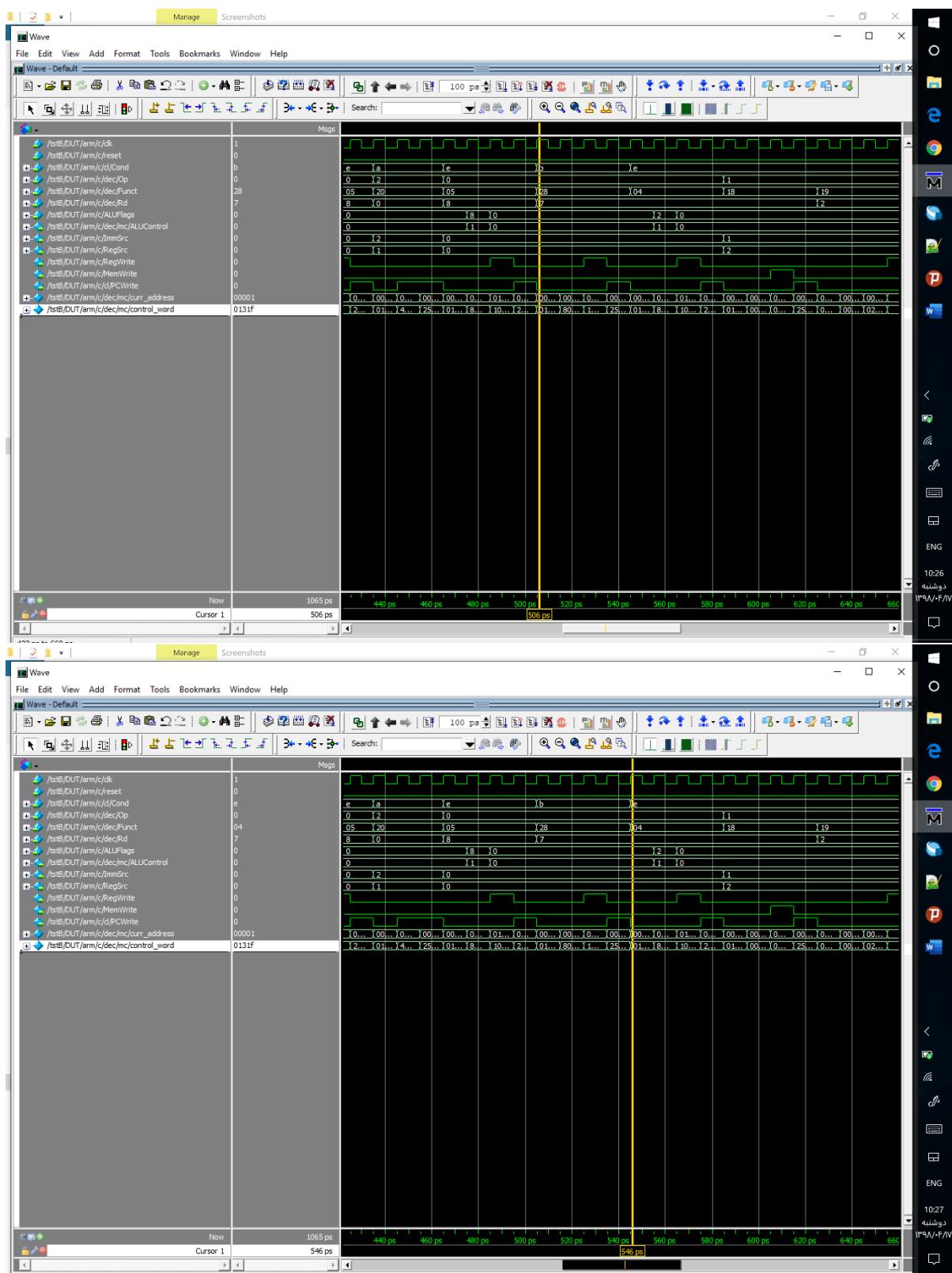


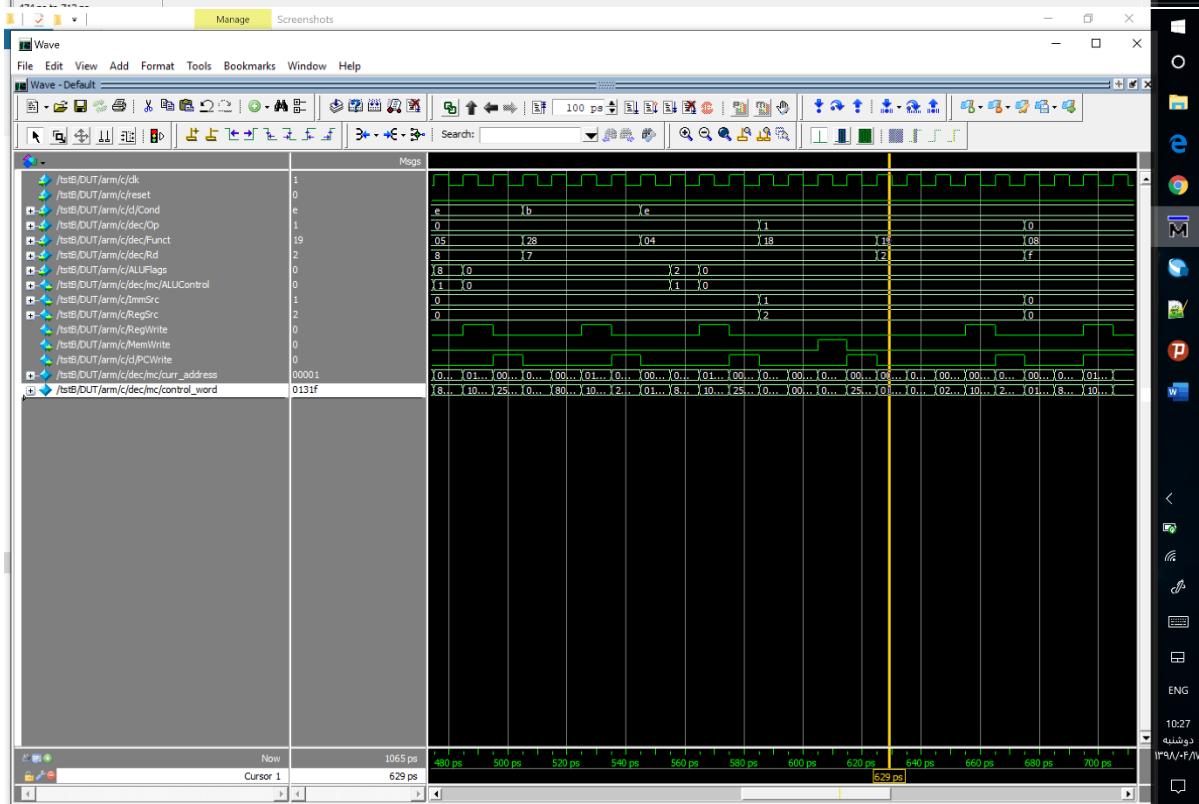
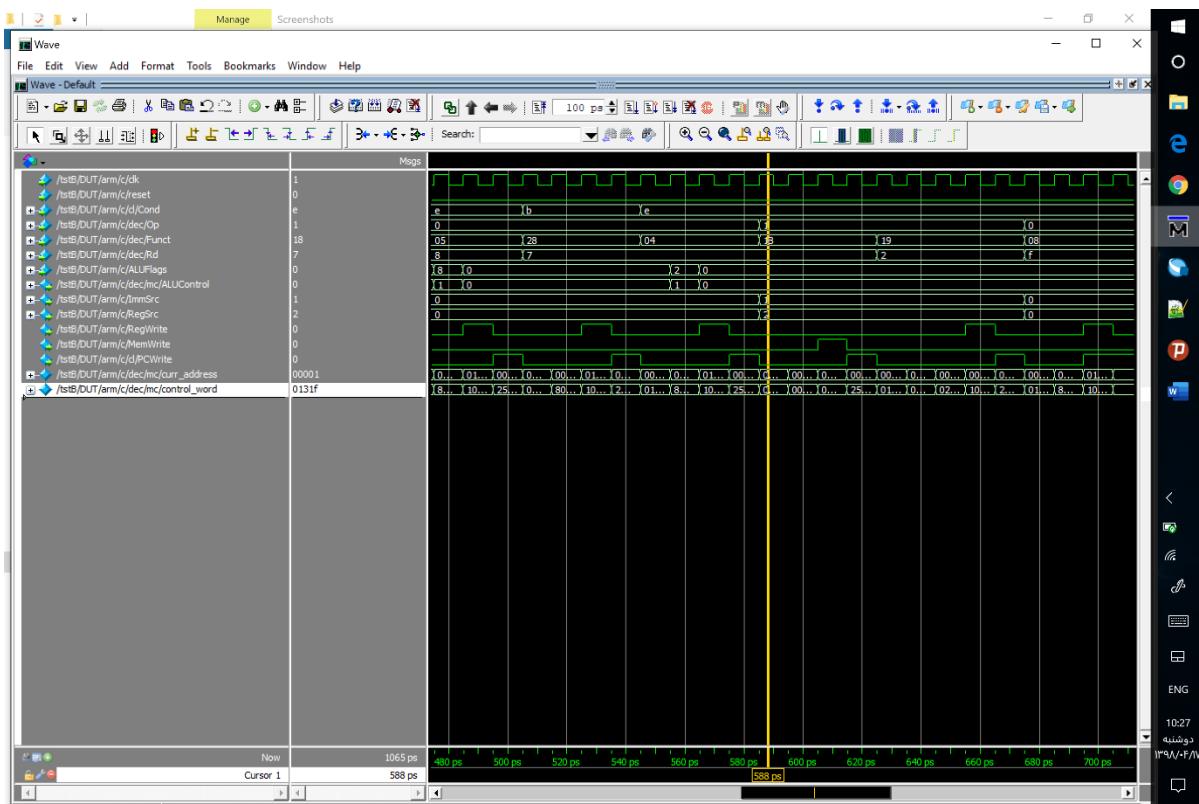


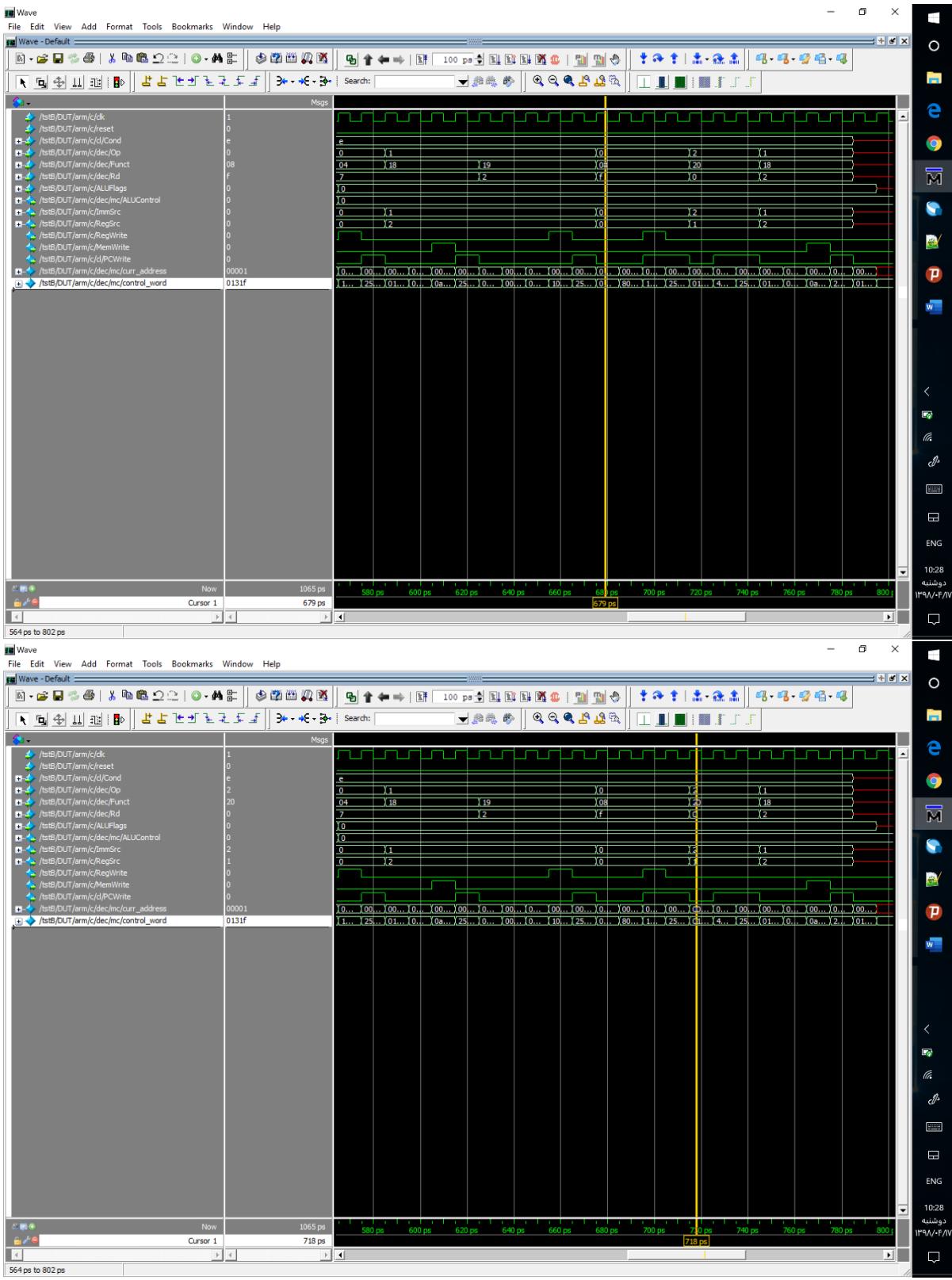


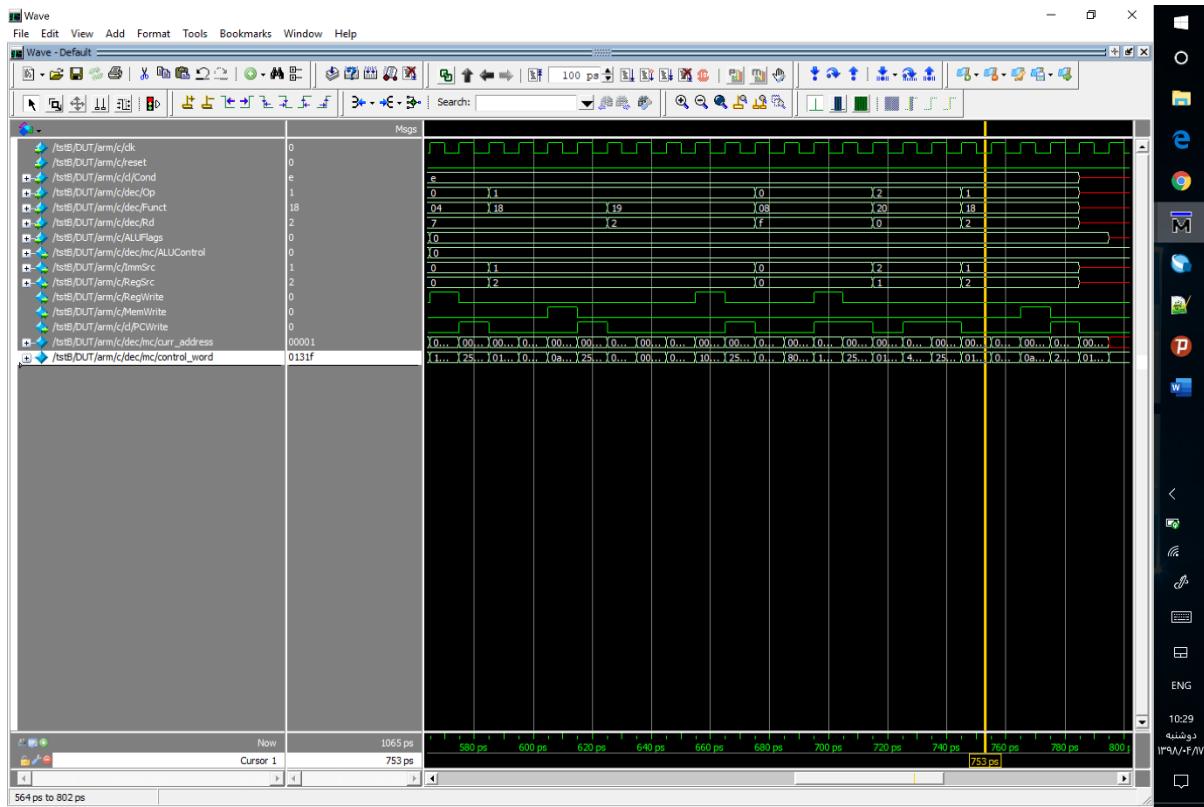












شکل موج های سیگنال های خواسته شده instruction ها (در کدام از datapath برای هر کدام از instruction ها (در کد نمونه داده شده) در ریزدستور العمل decode به ترتیب):

