پارسا نوری

فربد فولادی

فلسفه Microprogram بدان است که همانگونه که از اسمش پیداست بیاییم هر Instruction در مجموعه Instruction های زبان اسمبلی ARM را به یک Micro-Instruction تبدیل کنیم. حال به جای آنکه از برنامه نویس بخواهیم که برای ما با استفاده از این Micro-Instruction ها برنامه بنویسد خودمان با استفاده از Controller به نوعی یک Instruction را اجرا می‌کنیم که Data-Path طی چندین مرحله MicroInstruction ای مربوط به آن را انجام دهد.
برای این کار نیز میاییم همچون پردازنده تک‌سیکل عمل می‌کنیم و بخش decoder که کار تولید سیگنال های کنترلی خام هست را از بخش condlogic که سیگنال هایی که اجرای مشروط دارند را تولید می‌کند جدا می‌کنیم.
توضیح بخش decode:
دو بخش دارد که یکی ctrlunit نام است که بخش‌هایی که با توجه به نوع microinstruction متفاوت است را کنترل می‌کند و دیگری بخش‌هایی که با توجه به سیگنال های مربوط به microinstruction می‌توان سیگنال هایشان را تولید کرد را مدیریت می‌کند یعنی دقیقا سیگنال‌های RegSrc و ImmSrc و ALUControl و PCS. بخش مربوط به ALUControl و PCS کپی شده از کد آقای Harris است و ctrlunit با توجه به آموخته های آقای Stallingz پیاده سازی شده است. و سایر بخش های نیز با توجه به decoder تیک‌سیکل آقای هریس تولید و تنها با نگارشی دیگر تولید شده‌اند.
توضیح ctrlunit:
شامل سه بخش است:

1. Sequential log که در کد seqlogic نامیده شده است.
2. Control Memory که در کد ctrlmem نامیده شده است.
3. Control Buffer Register که در کد ctrlbffreg نامیده شده است.

توضیح Sequential Logic:
به این صورت عمل می‌کند که با توجه به مقادیر Op و Funct و nextAdr (که از ctrlmem آمده است) به این می‌پردازد microinstruction بعدی که باید اجرا بشود آدرسش در کجای Control Memory است.
توضیح Control Memory:
بدین صورت عمل می‌کند که با توجه به MicroInstruction به این مهم می‌پردازد که سیگنال‌های controlunit و آدرس بعدی MicroInstruction چه باشد آن ها را در قالب یک پک به نام controlword به بخش Control Buffer Register پاس می‌دهد.
یک مموری است که هر خانه اش مربوط به یک Mico-Instructin می‌باشد.وقتی می‌گوییم Firmware سخت‌افزارمان update شده است در اصل منظورمان این است که این بخش باز نویسی شده است.
توضیح Control Buffer Register:
بخشی ساده است که تنها کاری که باید انجام بدهد این است که سیگنال های pack شده آمده از ControlMemory را unpack کند.

توضیح CondLogic:
در Condlogic شبیه همان کد جناب آقای Harris برای تک‌سیکل عمل می‌کنیدم با این تفاوت که ارسال سیگنال های مشروط را یک چرخه تاخیر انجام می‌دهیم تا در microinstruction بعدی یعنی ALUWB کار را انجام دهند.

جدول اول:

NextPC	RegW	MemW	IRWrite	AdrSrc	ResultSrc	ALUSrcA	ALUSrcB	Branch	ALUOp	nextAdr	
1	0	0	1	0	10	1	10	0	0	0001	Fetch
0	0	0	0	0	10	1	10	0	0	1111	Decode
0	0	0	0	0	xx	0	01	0	0	1110	MemAdr
0	0	0	0	1	00	x	xx	0	x	0101	MemRead
0	0	1	0	1	00	x	xx	0	x	0000	MemWrite
0	1	0	0	0	01	x	xx	0	x	0000	MemWB
0	0	0	0	0	xx	0	00	0	1	1000	ExecuteR
0	0	0	0	0	xx	0	01	0	1	1000	ExecuteI
0	1	0	0	0	00	x	xx	0	x	0000	ALUWB
0	0	0	0	0	10	0	01	1	0	0000	Branch
x	x	x	x	x	xx	x	xx	x	x	xxxx	Default

جدول دوم در صفحه بعد امده است.

	Fetch	a:00000000	b:00000004	ALUResult:00000004
E04F000F	Decode	a:00000004	b:00000004	ALUResult:00000008
	ExecuteR	a:00000008	b:00000008	ALUResult:00000000
	ALUWB	a:XXX0X00X	b:XXX0X00x	ALUResult:xxxxxxxx
	Fetch	a:00000004	b:00000004	ALUResult:00000008
E2802005	Decode	a:00000008	b:00000004	ALUResult:0000000c
	ExecuteI	a:00000000	b:00000005	ALUResult:00000005
	ALUWB	a:0000000X	b:xxxxxxxx	ALUResult:xxxxxxxx
	Fetch	a:00000008	b:00000004	ALUResult:0000000c
E280300C	Decode	a:0000000c	b:00000004	ALUResult:00000010
	ExecuteI	a:00000000	b:0000000c	ALUResult:0000000c
	ALUWB	a:0000000X	b:xxxxxxxx	ALUResult:xxxxxxxx
	Fetch	a:0000000c	b:00000004	ALUResult:00000010
E2437009	Decode	a:00000010	b:00000004	ALUResult:00000014
	ExecuteI	a:0000000c	b:00000009	ALUResult:00000003
	ALUWB	a:000000XX	b:xxxxxxxx	ALUResult:xxxxxxxx
	Fetch	a:00000010	b:00000004	ALUResult:00000014
E1874002	Decode	a:00000014	b:00000004	ALUResult:00000018
	ExecuteR	a:00000003	b:00000005	ALUResult:00000007
	ALUWB	a:000000XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:00000014	b:00000004	ALUResult:00000018
E0035004	Decode	a:00000018	b:00000004	ALUResult:0000001c
	ExecuteR	a:0000000c	b:00000007	ALUResult:00000004
	ALUWB	a:000000XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:00000018	b:00000004	ALUResult:0000001c
E0855004	Decode	a:0000001c	b:00000004	ALUResult:00000020
	ExecuteR	a:00000004	b:00000007	ALUResult:0000000b
	ALUWB	a:000000XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:0000001c	b:00000004	ALUResult:00000020
E0558007	Decode	a:00000020	b:00000004	ALUResult:00000024
	ExecuteR	a:0000000b	b:00000003	ALUResult:00000008
	ALUWB	a:000000XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:00000020	b:00000004	ALUResult:00000024
0A00000C	Decode	a:00000024	b:00000004	ALUResult:00000028
	Branch	a:00000028	b:00000030	ALUResult:00000058
	Fetch	a:00000024	b:00000004	ALUResult:00000028

E0538004	Decode	a:00000028	b:00000004	ALUResult:0000002c
	ExecuteR	a:0000000c	b:00000007	ALUResult:00000005
	ALUWB	a:000000XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:00000028	b:00000004	ALUResult:0000002c
AA000000	Decode	a:0000002c	b:00000004	ALUResult:00000030
	Branch	a:00000030	b:00000000	ALUResult:00000030
	Fetch	a:00000030	b:00000004	ALUResult:00000034
E0578002	Decode	a:00000034	b:00000004	ALUResult:00000038
	ExecuteR	a:00000003	b:00000005	ALUResult:fffffffe
	ALUWB	a:000000XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:00000034	b:00000004	ALUResult:00000038
B2857001	Decode	a:00000038	b:00000004	ALUResult:0000003c
	ExecuteI	a:0000000b	b:00000001	ALUResult:0000000c
	ALUWB	a:000000XX	b:xxxxxxxx	ALUResult:xxxxxxxx
	Fetch	a:00000038	b:00000004	ALUResult:0000003c
E0477002	Decode	a:0000003c	b:00000004	ALUResult:00000040
	ExecuteR	a:0000000c	b:00000005	ALUResult:00000007
	ALUWB	a:000000Xc	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:0000003c	b:00000004	ALUResult:00000040
E5837054	Decode	a:00000040	b:00000004	ALUResult:00000044
	MemAdr	a:0000000c	b:00000054	ALUResult:00000060
	MemWrite	a:000000XX	b:000000XX	ALUResult:xxxxxxxx
	Fetch	a:00000040	b:00000004	ALUResult:00000044
E5902060	Decode	a:00000044	b:00000004	ALUResult:00000048
	MemAdr	a:00000000	b:00000060	ALUResult:00000060
	MemRead	a:000000XX	b:xxxxxxxx	ALUResult:xxxxxxxx
	MemWB	a:xxxxxxxx	b:xxxxxxxx	ALUResult:xxxxxxxx
	Fetch	a:00000044	b:00000004	ALUResult:00000048
E08FF000	Decode	a:00000048	b:00000004	ALUResult:0000004c
	ExecuteR	a:0000004c	b:00000000	ALUResult:0000004c
	ALUWB	a:XXX0X0XX	b:0000000X	ALUResult:xxxxxxxx
	Fetch	a:0000004c	b:00000004	ALUResult:00000050
E280200E	Decode	a:00000050	b:00000004	ALUResult:00000054
	Branch	a:00000054	b:00000004	ALUResult:00000058
	Fetch	a:00000058	b:00000004	ALUResult:0000005c
E5802064	Decode	a:0000005c	b:00000004	ALUResult:00000060
	MemAdr	a:00000000	b:00000064	ALUResult:00000064
	MemWrite	a:000000XX	b:000000XX	ALUResult:xxxxxxxx

اسکرین شات های فایل های سورس:

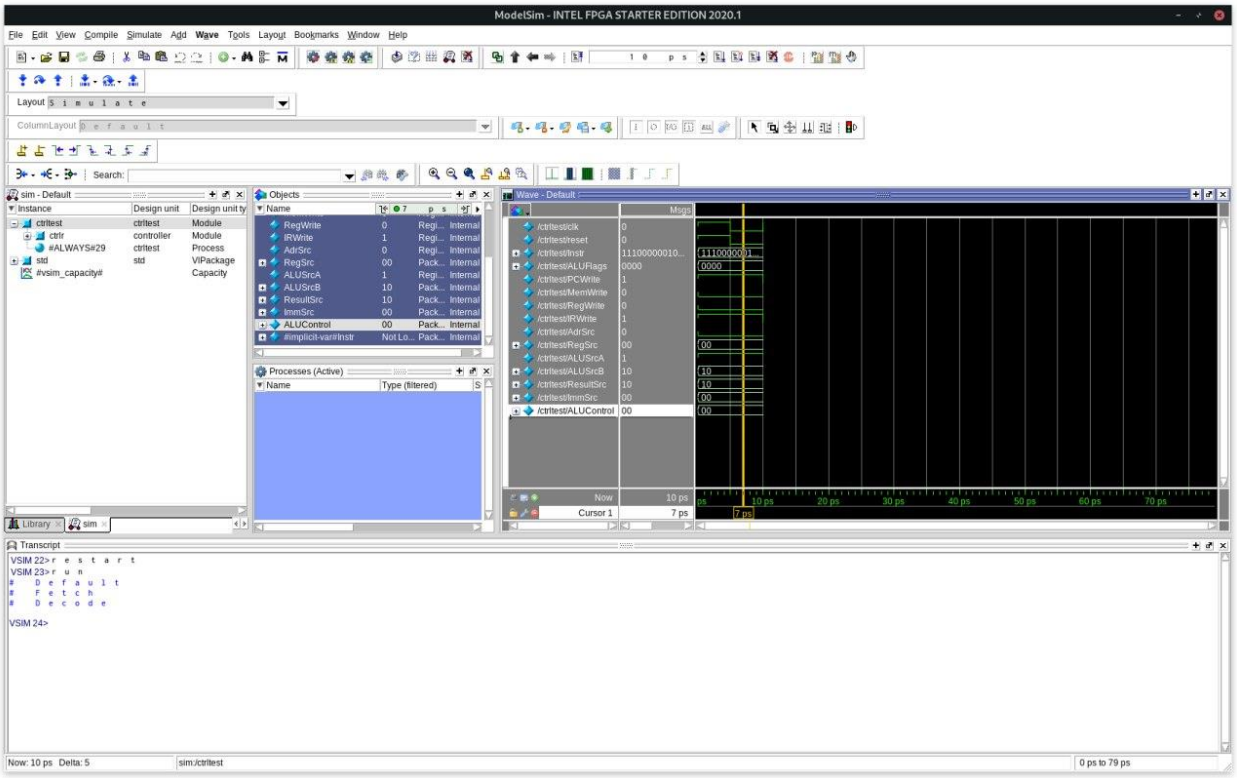
```
module arm(input  logic       clk, reset,
           output logic       MemWrite,
           output logic [31:0] Adr, WriteData,
           input  logic [31:0] ReadData);

  logic [31:0] Instr;
  logic [3:0]  ALUFlags;
  logic        PCWrite, RegWrite, IRWrite;
  logic        AdrSrc, ALUSrcA;
  logic [1:0]  RegSrc,  ALUSrcB, ImmSrc, ALUControl, ResultSrc;

  controller c(clk, reset, Instr[31:12], ALUFlags,
               PCWrite, MemWrite, RegWrite, IRWrite,
               AdrSrc, RegSrc, ALUSrcA, ALUSrcB, ResultSrc,
               ImmSrc, ALUControl);
  datapath dp(clk, reset, Adr, WriteData, ReadData, Instr, ALUFlags,
              PCWrite, RegWrite, IRWrite,
              AdrSrc, RegSrc, ALUSrcA, ALUSrcB, ResultSrc,
              ImmSrc, ALUControl);
endmodule
```

```
module testbench();
	logic clk;
	logic reset;
	logic [31:0] WriteData, DataAdr;
	logic MemWrite;
	// instantiate device to be tested
	top dut(clk, reset, WriteData, DataAdr, MemWrite);
	// initialize test
	initial
	begin
		reset <= 1; # 5; reset <= 0;
	end
	// generate clock to sequence tests
	always
	begin
		clk <= 1; # 5; clk <= 0; # 5;
end
	// check that 7 gets written to address 0x64
	// at end of program
	always @(negedge clk)
		begin
			if(MemWrite) begin
				if(DataAdr === 100 & WriteData === 7) begin
					$display("Simulation succeeded");
					$stop;
				end else if (DataAdr !== 96) begin
					$display("Simulation failed");
					$stop;
				end
			end
		end
endmodule
```

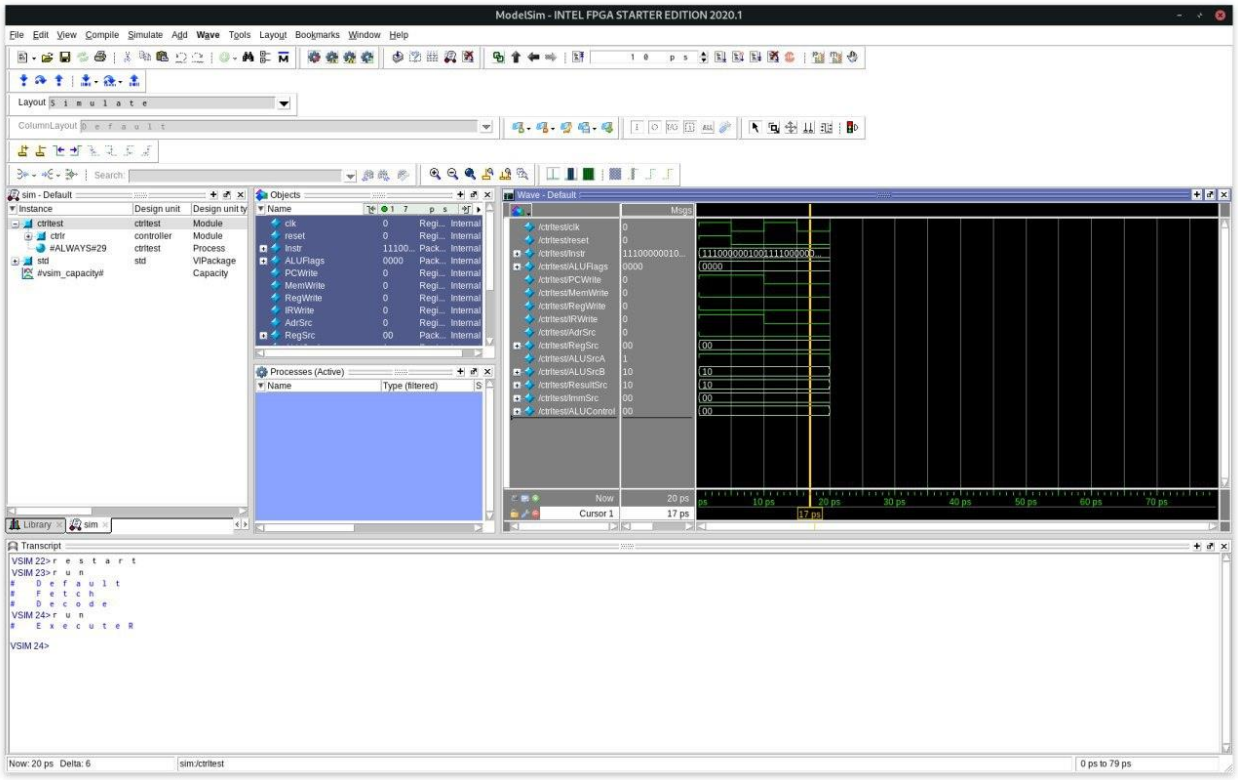
```
module ctrltest();
	logic	clk;
	logic	reset;
	logic [31:0]  Instr;
	logic [3:0]   ALUFlags;
	logic	PCWrite;
	logic	MemWrite;
	logic	RegWrite;
	logic	IRWrite;
	logic	AdrSrc;
	logic [1:0]   RegSrc;
	logic	ALUSrcA;
	logic [1:0]   ALUSrcB;
	logic [1:0]   ResultSrc;
	logic [1:0]   ImmSrc;
	logic [1:0]   ALUControl;

	controller ctrlr(clk, reset, Instr[31:12], ALUFlags, PCWrite, MemWrite, RegWrite, IRWrite, AdrSrc, RegSrc, ALUSrcA, ALUSrcB, ResultSrc, ImmSrc, ALUControl);

	// initialize test
	initial
	begin
		ALUFlags <= 0;
		Instr <= 'hE04F000F;
		reset <= 1; # 5; reset <= 0;
	end

	// generate clock to sequence tests
	always
	begin
		clk <= 1; # 5; clk <= 0; # 5;
	end


endmodule
```

شکل موج ها:

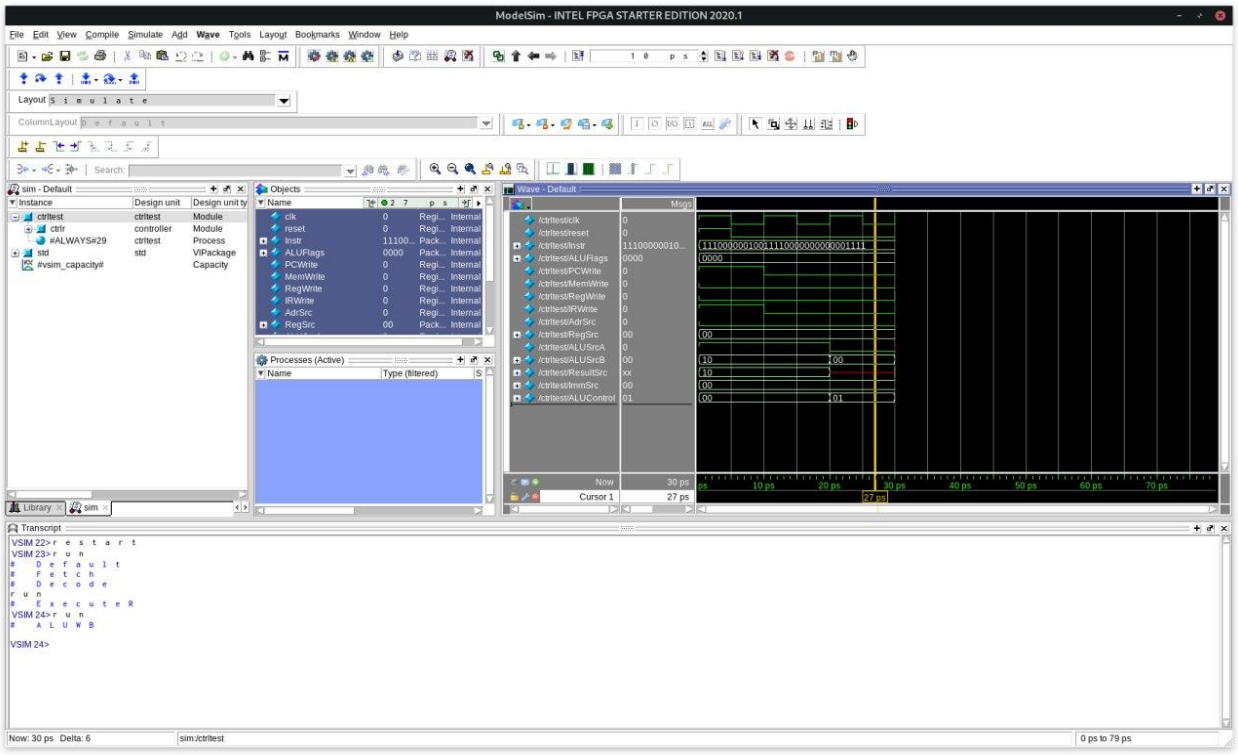
Fetch



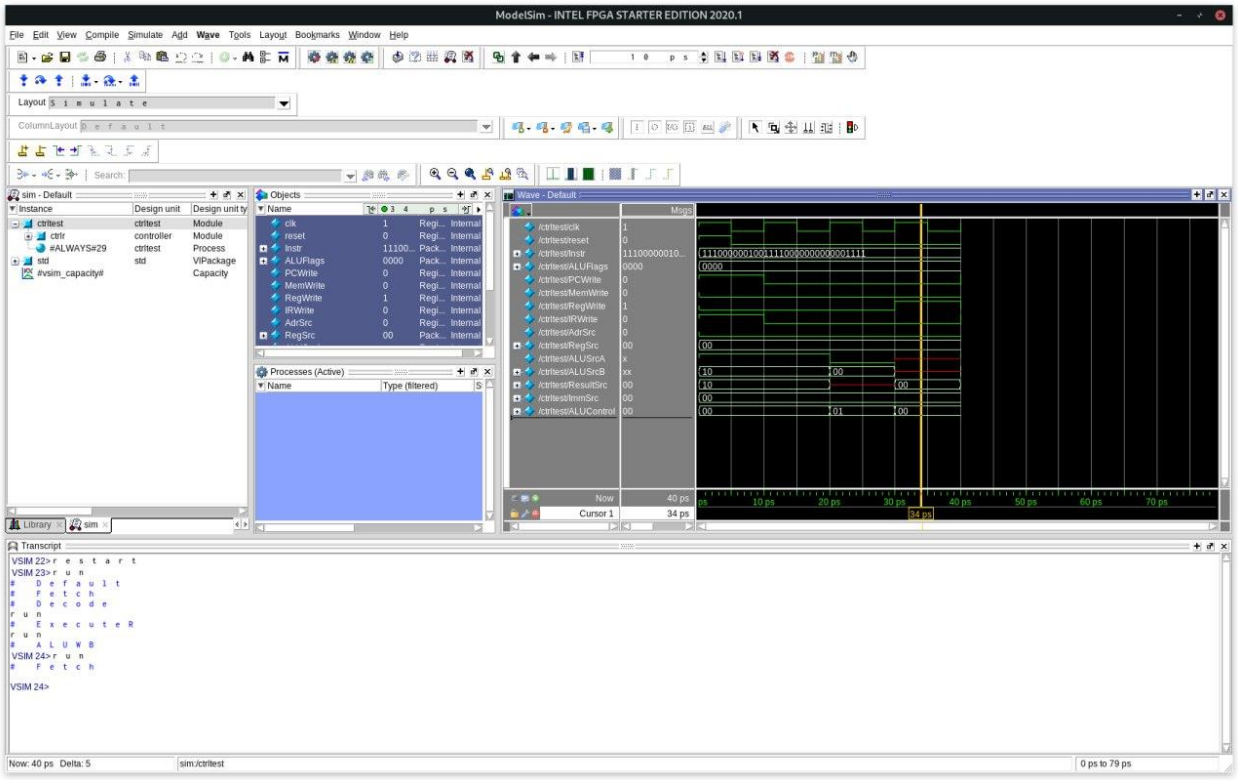
Decode



ExecuteR



ALUWriteback(شکل موج test controller)



تست خود CPU