

پروژه پایانی درس VLSI2 (فاز دوم)

**عنوان:**

Multicycle ARM Processor Control Unit

**استاد درس:**

دکتر شالچیان

**پدیدآورنده:**

سارا قائمی 9223089

28 تیر 96

# فایل های ارائه شده

## فایل گزارش انجام پروژه

این فایل شامل موارد مقابل است. میزان زمانی که برای این پروژه صرف شده است، جدول Expected Instruction Trace که کامل شده است، کد arm\_multi که بخش هایی که به آن اضافه شده است Highlight شده است و تصاویر خروجی مدار که به درستی کار میکند.

## فایل arm\_multi\_9223089.sv

این فایل در واقع کد اصلی برنامه میباشد که بخش های لازم تکمیل شده است. در ادامه به توضیح بیشتر این کد خواهیم پرداخت.

## فایل alu\_9223089.sv

این فایل alu نوشته شده در کارگاه قبلی میباشد که در ادامه به تفصیل توضیح داده خواهد شد.

## فایل testbench.sv

این فایل تست بنچ برنامه میباشد که مشابه آنچه در کارگاه single cycle داشتیم میباشد.

## فایل memfile.dat

این فایل در memory ذخیره میگردد و دستورات به ترتیب اجرا میشوند.

## فایل memfile.asm

از این فایل برای تکمیل جدول استفاده شده است.

## فایل vsim.wlf

این فایل خروجی modelsim را شامل میشود.

# زمان صرف شده

کد اصلی برنامه حدود 30 دقیقه، تکمیل جدول داده شده حدود 1.5 ساعت، دیباگ حدود 1 ساعت و گزارش نیز حدود 2 ساعت زمان برد.

مجموعا من حدود 5 ساعت روی این بخش از پروژه زمان گذاشتم.

# روند انجام پروژه

من این پروژه را دقیقا به ترتیبی که در ادامه می آید انجام دادم.

## نوشتن کد Datapath

با توجه به شکل ARM Multicyle Processor داده شده ابتدا المان های مورد نیاز در این بخش که قبلا آن ها را نداشتیم به پروژه اضافه شدند. همه المان های مورد نیاز در این بخش شامل flip flop با enable، flip flop بدون enable، multiplexer با 3 ورودی، multiplexer با 2 ورودی، register file و بخشextend میباشد. 3 المان اول را قبلا در کد داشتیم و دیگر المان ها را با کمک کد های کارگاه single cycle به صورت زیر به کد اضافه میکنیم.

**module** mux2 **#(parameter** WIDTH **=** 8**)**

**(input** **logic** **[**WIDTH**-**1**:**0**]** d0**,** d1**,**

**input** **logic** s**,**

**output** **logic** **[**WIDTH**-**1**:**0**]** y**);**

**assign** y **=** s **?** d1 **:** d0**;**

**endmodule**

**module** regfile**(input** **logic** clk**,**

**input** **logic** we3**,**

**input** **logic** **[**3**:**0**]** ra1**,** ra2**,** wa3**,**

**input** **logic** **[**31**:**0**]** wd3**,** r15**,**

**output** **logic** **[**31**:**0**]** rd1**,** rd2**);**

**logic** **[**31**:**0**]** rf**[**14**:**0**];**

// three ported register file

// read two ports combinationally

// write third port on rising edge of clock

// register 15 reads PC+8 instead

**always\_ff** **@(posedge** clk**)**

**if** **(**we3**)** rf**[**wa3**]** **<=** wd3**;**

**assign** rd1 **=** **(**ra1 **==** 4'b1111**)** **?** r15 **:** rf**[**ra1**];**

**assign** rd2 **=** **(**ra2 **==** 4'b1111**)** **?** r15 **:** rf**[**ra2**];**

**endmodule**

**module** extend**(input** **logic** **[**23**:**0**]** Instr**,**

**input** **logic** **[**1**:**0**]** ImmSrc**,**

**output** **logic** **[**31**:**0**]** ExtImm**);**

**always\_comb**

**case(**ImmSrc**)**

// 8-bit unsigned immediate

2'b00**:** ExtImm **=** **{**24'b0**,** Instr**[**7**:**0**]};**

// 12-bit unsigned immediate

2'b01**:** ExtImm **=** **{**20'b0**,** Instr**[**11**:**0**]};**

// 24-bit two's complement shifted branch

2'b10**:** ExtImm **=** **{{**6**{**Instr**[**23**]}},** Instr**[**23**:**0**],** 2'b00**};**

**default:** ExtImm **=** 32'bx**;** // undefined

**endcase**

**endmodule**

سپس برای بخش datapath از این المان ها به درستی instance گرفته و در واقع با اینکار سیم های مدار را وصل میکنیم. در ادامه کد این بخش را میبینید که بخش های اضافه شده توسط من در آن highlight شده اند.

**module** datapath**(input** **logic** clk**,** reset**,**

**output** **logic** **[**31**:**0**]** Adr**,** WriteData**,**

**input** **logic** **[**31**:**0**]** ReadData**,**

**output** **logic** **[**31**:**0**]** Instr**,**

**output** **logic** **[**3**:**0**]** ALUFlags**,**

**input** **logic** PCWrite**,** RegWrite**,**

**input** **logic** IRWrite**,**

**input** **logic** AdrSrc**,**

**input** **logic** **[**1**:**0**]** RegSrc**,**

**input** **logic** **[**1**:**0**]** ALUSrcA**,** ALUSrcB**,** ResultSrc**,**

**input** **logic** **[**1**:**0**]** ImmSrc**,** ALUControl**);**

**logic** **[**31**:**0**]** PCNext**,** PC**;**

**logic** **[**31**:**0**]** ExtImm**,** SrcA**,** SrcB**,** Result**;**

**logic** **[**31**:**0**]** Data**,** RD1**,** RD2**,** A**,** ALUResult**,** ALUOut**;**

**logic** **[**3**:**0**]** RA1**,** RA2**;**

//Registers

flopenr **#(**32**)** pcreg**(**clk**,** reset**,** PCWrite**,** PCNext**,** PC**);**

flopenr **#(**32**)** instrreg**(**clk**,** reset**,** IRWrite**,** ReadData**,** Instr**);**

flopr **#(**32**)** datareg**(**clk**,** reset**,** ReadData**,** Data**);**

flopr **#(**32**)** rd1reg**(**clk**,** reset**,** RD1**,** A**);**

flopr **#(**32**)** rd2reg**(**clk**,** reset**,** RD2**,** WriteData**);**

flopr **#(**32**)** alureg**(**clk**,** reset**,** ALUResult**,** ALUOut**);**

//Multiplexers

mux2 **#(**32**)** adrmux**(**PC**,** Result**,** AdrSrc**,** Adr**);**

mux2 **#(**4**)** ra1mux**(**Instr**[**19**:**16**],** 4'b1111**,** RegSrc**[**0**],** RA1**);**

mux2 **#(**4**)** ra2mux**(**Instr**[**3**:**0**],** Instr**[**15**:**12**],** RegSrc**[**1**],** RA2**);**

mux3 **#(**32**)** srcAmux**(**A**,** PC**,** ALUOut**,** ALUSrcA**,** SrcA**);**

mux3 **#(**32**)** srcBmux**(**WriteData**,** ExtImm**,** 32'h0000\_0004**,** ALUSrcB**,** SrcB**);**

mux3 **#(**32**)** alumux**(**ALUOut**,** Data**,** ALUResult**,** ResultSrc**,** Result**);**

//Register File

regfile rf**(**clk**,** RegWrite**,** RA1**,** RA2**,** Instr**[**15**:**12**],** Result**,** Result**,** RD1**,** RD2**);**

//Extend

extend ext**(**Instr**[**23**:**0**],** ImmSrc**,** ExtImm**);**

//ALU

alu alu**(**SrcA**,** SrcB**,** ALUControl**,** ALUResult**,** ALUFlags**);**

**assign** PCNext **=** Result**;**

**endmodule**

## تکمیل جدول Expected Instruction Trace

ابتدا جدول زیر تکمیل گردید. برای این منظور فایل memfile.asm در نظر گرفته شد و با توجه به دانسته های پیشین از نحوه اجرای دستورات، این فایل خط به خط تحلیل گردید.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Cycle | Reset | PC | Instr | (FSM) state | SrcA | SrcB | ALUResult |
| 1 | **1** | **00** | **0** | **FETCH** | **0** | **4** | **4** |
| 2 | **0** | **04** | **SUB E04F000F** | **DECODE** | **4** | **4** | **8** |
| 3 | **0** | **04** |  | **EXECUTER** | **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** |  | **EXECUTEI** | **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 |  | EXECUTEI | 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 |  | EXECUTEI | C | 9 | 3 |
| 16 | 0 | 10 |  | ALUWB | X | X | X |
| 17 | 0 | 10 |  | FETCH | 10 | 4 | 14 |
| 18 | 0 | 14 | ORR E1874002 | DECODE | 14 | 4 | 18 |
| 19 | 0 | 14 |  | EXECUTER | 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 |  | EXECUTER | C | 7 | 4 |
| 24 | 0 | 18 |  | ALUWB | X | X | X |
| 25 | **0** | **18** |  | **FETCH** | **18** | **4** | **1C** |
| 26 | **0** | **1C** | **ADD EO855004** | **DECODE** | **1C** | **4** | **20** |
| 27 | **0** | **1C** |  | **EXECUTER** | **4** | **7** | **B** |
| 28 | **0** | **1C** |  | **ALUWB** | **X** | **X** | **X** |
| 29 | 0 | 1C |  | FETCH | 1C | 4 | 20 |
| 30 | 0 | 20 | SUBS E0558007 | DECODE | 20 | 4 | 24 |
| 31 | 0 | 20 |  | EXECUTER | B | 3 | 8 |
| 32 | 0 | 20 |  | ALUWB | X | X | X |
| 33 | 0 | 20 |  | FETCH | 20 | 4 | 24 |
| 34 | 0 | 24 | BEQ 0A00000C | DECODE | 24 | 4 | 28 |
| 35 | 0 | 24 |  | BRANCH | 28 (PC + 8) | 30 (12\*4) | 58 |
| 36 | 0 | 24 |  | FETCH | 24 | 4 | 28 |
| 37 | 0 | 28 | SUBS E0538004 | DECODE | 28 | 4 | 2C |
| 38 | 0 | 28 |  | EXECUTER | C | 7 | 5 |
| 39 | 0 | 28 |  | ALUWB | X | X | X |
| 40 | 0 | 28 |  | FETCH | 28 | 4 | 2C |
| 41 | 0 | 2C | BGE AA000000 | DECODE | 2C | 4 | 30 |
| 42 | 0 | 2C |  | BRANCH | 30 (PC + 8) | 0 | 30 |
| 43 | 0 | 30 |  | FETCH | 30 | 4 | 34 |
| 44 | 0 | 34 | SUBS E0578002 | DECODE | 34 | 4 | 38 |
| 45 | 0 | 34 |  | EXECUTER | 3 | 5 | FFFFFFFE |
| 46 | 0 | 34 |  | ALUWB | X | X | X |
| 47 | 0 | 34 |  | FETCH | 34 | 4 | 38 |
| 48 | 0 | 38 | ADDLT B2857001 | DECODE | 38 | 4 | 3C |
| 49 | 0 | 38 |  | EXECUTEI | 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 |  | EXECUTER | 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 |  | EXECUTER | 4C | 0 | 4C |
| 67 | 0 | 48 |  | ALUWB | X | X | X |
| 68 | 0 | 4C |  | FETCH | 4C | 4 | 50 |
| 69 | 0 | 50 | B EA000001 | DECODE | 50 | 4 | 54 |
| 70 | 0 | 50 |  | BRANCH | 54 (PC+8) | 4 (1\*4) | 58 |
| 71 | 0 | 58 |  | FETCH | 58 | 4 | 5C |
| 72 | 0 | 5C | STR E5802064 | DECODE | 5C | 4 | 60 |
| 73 | 0 | 5C |  | MEMADR | 0 | 64 | 64 |
| 74 | 0 | 5C |  | MEMWRITE | X | X | X |

جدول شماره 1: Expected Instruction Trace

## کد testbench

همانطور که گفته شده بود برای کد تست بنچ از همان کد کارگاه قبلی استفاده شد و فایل memfile.dat نیز به فولدر برنامه اضافه شد. کد تست بنچ به صورت زیر میباشد.

**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**;** **#** 22**;** reset **<=** 0**;**

**end**

// generate clock to sequence tests

**always**

**begin**

clk **<=** 1**;** **#** 5**;** clk **<=** 0**;** **#** 5**;**

**end**

// check results

**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**

## کد ALU

برای ALU از همان کد کارگاه ALU استفاده گردید که به صورت زیر میباشد.

//`timescale 1ns / 1ps

**module** alu **#(parameter** N **=** 32**)**

**(input** **logic** **[**N**-**1**:**0**]** a**,** b**,**

**input** **logic** **[**1**:**0**]** ALUControl**,**

**output** **logic** **[**N**-**1**:**0**]** Result**,**

**output** **logic** **[**3**:**0**]** ALUFlags**);**

**logic** **[**N**-**1**:**0**]** sum**,** bb**;**

**always\_comb**

**begin**

ALUFlags**[**1**]** **=** 1'b0**;**

**case(**ALUControl**)**

2'b00**:** **{**ALUFlags**[**1**],** Result**}** **=** a **+** b**;**

2'b01**:** Result **=** a **-** b**;**

2'b10**:** Result **=** a **&** b**;**

2'b11**:** Result **=** a **|** b**;**

**endcase;**

ALUFlags**[**3**]** **=** Result**[**31**];**

ALUFlags**[**2**]** **=** **(**Result **==** 0**);**

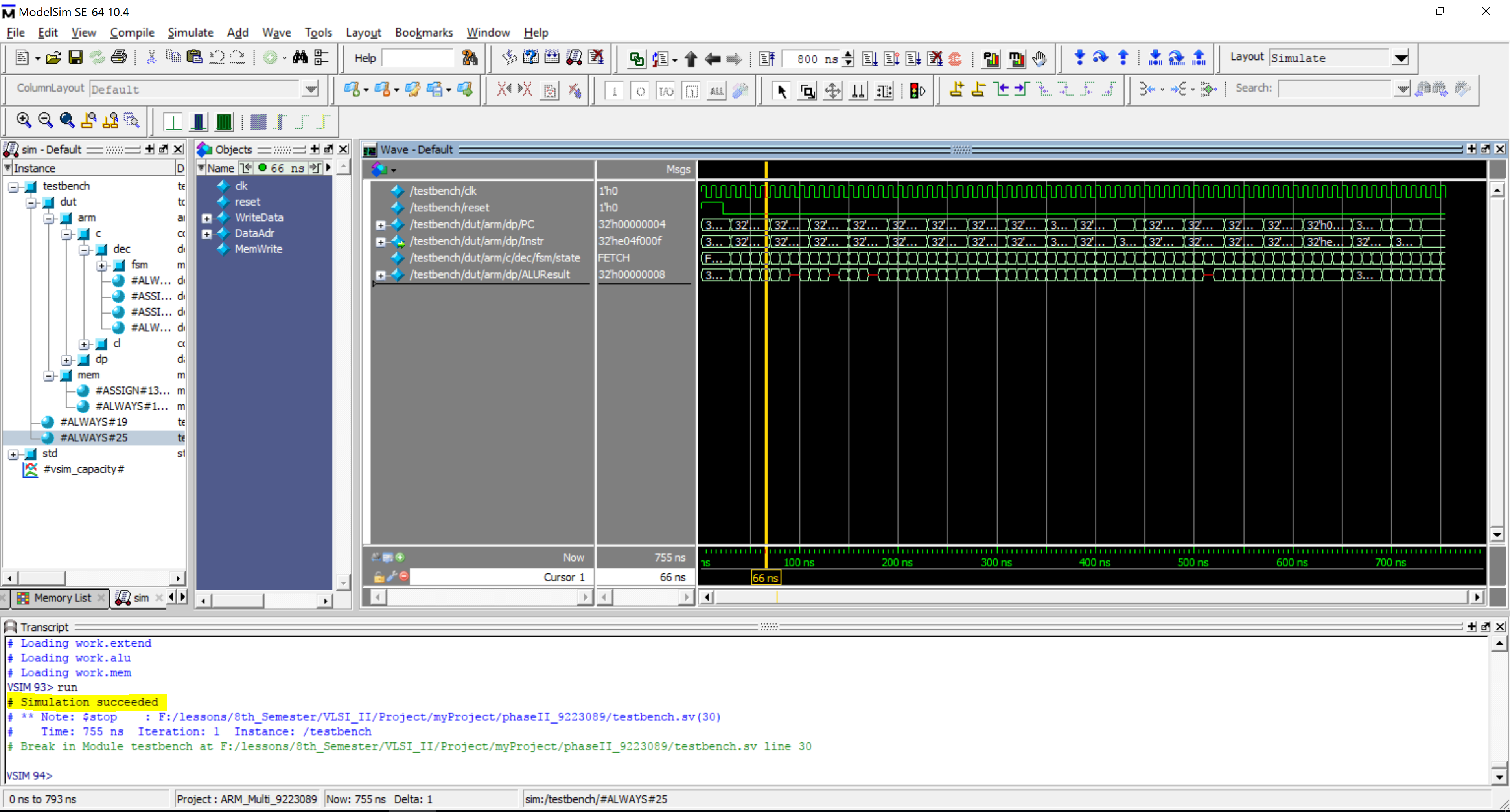
ALUFlags**[**0**]** **=** **(~**ALUControl**[**1**])** **&** **(**Result**[**31**]** **^** a**[**31**])** **&** **(~(**a**[**31**]** **^** b**[**31**]** **^** ALUControl**[**0**]));**

**end**

**endmodule**

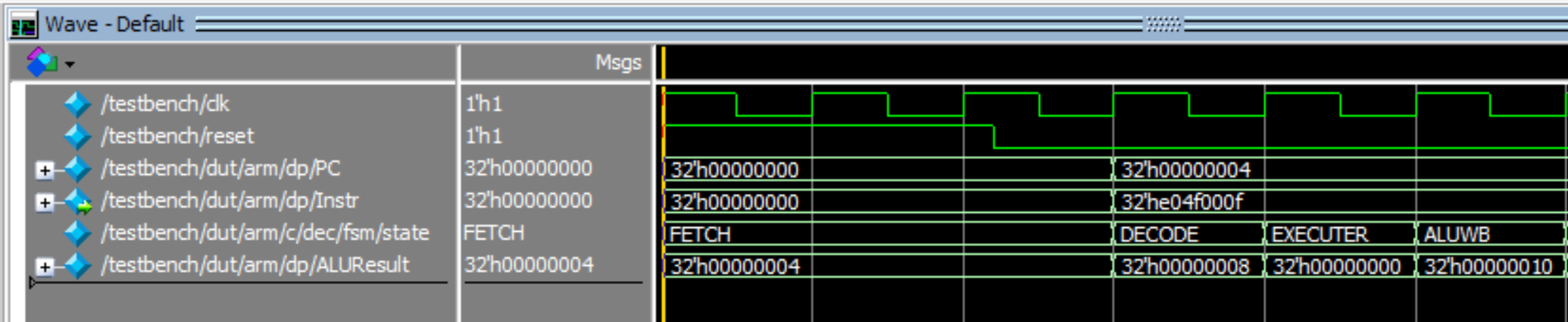
## اجرای شبیه سازی و چک کردن خروجی ها

در انتها کد برنامه را در modelsim ران کرده و سیگنال های گفته شده به ترتیب در آن اضافه گردید. پس از اجرای شبیه سازی پیام simulation succeeded مشاهده گردید که در شکل زیر نیز تصویر آن دیده میشود. که این پیام در آن highlight شده است.

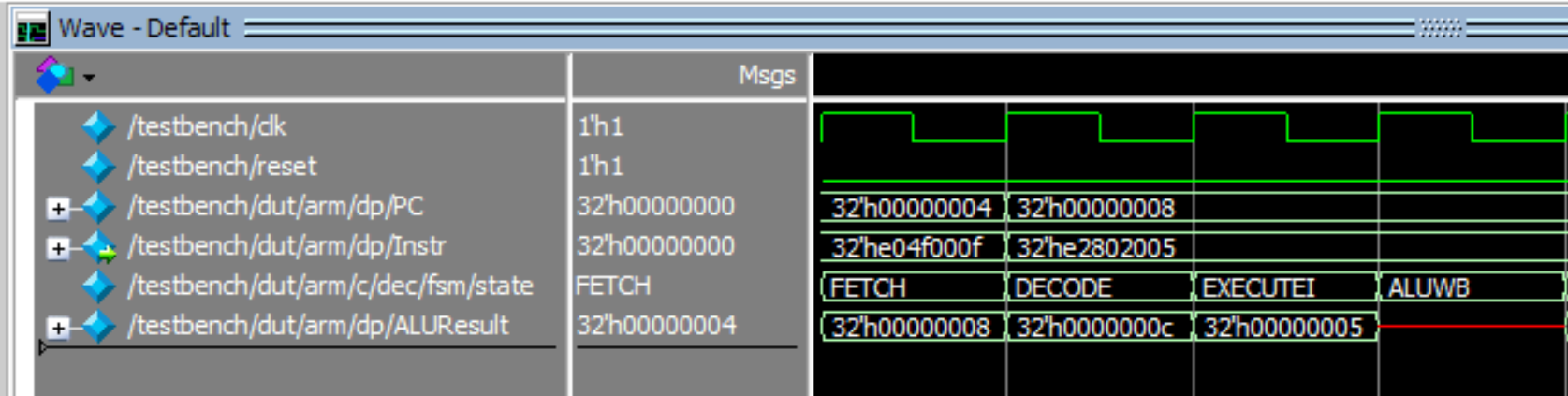


شکل شماره 1: خروجی کلی برنامه

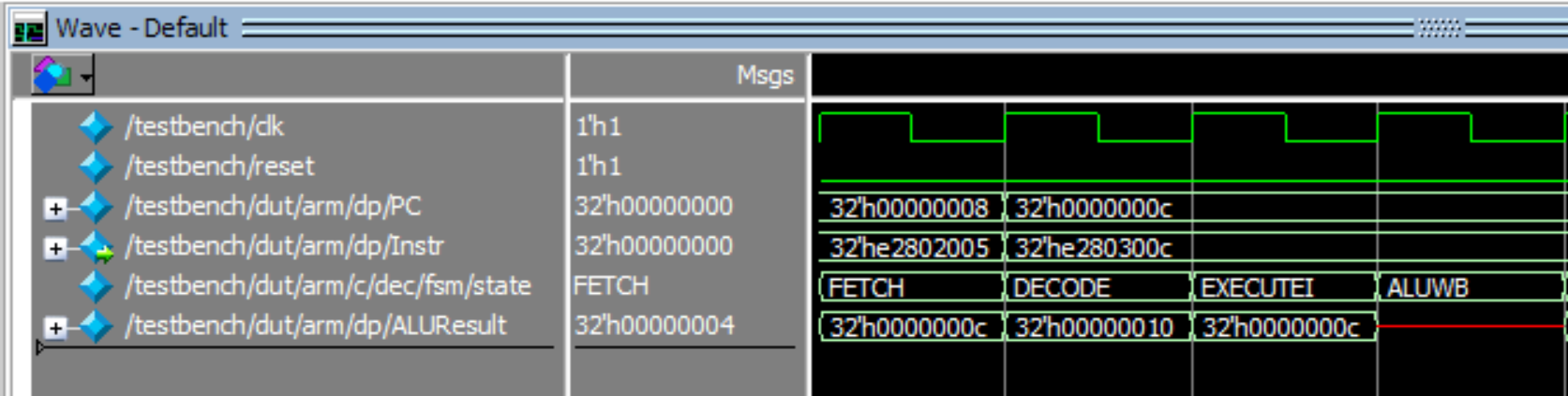
سپس برای اطمینان بیشتر تک تک دستورات جداگانه با جدول فوق چک گردید.



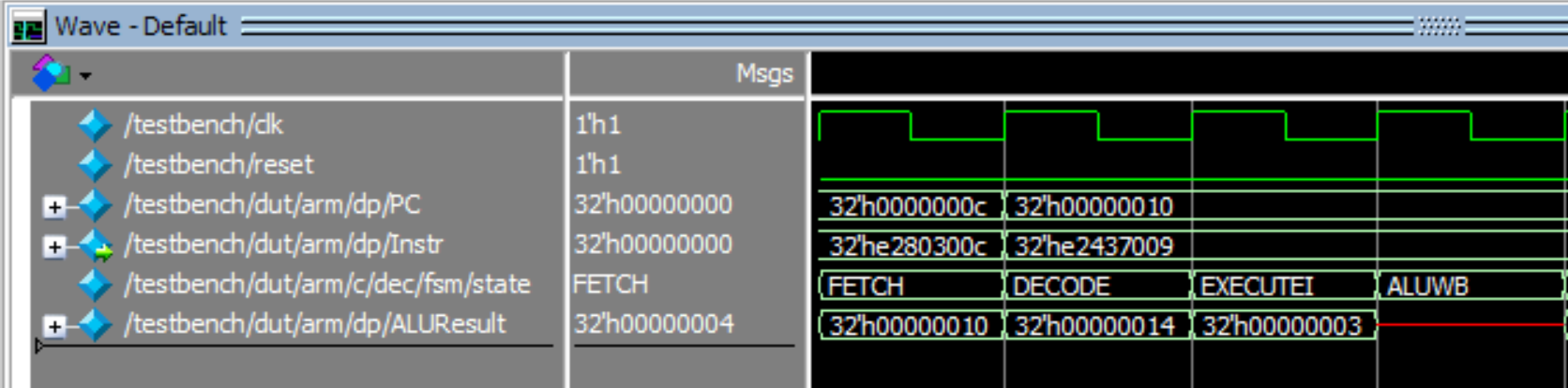
شکل شماره 2: خروجی اجرای کد SUB R0, R15, R15



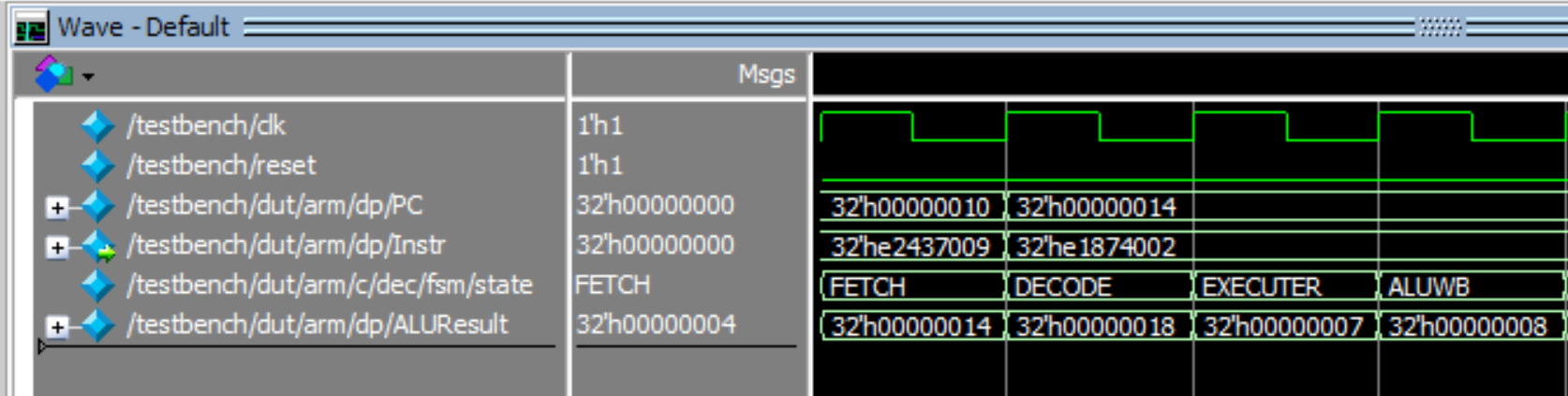
شکل شماره 3: خروجی اجرای کد ADD R2, R0, #5



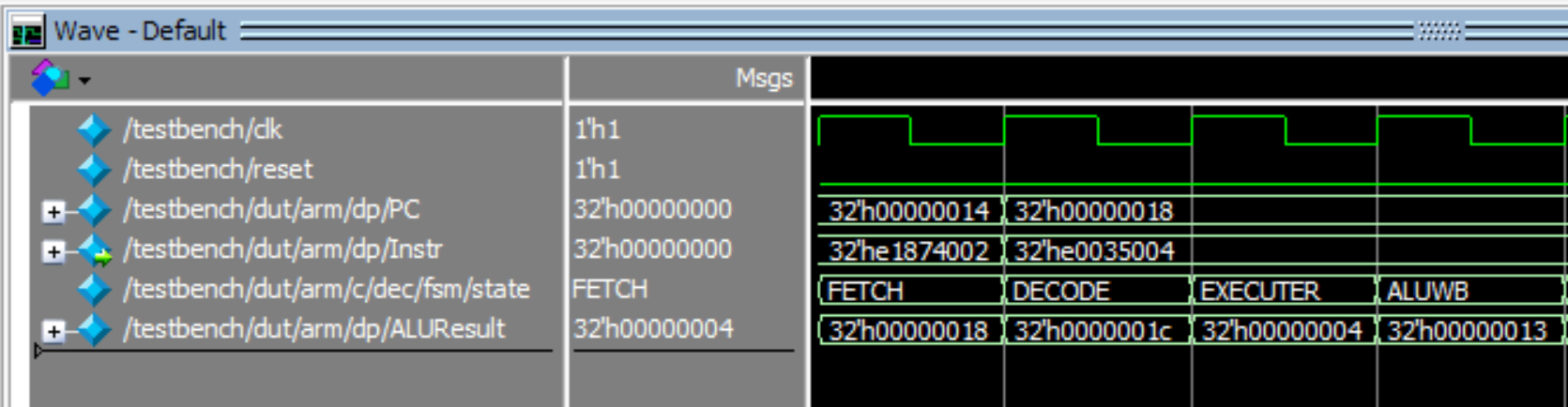
شکل شماره 4: خروجی اجرای کد ADD R3, R0, #12



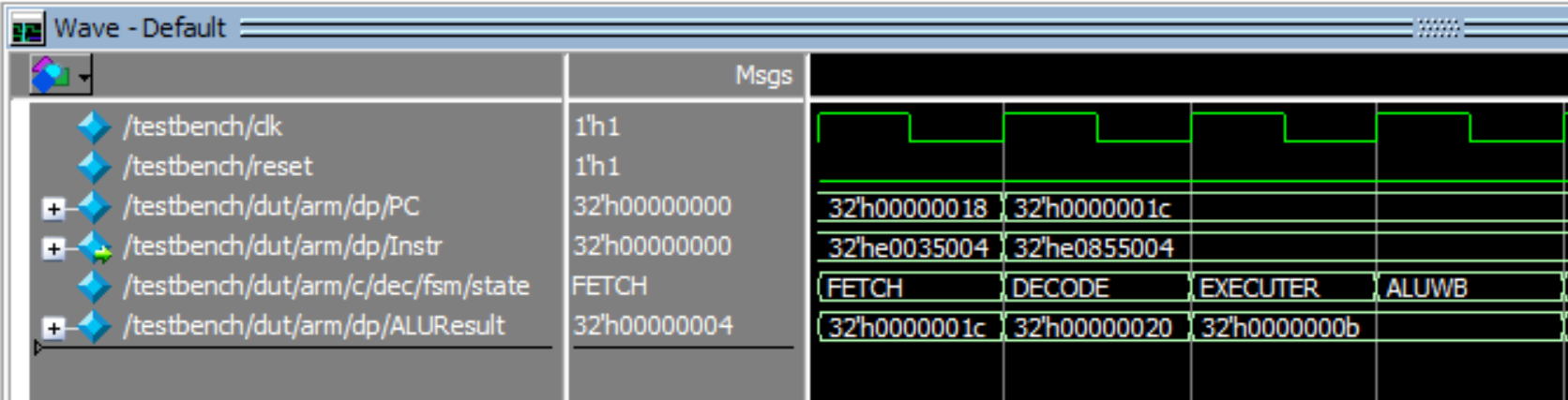
شکل شماره 5: خروجی اجرای کد SUB R7, R3, #9



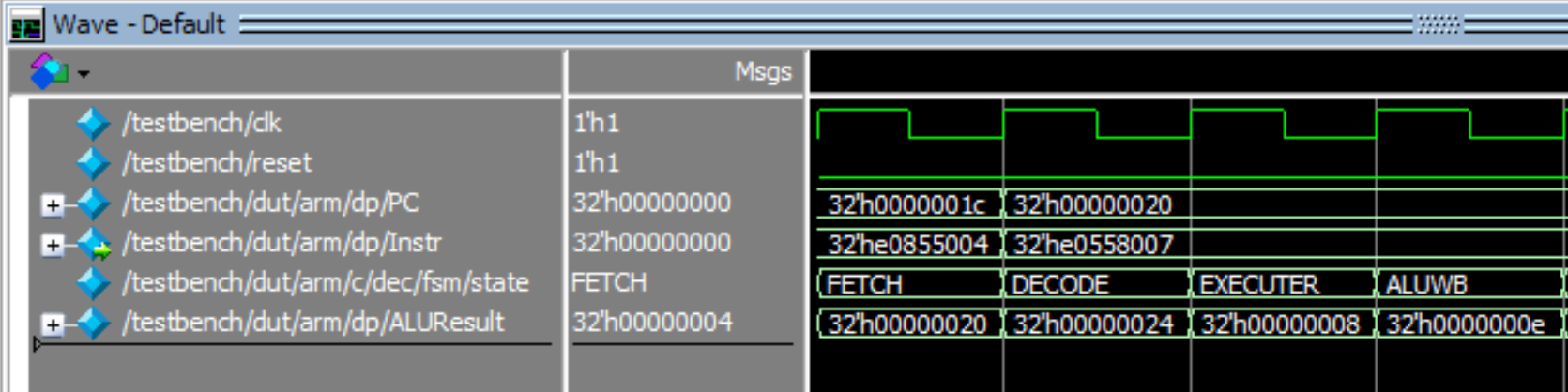
شکل شماره 6: خروجی اجرای کد ORR R4, R7, R2



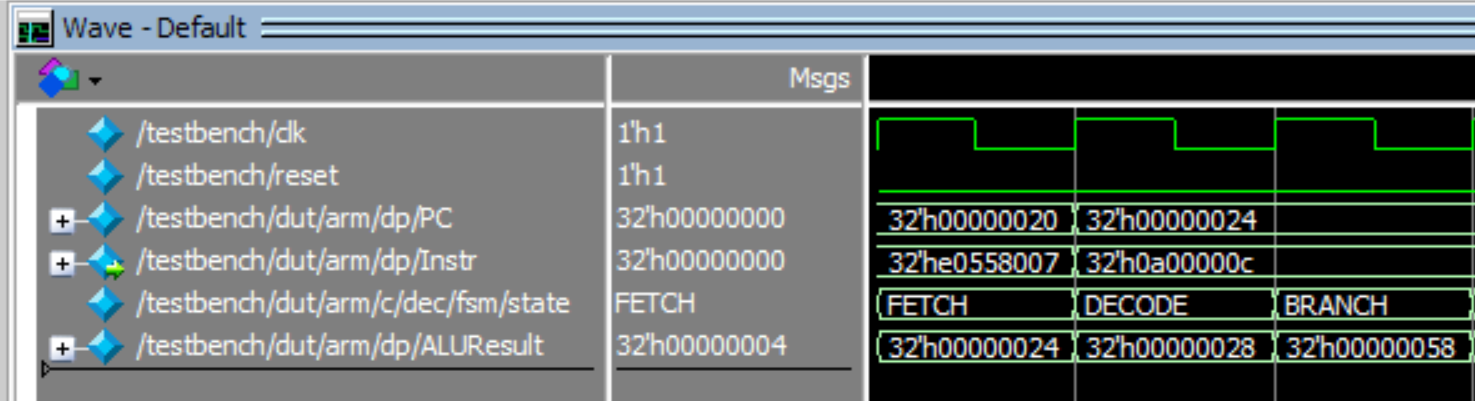
شکل شماره 7: خروجی اجرای کد AND R5, R3, R4



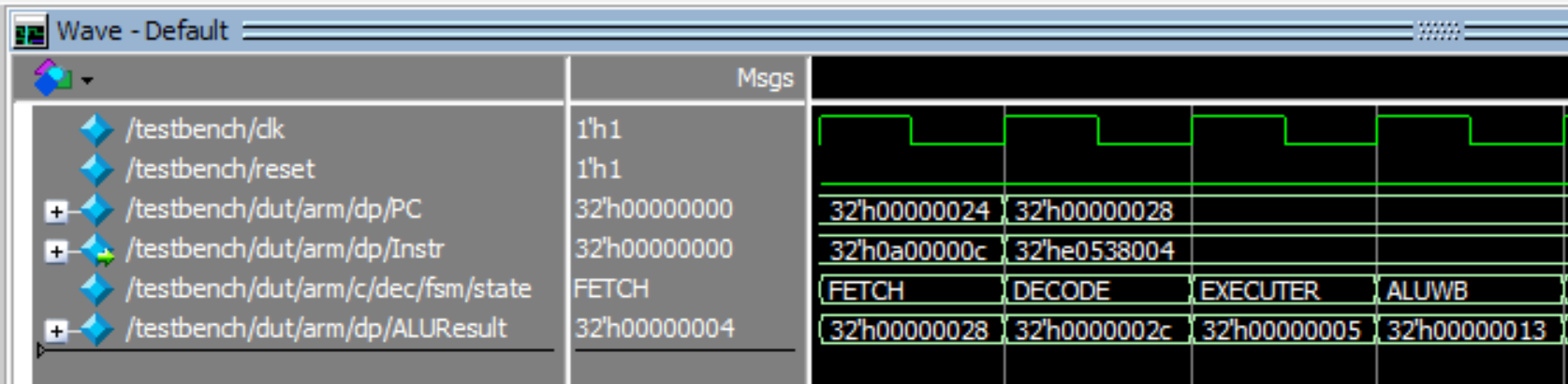
شکل شماره 8: خروجی اجرای کد ADD R5, R5, R4



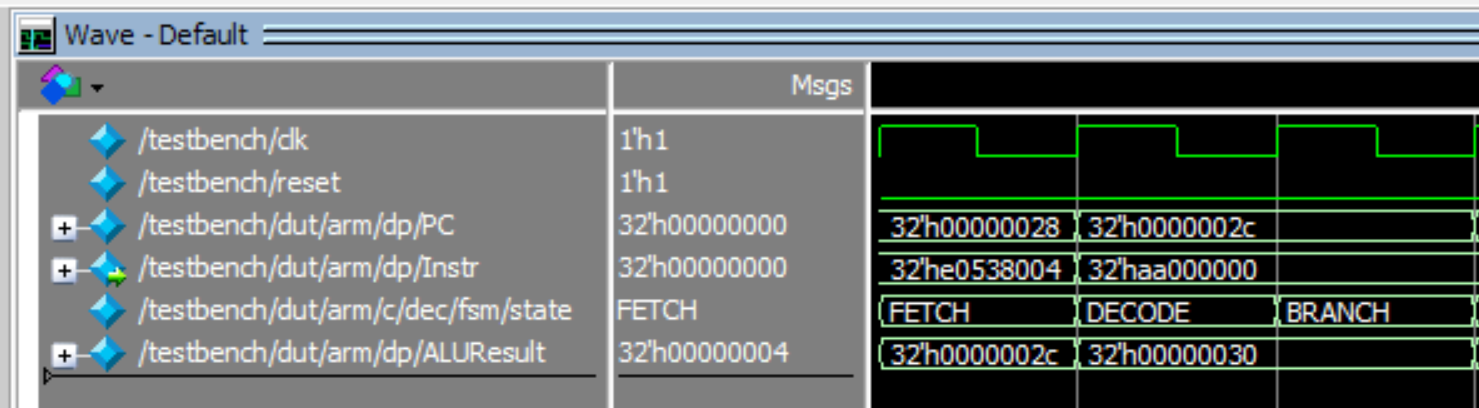
شکل شماره 9: خروجی اجرای کد SUBS R8, R5, R7



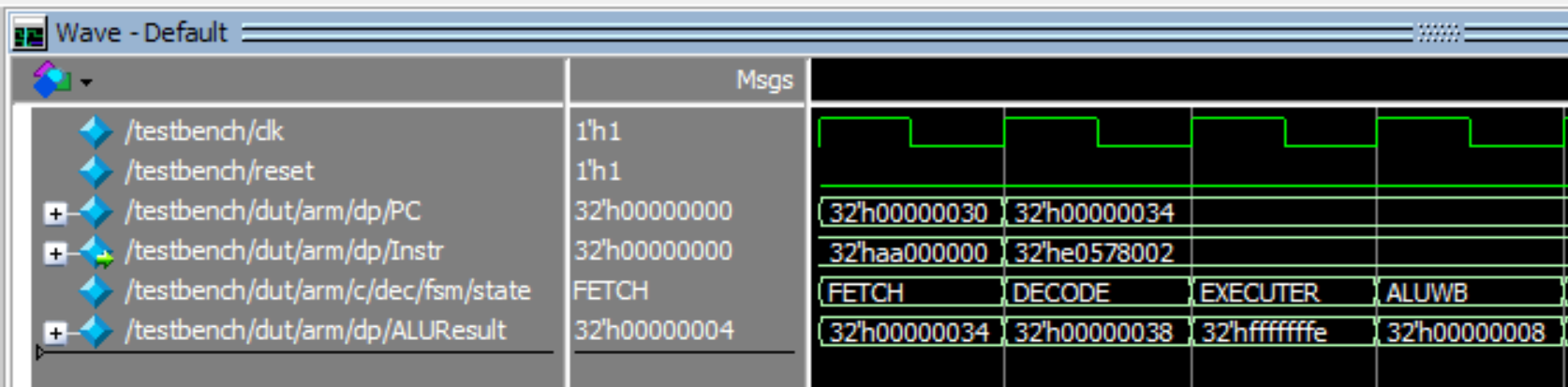
شکل شماره 10: خروجی اجرای کد BEQ END



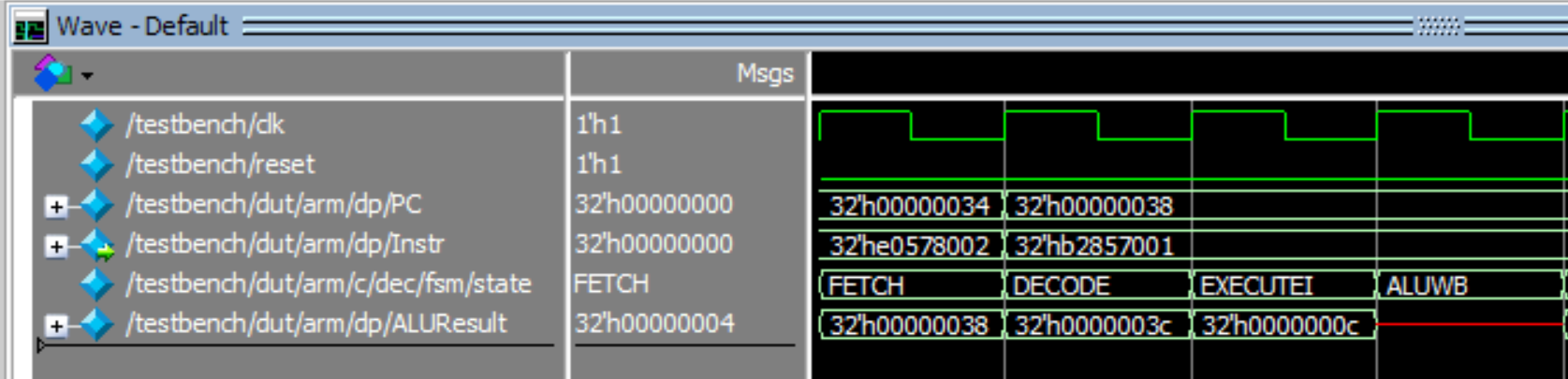
شکل شماره 11: خروجی اجرای کد SUBS R8, R3, R4



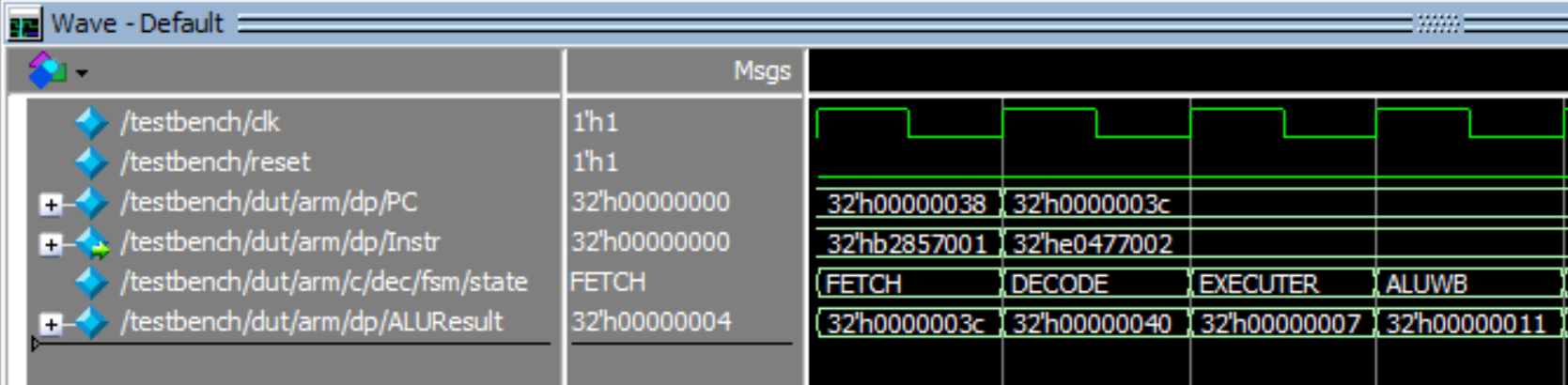
شکل شماره 12: خروجی اجرای کد BGE AROUND



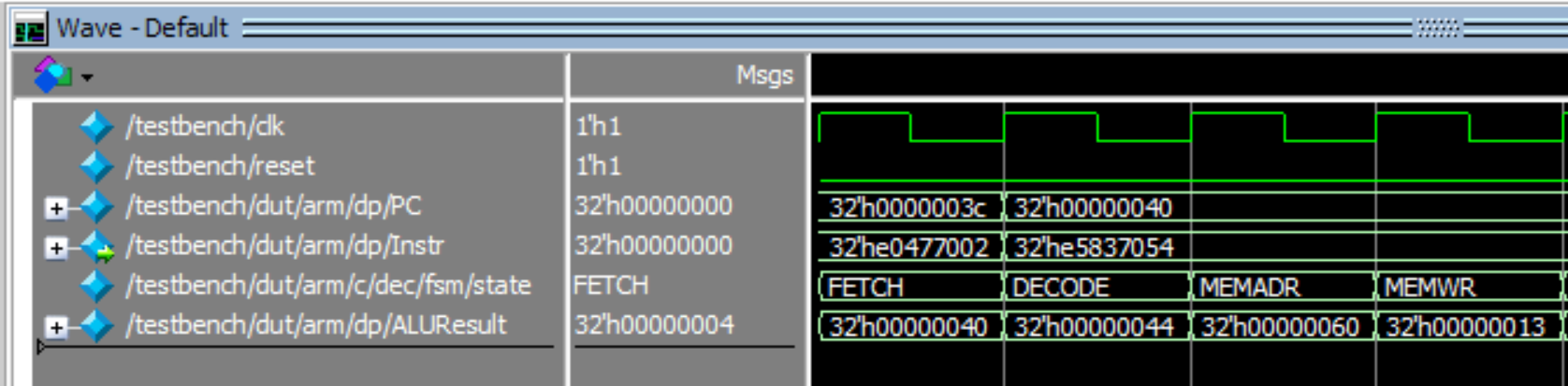
شکل شماره 13: خروجی اجرای کد SUBS R8, R7, R2



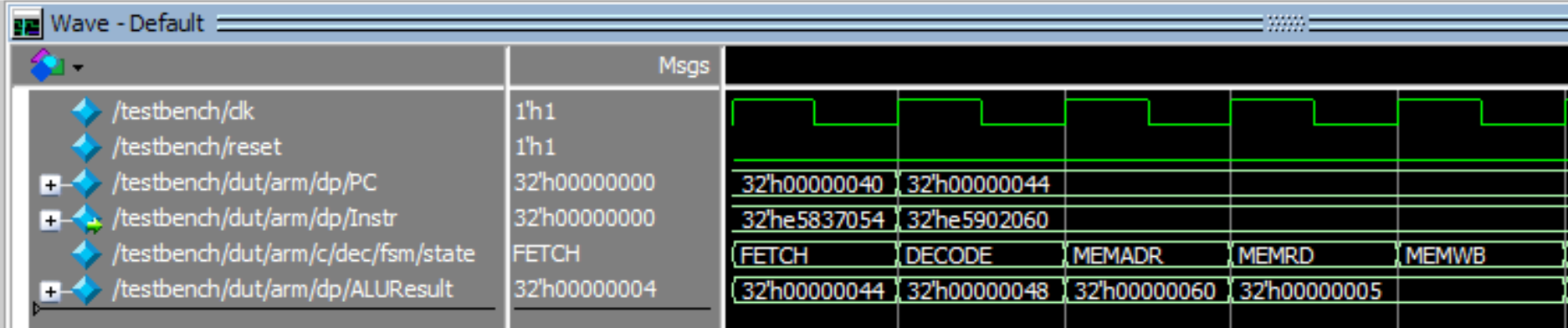
شکل شماره 14: خروجی اجرای کد ADDLT R7, R5, #1



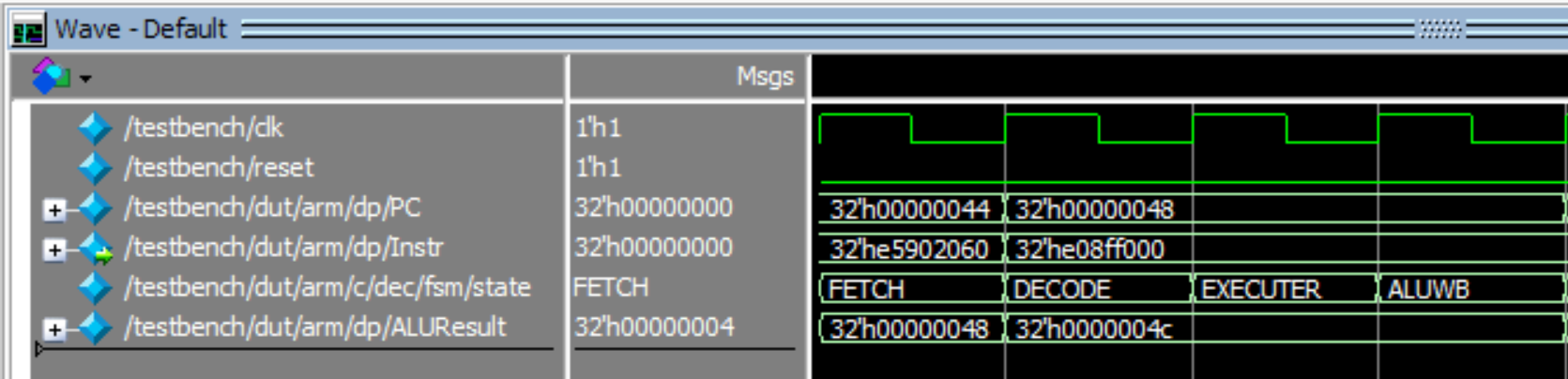
شکل شماره 15: خروجی اجرای کد SUB R7, R7, R2



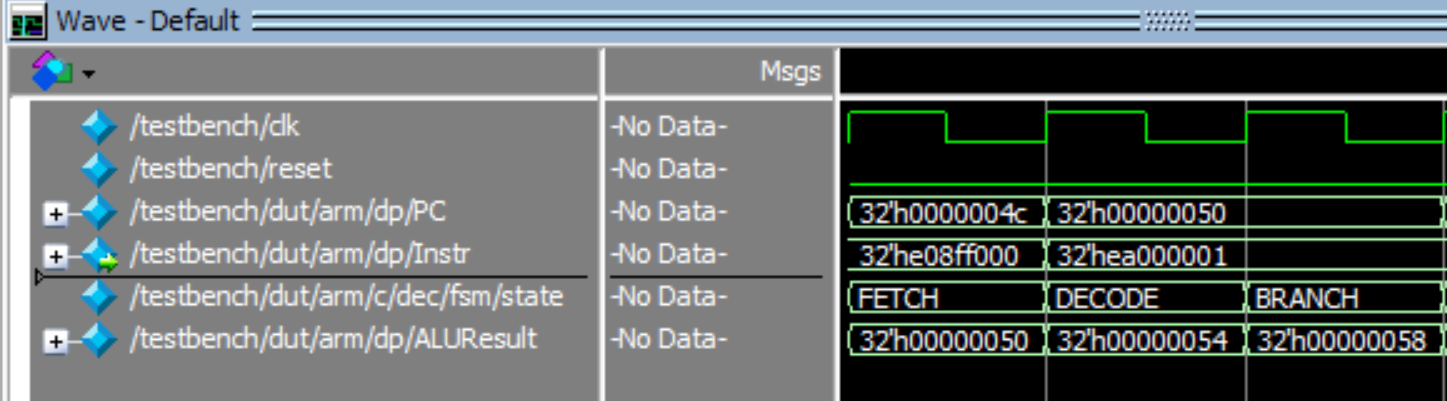
شکل شماره 16: خروجی اجرای کد STR R7, [R3, #84]



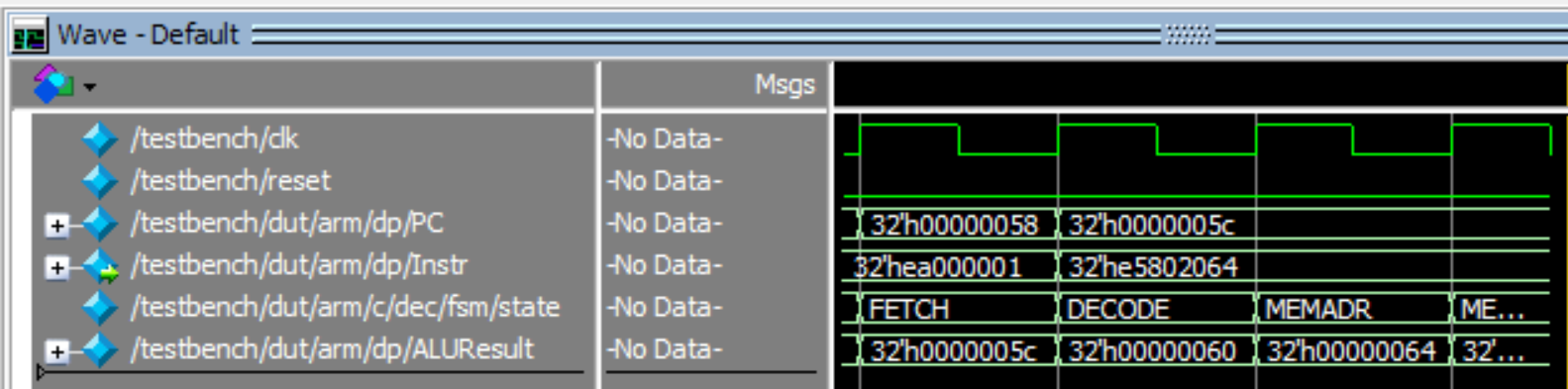
شکل شماره 17: خروجی اجرای کد LDR R2, [R0, #96]



شکل شماره 18: خروجی اجرای کد ADD R15, R15, R0



شکل شماره 19: خروجی اجرای کد B END



شکل شماره 20: خروجی اجرای کد STR R2, [R0, #100]