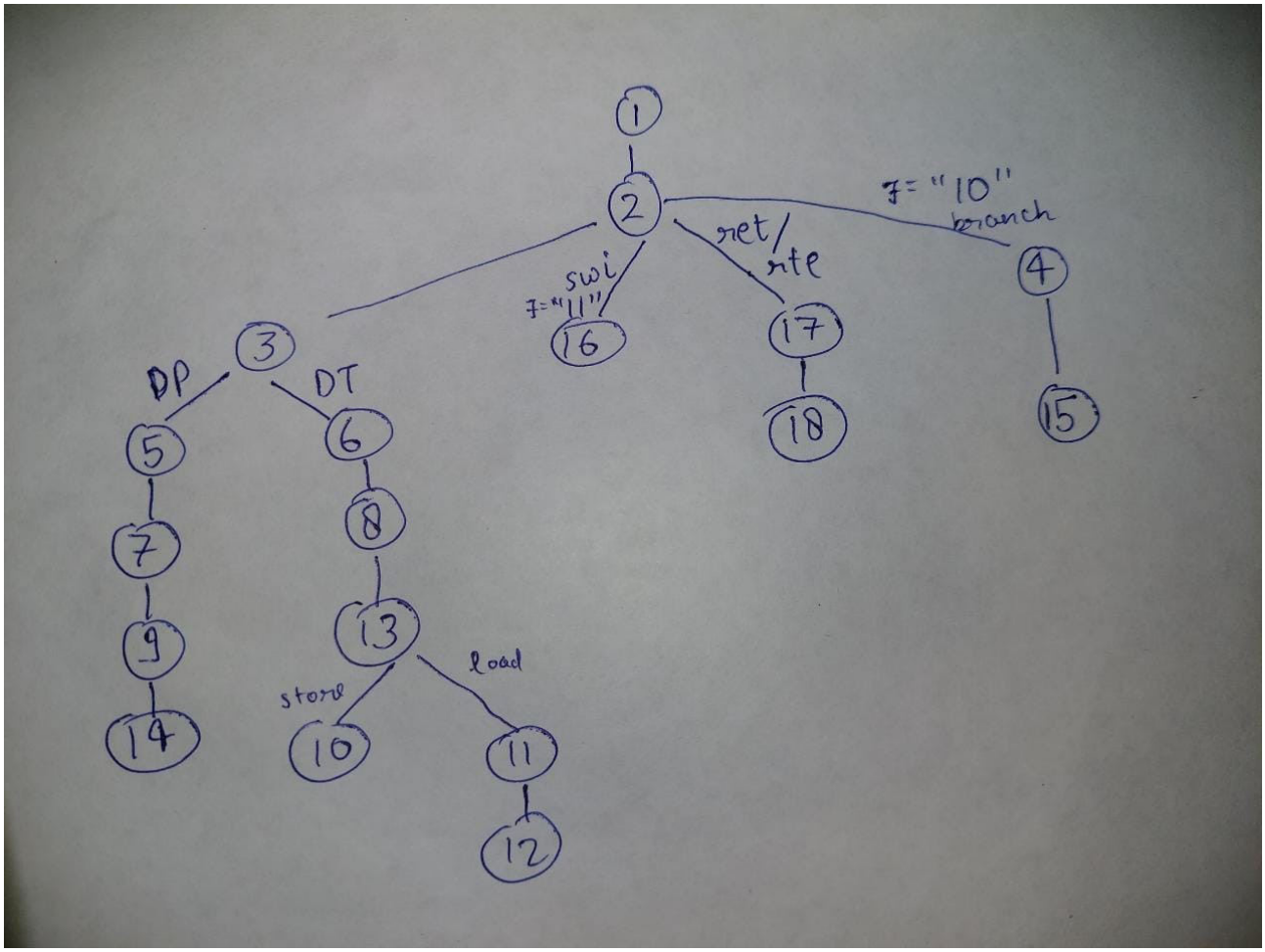
Introduction:

There are 12 files namely alu.vhd, control_fsm.vhd, data_memory.vhd, flags.vhd, register_file.vhd, stage8.vhd, shifter.vhd, Pmconnect.vhd, Mul.vhd, testReset.vhd, testSwi.vhd and testbench.vhd.

alu.vhd:	handles all the arithmetic operations and also incrementes pc.
data_memory.vhd:	reads and write in the program memory and the data memory. The size of data_memory is 512x32.
flags.vhd:	it sets the C, N, V and Z flags.
register_file.vhd:	it reads and write in the 16 registers of the program.
control_fsm.vhd:	it sets all the control signals depending on the current state and update the state also.
stage8.vhd:	this is glue code for this stage.
shifter.vhd:	this handles all the shift operations. data, amount and type of shift are its input and it outputs final data and carry.
Pmconnect.vhd	Connector the register file and data memory.
Mul.vhd	Handle all 6 types of multiplications.
testReset.vhd	to test reset command.
testSwi.vhd	to test swi command.
testbench.vhd	to test the code.

- To run new code paste instruction set in data memory.
- Instructions start from 128 before it is system area.
- I have added 4 new states to fsm, now it has total 18 states.
- All the states are explained on next page/.
- New control signals are
    - Link: Signals when to store pc into Lr.
    - change_mode: Signals when to change mode from user to superviser and when superviser to user.



States:

1: IR = Mem[PC], PC = PC+4
2: A = RF[IR[19-16]], B = RF[IR[3-0]]
3: Read Rm (IR[3-0]) or Rs (IR[11-8]) and store in C. Read Rs and store in mul_Rs.
4: PC = PC + S2(IR[23-0])+4 depending on predicate.
5: Shift for DP instruction.
6: Shift for DT instruction.
7: Res,Flags = ALU(A,B,C,IR[24-21]) and do multiplication opeartions.
8: Res = A ± ex(IR[11-0])
9: RF[IR[19-16]] = mul_result(63 downto 32) when multiplication operation and RF[IR[15-12]] = Res for rest of DP instructions.
10: Mem[Res] = B depending on predicate and Pmconnect for store.
11: DR = Mem[Res]
12: RF[IR[15-12]] = DR depending on predicate and Pmconnect for store.
13: Write back in Rn if needed.
14: RF[IR[15-12]] = mul_result(31 downto 0) when multiplication operation.
15: If link = 1 (in bl) then store pc into Lr (register 14).
16: For swi instructions it stores pc into Lr (register 14) and changes mode.
17: Read register 14 (Lr).
18: Move Lr into pc and change mode back to user mode.

Test done:
1: for testing bl,

I have majorly tested the commands for bl in this test.

ARMSim code:	Result of alu
mov r0, #4	
mov r1, #1	
mov r2, #0	
b A	
B:add r1, r1, #1	1 + 1 = 2
add r2, r2, #2	0 + 2 = 2
ret	
A:add r0, r0, #4	4 + 4 = 8
bl B	
add r3, r1, r2	2 + 2 = 4
mov r2, #0	
add r4, r0, r2	8 + 0 = 8

Instruction set for Data Memory:

```
  128 => X"E3A00004",
  129 => X"E3A01001",
  130 => X"E3A02000",
  131 => X"EA000002",
  132 => X"E2811001",
  133 => X"E2822002",
  134 => X"E6000010",
  135 => X"E2800004",
  136 => X"EBFFFFFA",
  137 => X"E0813002",
  138 => X"E3A02000",
  139 => X"E0804002",
  others => X"00000000");
```

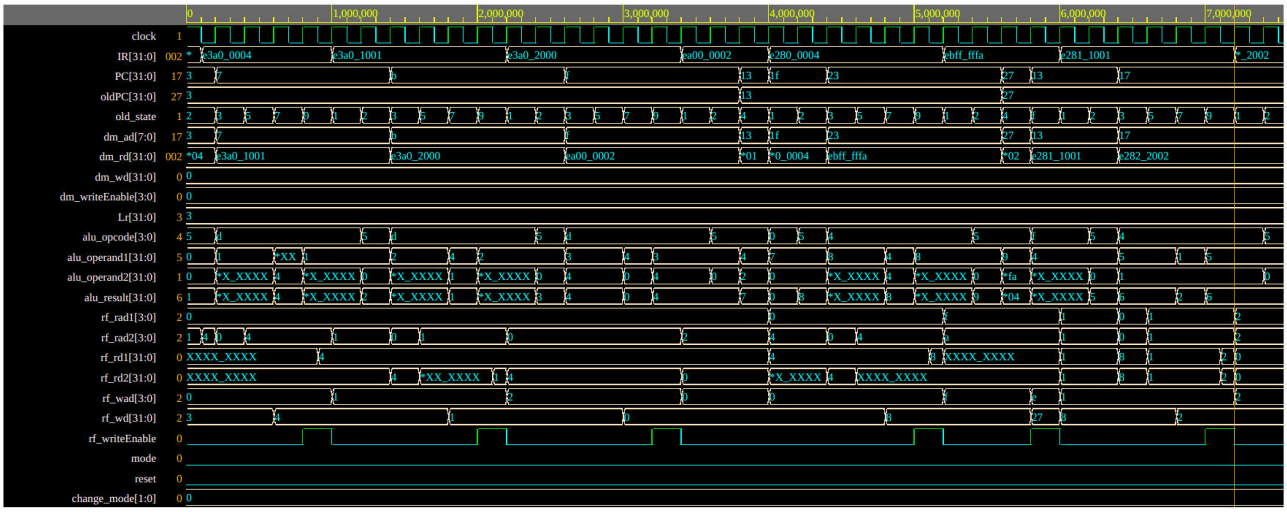
Order in which instructions should be implemented:

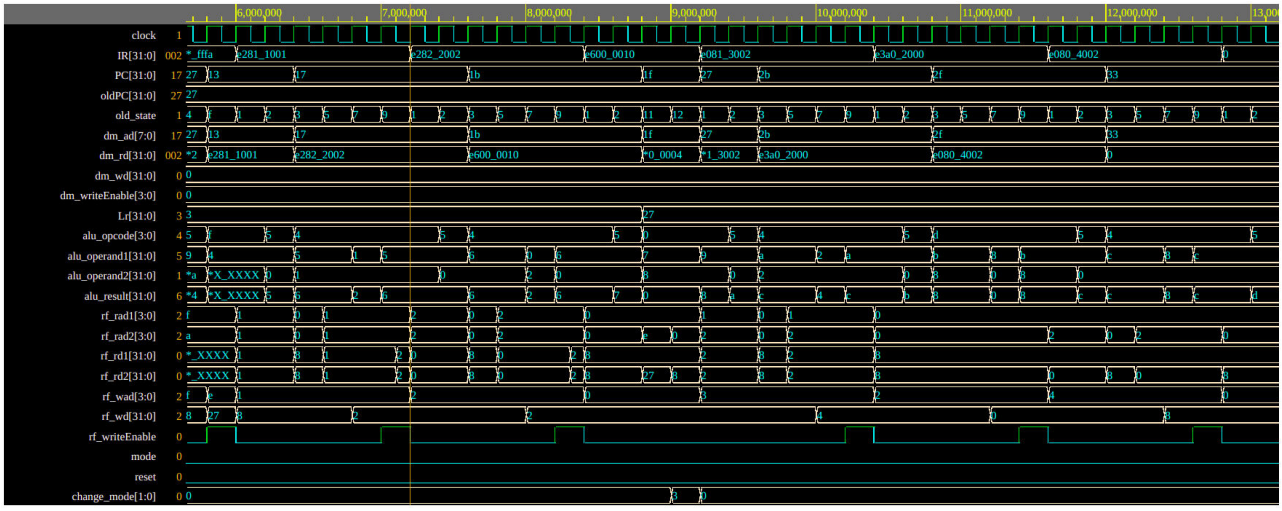
X"E3A00004", X"E3A01001", X"E3A02000", X"EA000002", X"E2800004", X"EBFFFFFA", X"E2811001", X"E2822002", X"E6000010", X"E0813002", X"E3A02000", X"E0804002"

The result in alu and order of instructions are as expected.

EPWave:

- The images are on the right side of the previous image
- Only important signals are shown.



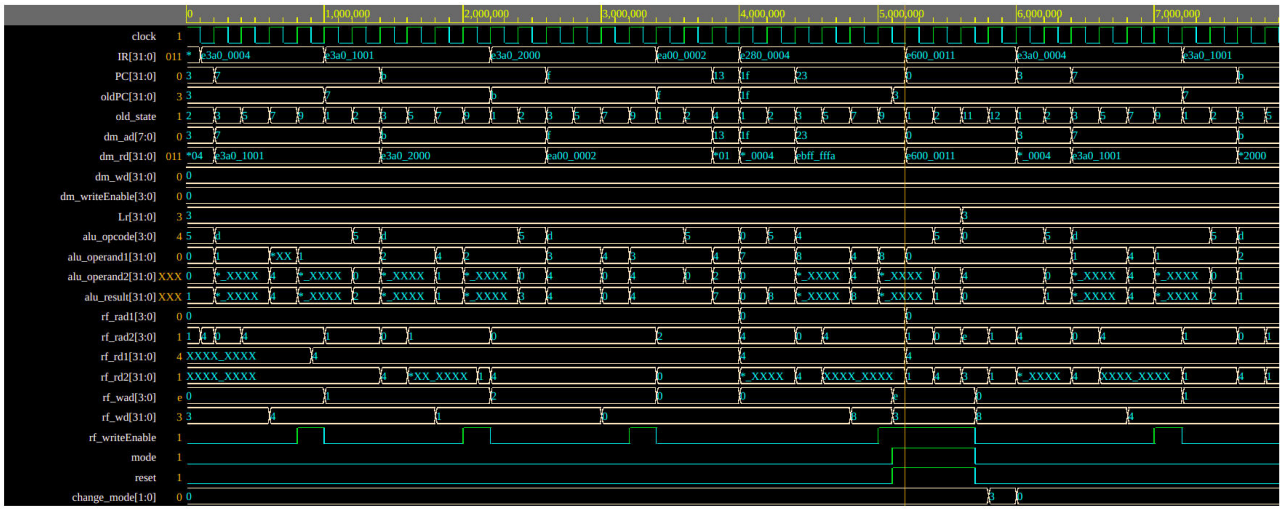


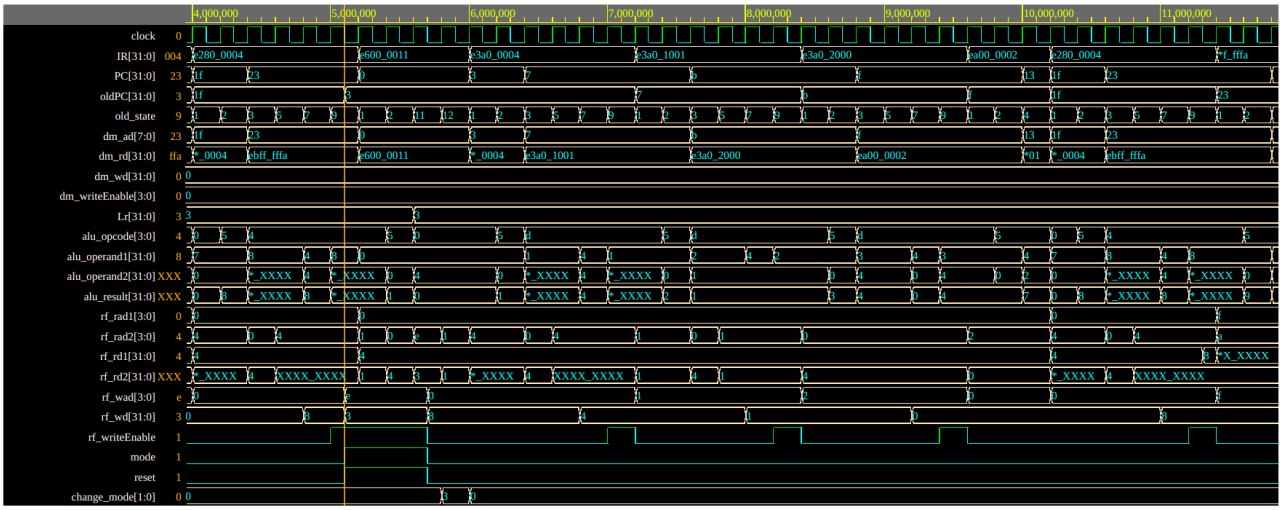
Test done:
2: for testing reset,

I have tested the reset command in this test. Reset has to be done in the testbench.
I have uploaded a separate testbench to test reset command.

EPWave:

Commands in data memory were same as previous test.





Test done:
3: for testing flags,

I have added and tested full prediction in stage 4. So in this test I have majorly tested the full flag module

ARMSim code:	Flags Value (N, Z, C, V)
mov r0, #4	
mov r1, #1	
adds r2, r1, r0	0, 0, 0, 0
sub r3, r2, r0	0, 0, 0, 0
cmp r3, r2	1, 0, 0, 0
and r4, r3, r2, LSL #2	0, 1, 0, 0
orr r5, r2, r3	0, 0, 0, 0
tst r2, r3	0, 0, 0, 0

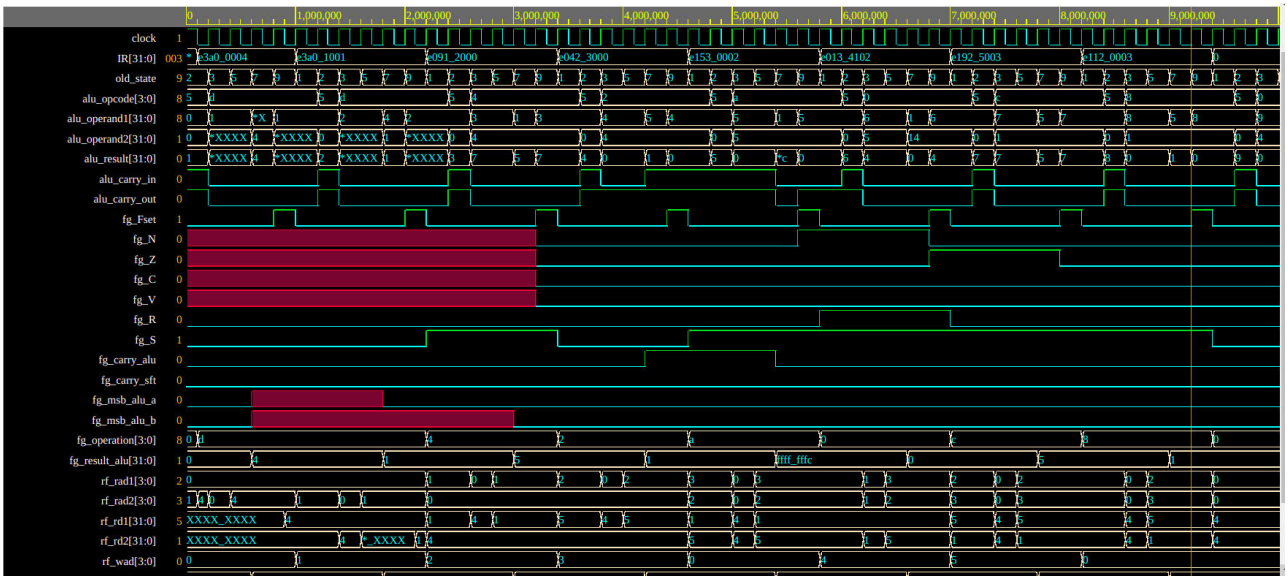
Instruction set for Data Memory:

```
128 => X"E3A00004",
129 => X"E3A01001",
130 => X"E0912000",
131 => X"E0423000",
132 => X"E1530002",
133 => X"E0134102",
134 => X"E1925003",
135 => X"E1120003",
others => X"00000000" );
```

The value of flags by ARMSim and my code are both same, which could be seen in screenshot of EP Wave.
The value of flags get updated in state 9.
The flags value will be undefined for first 2 mov commands.

EPWave:

- Only important signals are shown.



Test done:
4: for testing swi,

I have tested the input swi in this test.

ARMSim code:	Result of alu
mov r0, #4	
mov r1, #1	
swi 0	(input = 5 in testbench)
add r3, r0, r1	1 + 5 = 6

If in last command alu output is 6 then input (assume input = 5) is taken correctly and my code gives the correct ouput.

Instruction set for Data Memory:

```
128 => X"E3A00004",
129 => X"E3A01001",
130 => X"EF000000",
131 => X"E0803001",
others => X"00000000" );
```

- After encoutering swi command (X"EF000000") the mode changes = 1 and 2 instructions from ISR are implemented.
- First intsruction stores the input in register r0.
- Second instruction load Lr into PC and changes the mode back to 0.

EPWave:

- Only important signals are shown.