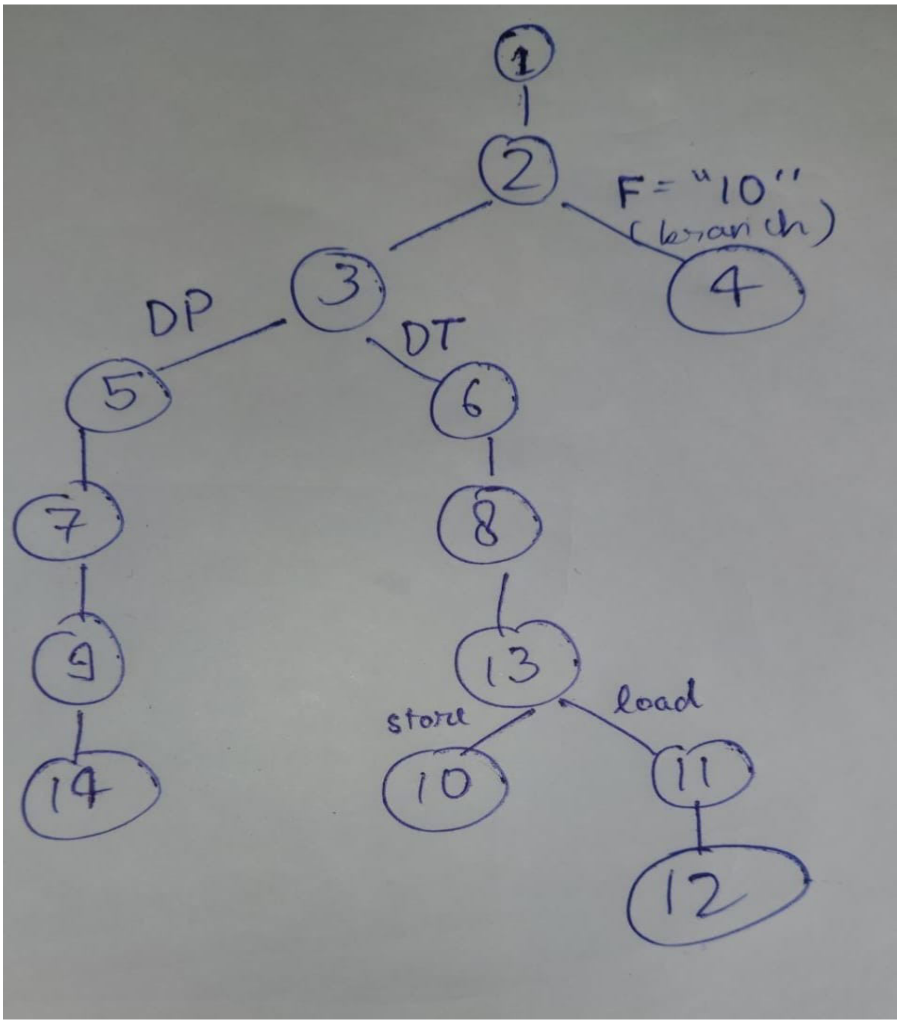
Introduction:

There are 11 files namely alu.vhd, control_fsm.vhd, data_memory.vhd, flags.vhd, register_file.vhd, stage5.vhd, shifter.vhd, Pmconnect.vhd, Mul.vhd, testmul.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 128x32 out of which first half stores program memory and second half stores data memory.
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.
stage6.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.
testmul.vhd	to test multiplication operations.
testbench.vhd	to test the code.

- To run new code paste instruction set in data memory.
- I have added 1 new state to fsm, now it has 14 states.
- The new state(14) added write back into the register for 64 bit multiplication instrustions if needed.
- New control signals are
    - MulIns: stores instruction to be passed to mul.
    - EW: control signal to store value at register Rs in cycle 3.
    - Rsrc1: Control signal for multiplexer before rf_rad1.



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.

Test done:

I have majorly tested the commands related to this stage only as my code for previous stages was working fine.

ARMSim code:	Result of mul operation
mov r0, #4	
mov r1, #1	
mov r2, #0xff	
mov r2, r2, ROR #0x1	
mov r3, #3	
mul r4, r0, r1	1 * 4 = 4
mla r5, r0, r1, r3	1 * 4 + 3 = 7
smull r6, r7, r2, r3	8000007f * 3 = fffffffe8000017d
umull r6, r7, r2, r3	8000007f * 3 = 18000017d
smlal r0, r1, r2, r3	8000007f * 3 + 100000004 = fffffffe80000181
mov r0, #4	
mov r1, #1	
umlal r0, r1, r2, r3	8000007f * 3 + 100000004 = 280000181

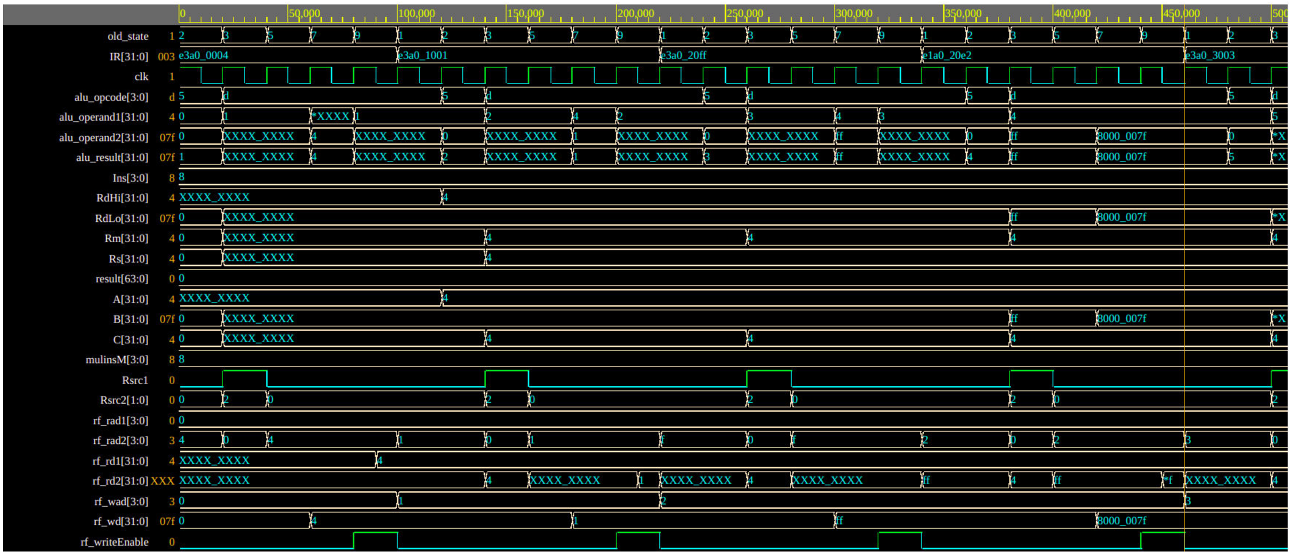
(On right side numbers are in hex. The result is same in ARMSim and my processor.)

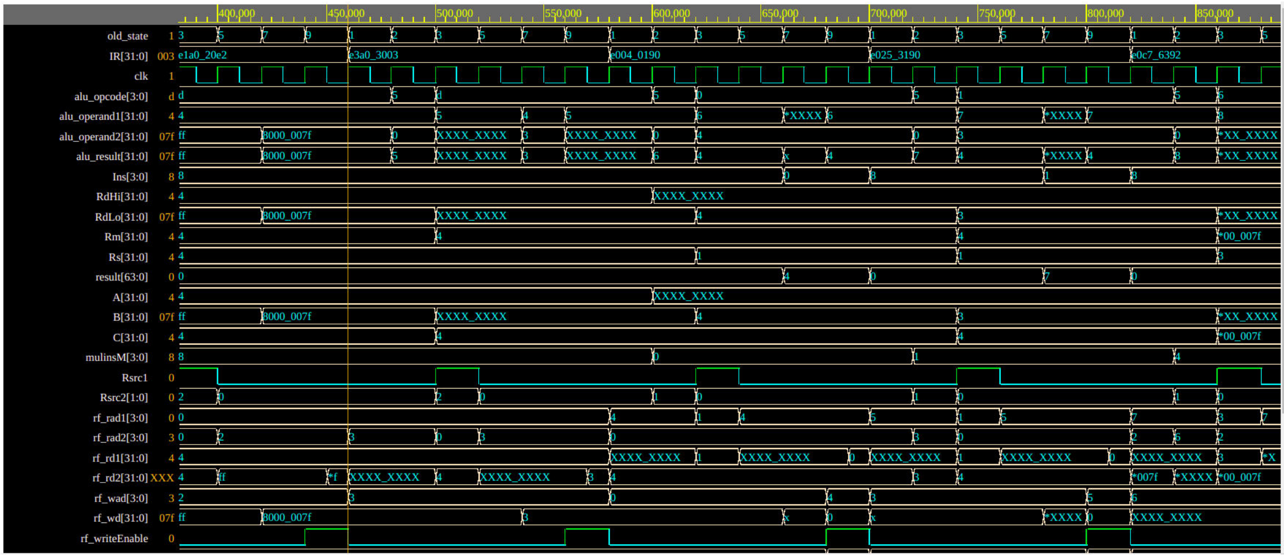
Instruction set for Data Memory:

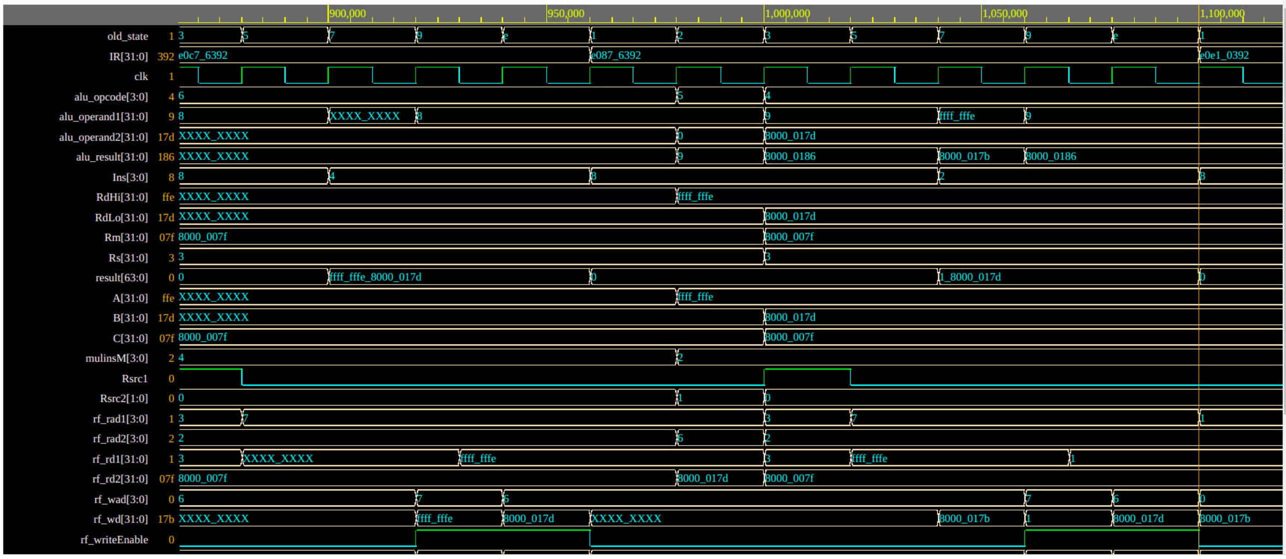
```
(   0 => X"E3A00004",
    1 => X"E3A01001",
    2 => X"E3A020FF",
    3 => X"E1A020E2",
    4 => X"E3A03003",
    5 => X"E0040190",
    6 => X"E0253190",
    7 => X"E0C76392",
    8 => X"E0876392",
    9 => X"E0E10392",
    10 => X"E3A00004",
    11 => X"E3A01001",
    12 => X"E0A10392",
    others => X"00000000" );
```

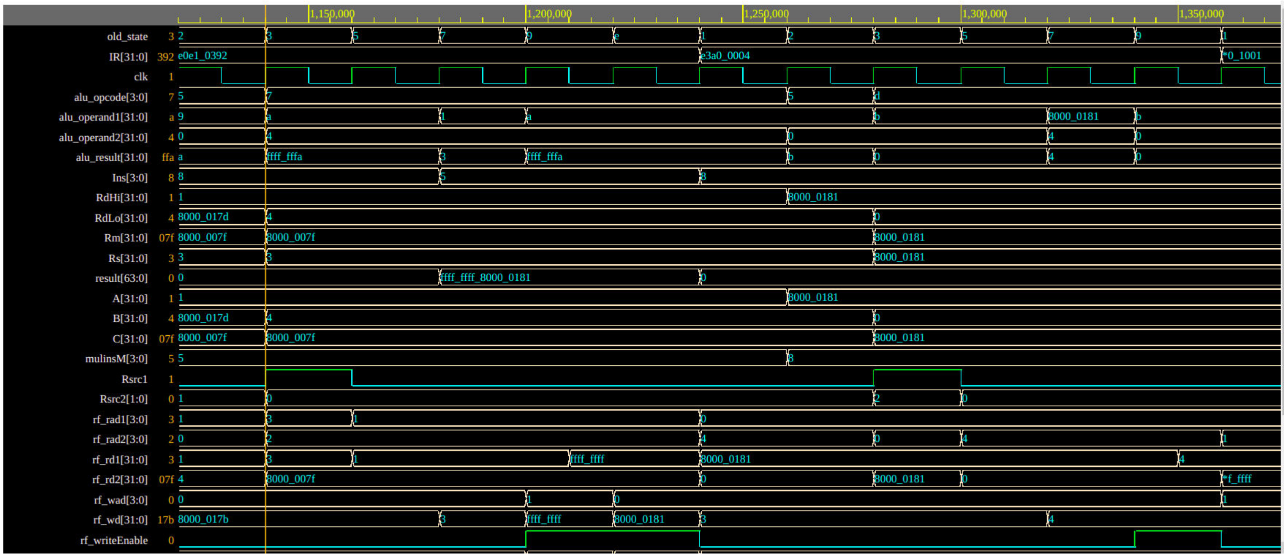
EPWave:

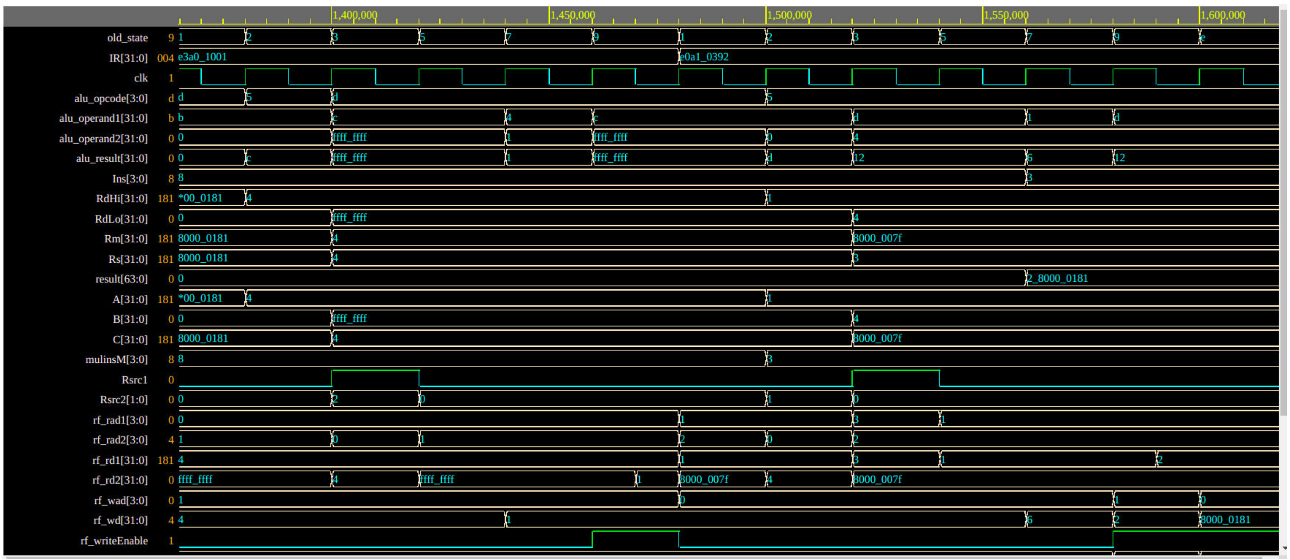
- The images are on the right side of the previous image
- Only important signals are shown.






900,000
950,000
1,000,000
1,050,000
1,100,000
old_state
IR[31:0]
clk
alu_opcode[3:0]
alu_operand1[31:0]
alu_operand2[31:0]
alu_result[31:0]
Ins[3:0]
RdHi[31:0]
RdLo[31:0]
Rm[31:0]
Rs[31:0]
result[63:0]
A[31:0]
B[31:0]
C[31:0]
mulinsM[3:0]
Rsrc1
Rsrc2[1:0]
rf_rad1[3:0]
rf_rad2[3:0]
rf_rd1[31:0]
rf_rd2[31:0]
rf_wad[3:0]
rf_wd[31:0]
rf_writeEnable


1,150,000
1,200,000
1,250,000
1,300,000
1,350,000
old_state
IR[31:0]
clk
alu_opcode[3:0]
alu_operand1[31:0]
alu_operand2[31:0]
alu_result[31:0]
Ins[3:0]
RdHi[31:0]
RdLo[31:0]
Rm[31:0]
Rs[31:0]
result[63:0]
A[31:0]
B[31:0]
C[31:0]
mulinsM[3:0]
Rsrc1
Rsrc2[1:0]
rf_rad1[3:0]
rf_rad2[3:0]
rf_rd1[31:0]
rf_rd2[31:0]
rf_wad[3:0]
rf_wd[31:0]
rf_writeEnable


1,400,000
1,450,000
1,500,000
1,550,000
1,600,000
old_state
IR[31:0]
clk
alu_opcode[3:0]
alu_operand1[31:0]
alu_operand2[31:0]
alu_result[31:0]
Ins[3:0]
RdHi[31:0]
RdLo[31:0]
Rm[31:0]
Rs[31:0]
result[63:0]
A[31:0]
B[31:0]
C[31:0]
mulinsM[3:0]
Rsrc1
Rsrc2[1:0]
rf_rad1[3:0]
rf_rad2[3:0]
rf_rd1[31:0]
rf_rd2[31:0]
rf_wad[3:0]
rf_wd[31:0]
rf_writeEnable