Introduction:

There are 9 files namely alu.vhd, control_fsm.vhd, data_memory.vhd, flags.vhd, register_file.vhd, stage5.vhd, shifter.vhd, test_shifter.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.
stage5.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.
test_shifter.vhd:	module to test the shifter.

- I have added 3 new states to fsm, now my fsm has 12 states.
- Out of those 3 states one state reads from the register file and store it in a register C, one provides control signals to shifter in case of DP instruction other in case of DT instruction.
- New control signals are
    - CW: write enable for register C
    - SW: enable for shifter to operate
    - Dsrc: for multiplexer of data_in of shifter
    - Tsrc: for multiplexer of type of shifter
    - AMsrc: for multiplexer of amount of shifter

Test done:

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

ARMSim code:	Alu operation done: (done when old_state is in 7 or 8)
mov r0, #256	mov 100
mov r1, #7	mov 7
mov r2, #15	mov 15
mov r3, #3	mov 3
str r1, [r2, r0, LSL #3]	f + 800
add r3, r2, r0, LSL r3	f + 800
add r2, r2, r0, LSL #3	f + 800
add r2, r2, r0, LSR #3	80f + 20
add r2, r2, r0, ROR #3	82f + 20
add r2, r2, r0, ASR #3	84f + 20
ldr r4, [r2]	86f + 0
add r5, r4, #4	0 + 4
str r5, [r2, #-10]	86f - a
ldr r6, [r2, #-10]	86f - a
add r7, r6, r4	4 + 0
add r8, r2, #1024	86f + 400
add r8, r7, r5, ROR #1	4 + 2
ldr r8, [r2, #31]	86f + 1f

(on right side numbers are in hex same as showed in EPWave)

Data Memory initialisation:

```
(       0 => X"E3A00C01",
        1 => X"E3A01007",
        2 => X"E3A0200F",
        3 => X"E3A03003",
        4 => X"E7821180",
        5 => X"E0823310",
        6 => X"E0822180",
        7 => X"E08221A0",
        8 => X"E08221E0",
        9 => X"E08221C0",
        10 => X"E5924000",
        11 => X"E2845004",
        12 => X"E502500A",
        13 => X"E512600A",
        14 => X"E0867004",
        15 => X"E2828B01",
        16 => X"E08780E5",
        17 => X"E592801F",
        others => X"00000000" );
```

EPWave:

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