Report of Lab 2.5

Lalit Meena,2019CS50439

For this Lab 2 we are building microarchitecture for subset of arm instructions.

This stage involves putting together stage 1 modules, to form a simple processor which can execute the following subset of ARM instructions with limited variants/features. {add, sub, cmp, mov, ldr, str, beq, bne, b} .Then in stage 3 ,we extend single cycle datapath to flexible and efficient clock period multicycle path.

In stage4 ,we have tested our design exhaustively and reported waves/test-cases for most DP instructions.

Here in stage 5 ,we are adding one more features to our implementation which supports shift operations on register and immediate operand variations.

Zip folder name – L2.5_2019CS50439

Submission folder, **L2.5_2019CS50439** contains-

1. **alu.vhd –** alu module design file
2. **data_mem.vhd -** data memory 128x32 module design file
3. **rfile.vhd -** design file for module register file 16x32
4. **others.vhd-**contains additional modules require for stage-2**(Instruction Decoder, Condition Checker)**
5. **flags.vhd-** design file for module flags
6. **multicycle.vhd-** SIMPLE multi CYCLE PROCESSOR
7. **shifter.**vhd- Shifter which support 4 shift types-LSL,LSR,ASR,ROR

package.vhd -package to declare some types-words,bytes,etc **run.do**

testbench files-

testbench_multicycle.vhd - testbench file for module simple multicycle processor

testbench_shifter.vhd - testbench file for module shifter

shifter TESTCASES EPWAVE PICS-stestcase-1,2,3,4,5,6….

Shifter TESTCASES files- stestcasen.s, n is 1,2,…..

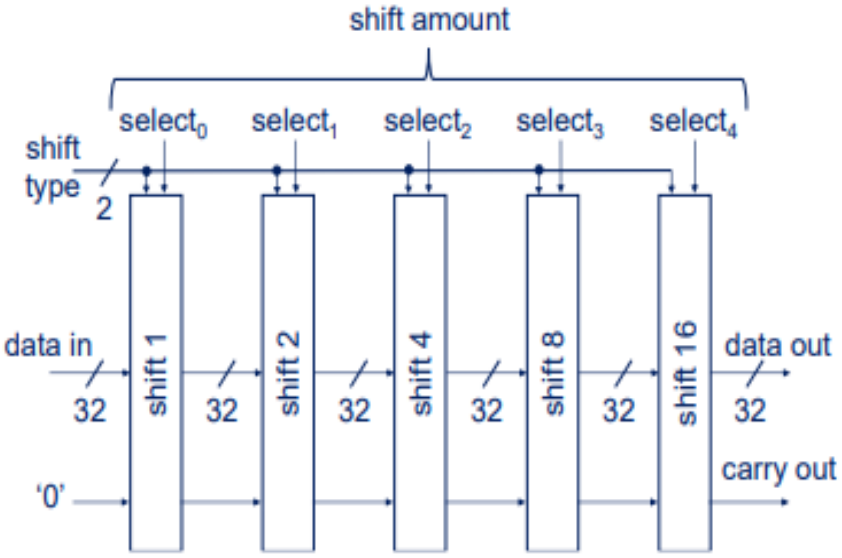
Report of Lab 2.5-lab 2 stage 5 description and test cases(also screenshots)

More details are in next pages of each module-

shifter.vhd –Shifter

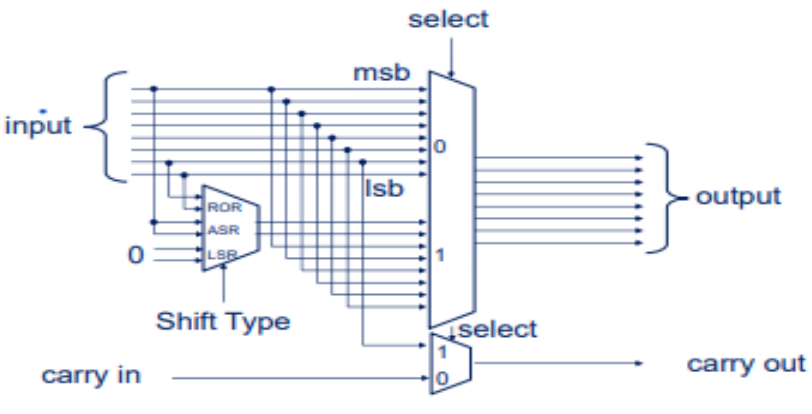
explanation-

We want to care of the range of shift amount from 0 to 31,for which we need 5 bit amount input.We also treat this shift amount value as a selector to various stages for shifting in parts.Therefore we need 5 stages as shown in the figure below. Five bits of shift amount are applied as select input at different stages. Stage i performs shift by 2 power i bits, i = 0 to 4.

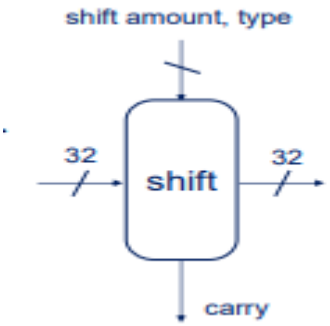


Our implementation also follows this 5 stage approach.Here in each stage ,we have Integrated all shift/rotate operations (LSL,LSR, ASR and ROR) together into one sub-unit ,for Example 32-bit data for stage i (range from 0 to 4)(i.e. shift amount = 2 power I, bits).

Our approach for each stage is similar to below but For simplicity we have used built-in VHDL functions(SLL,SRL,ROR,ASR) directly (as told) with multiplexing in each stage.



Note that if a stage performs a shift (select bit = '1'), the carry ,output generated during shift is selected and if it performs no shift (select bit = '0') then the carry ,output result from the previous stage is passed on.



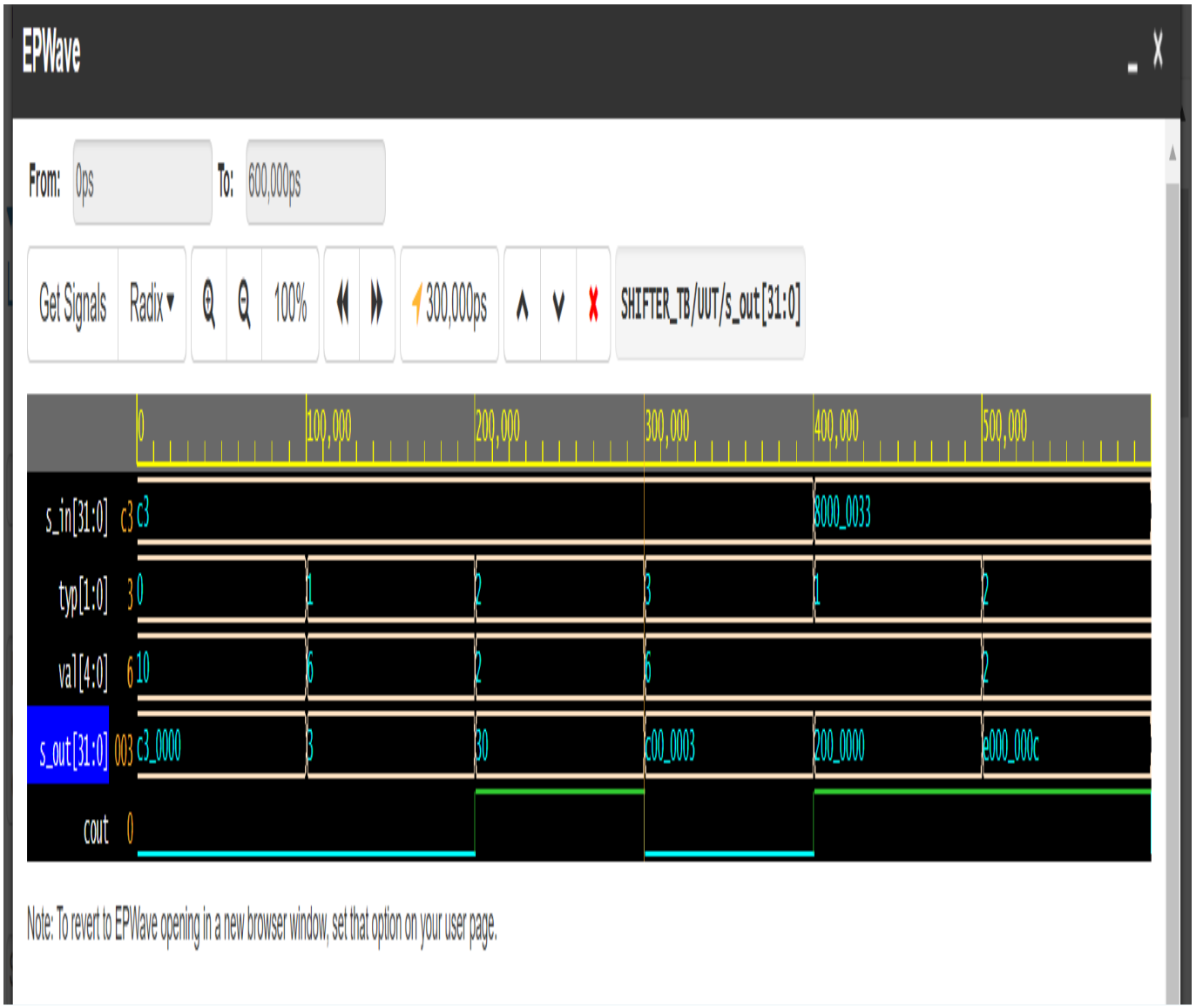
This is purely combinational circuit.

entity and ports info-

```
-- entity
entity SHIFTER is
port(
  s_in: in word ;
  --"00" for LSL, "01" for LSR, "10" for ASR and "11" for ROR
  typ:in std_logic_vector (1 downto 0);
  val: in std_logic_vector (4 downto 0);--deal overflow amount
?-outside...read
  s_out: out word;
  cout:out std_logic);
end entity;
```

simulated with help of testbench_shifter.vhd-

Here in these testcases,we have took 2 different inputs and applied different Shift operations.



Shift.

States and shifter implementation in main multicycle processor

Number of states used inside main clocked process is states-(fetch,read_AB,arithm,MSTR,MLDR,w2RF,READC,RSHIFT).Also used state signal in testbench to show at which state running currently.

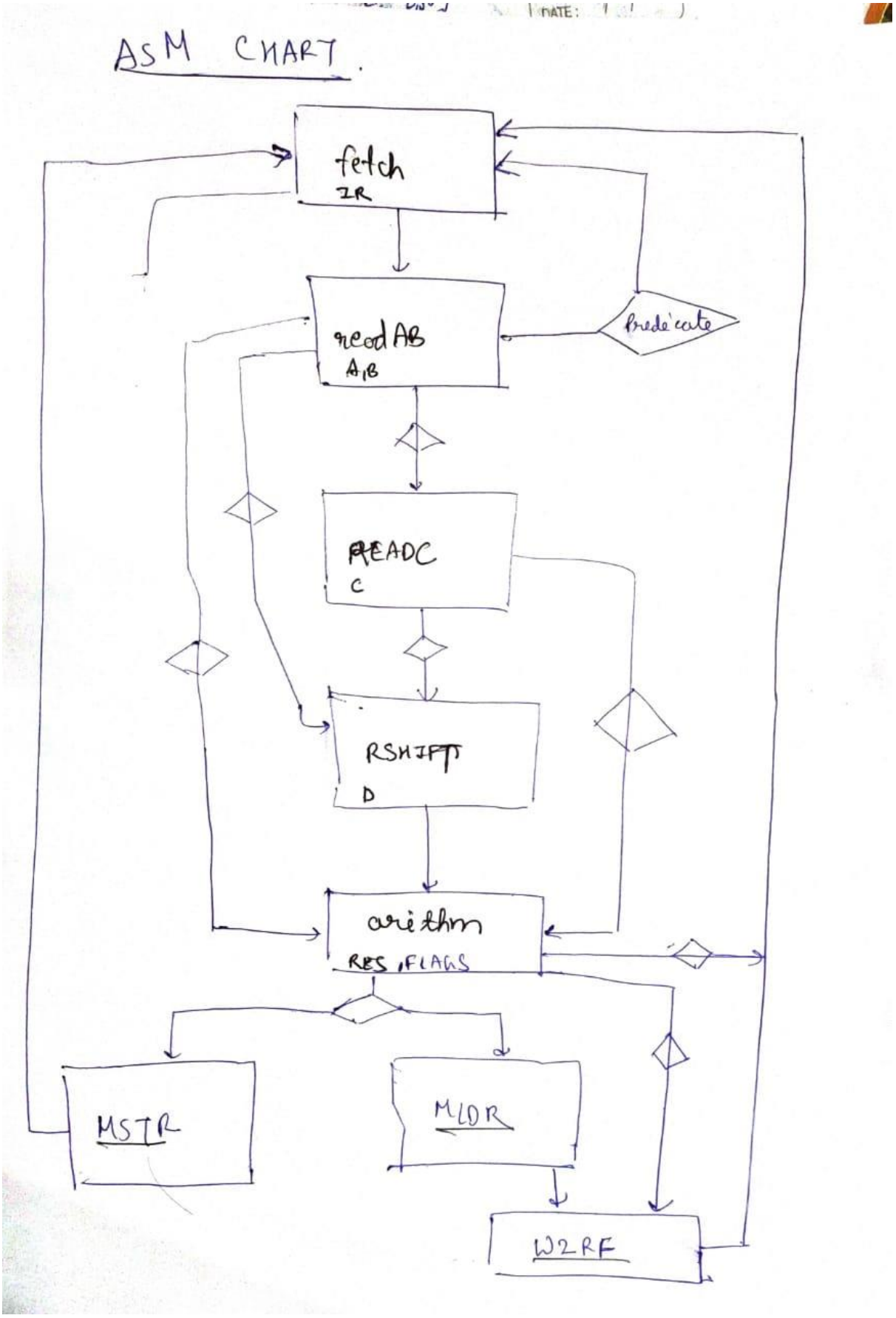
At this stage our design have only 8 states ,here we only added two new states –

READC- read 3rd register when need

RSHIFT-state need for shift/rotate(also in DP operand constant) operations

less possible state diagram,given next page-

ASM CHART.


fetch
IR
read AB
A,B
Predicate
READC
C
RSHIFT
D
arithm
RES, FLAGS
MSTR
MLDR
W2RF

Testcases

simulated with help of testbench_multicycle.vhd-

state change like flags and registers at end of program or step can be checked by signal like A,B,C,D,FLAGS, IR,PC,RES,state etc.

Correctness can be justified by values of registers read after instructions,etcAlso added extra mov instructions to read and show value of B,D OR 2nd operand.

We can observe number of cycles etc,also used state signal to show which state currently .

Below are special shifter related testcases-which covers all variants of shift/rotate instruction possible in DP AND DT.

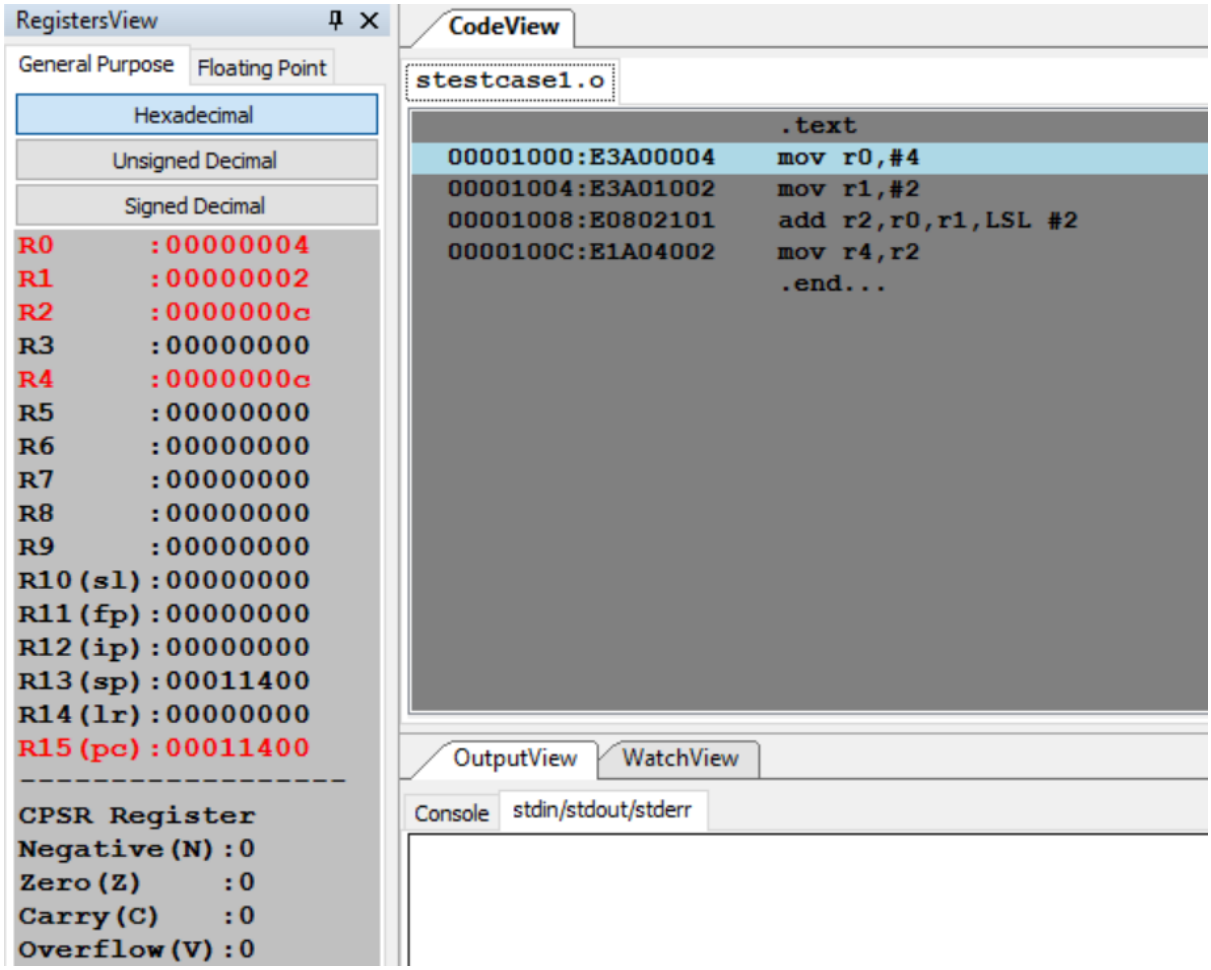
We can identify shift operation related parameters in middle-styp(shifter type),sftr(shifter need),amt(amount),c_sft,sout,etc.

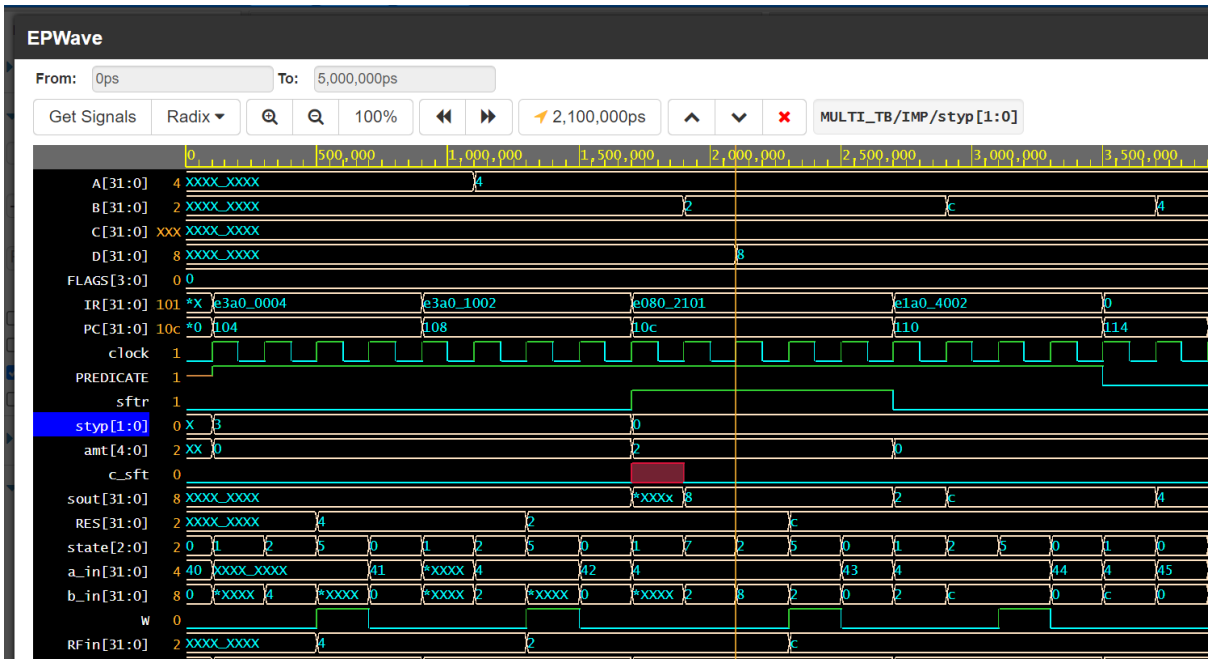
Take help of **run time limit** on simulator for taking required PC increments.

TESTCASES SEPARATE EPWAVE PICS and assembly files CAN also BE FOUND IN SUBMITTED FOLDER.

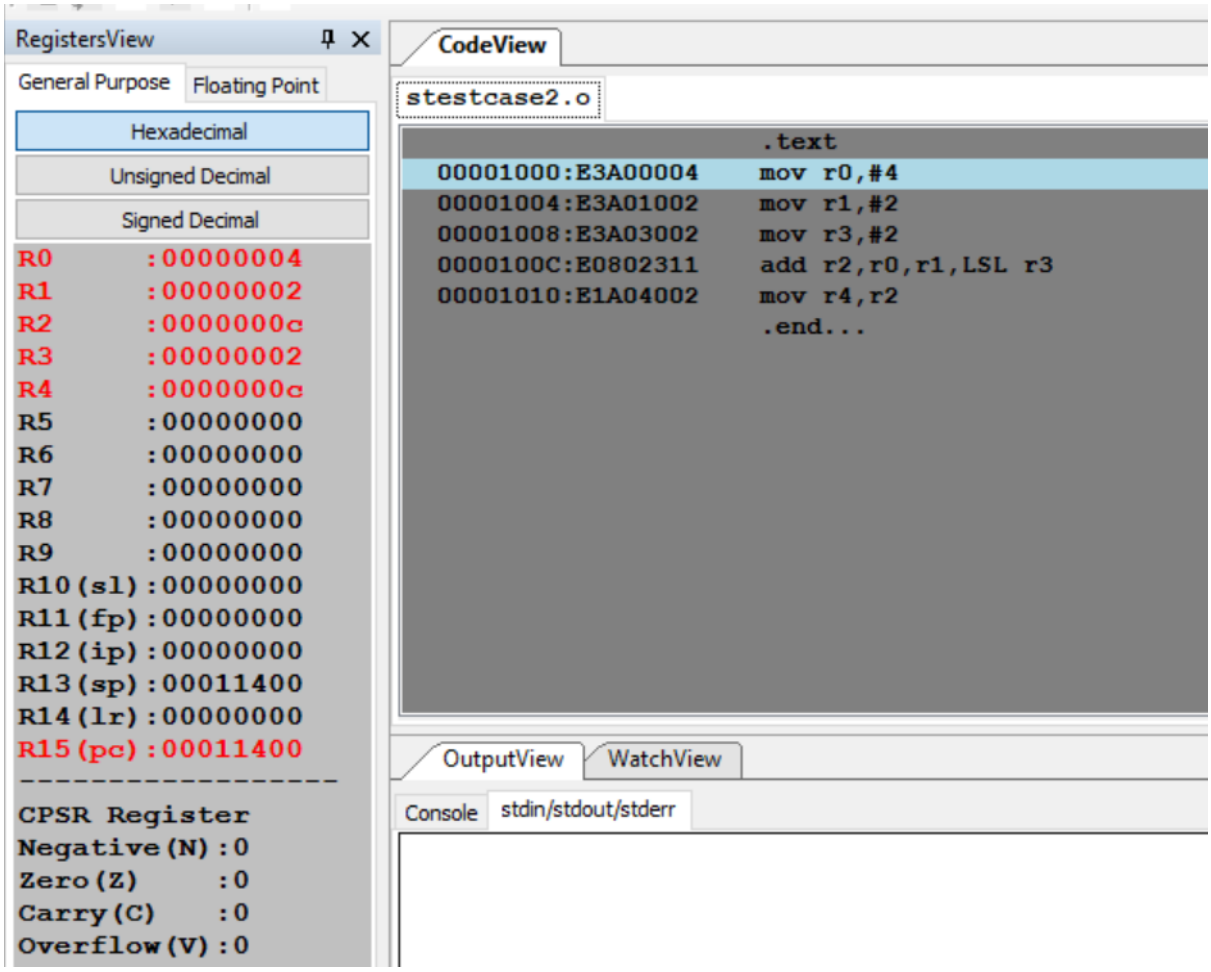
Check with previous testcase-1,found that previous implementation is working same for previous testcases.

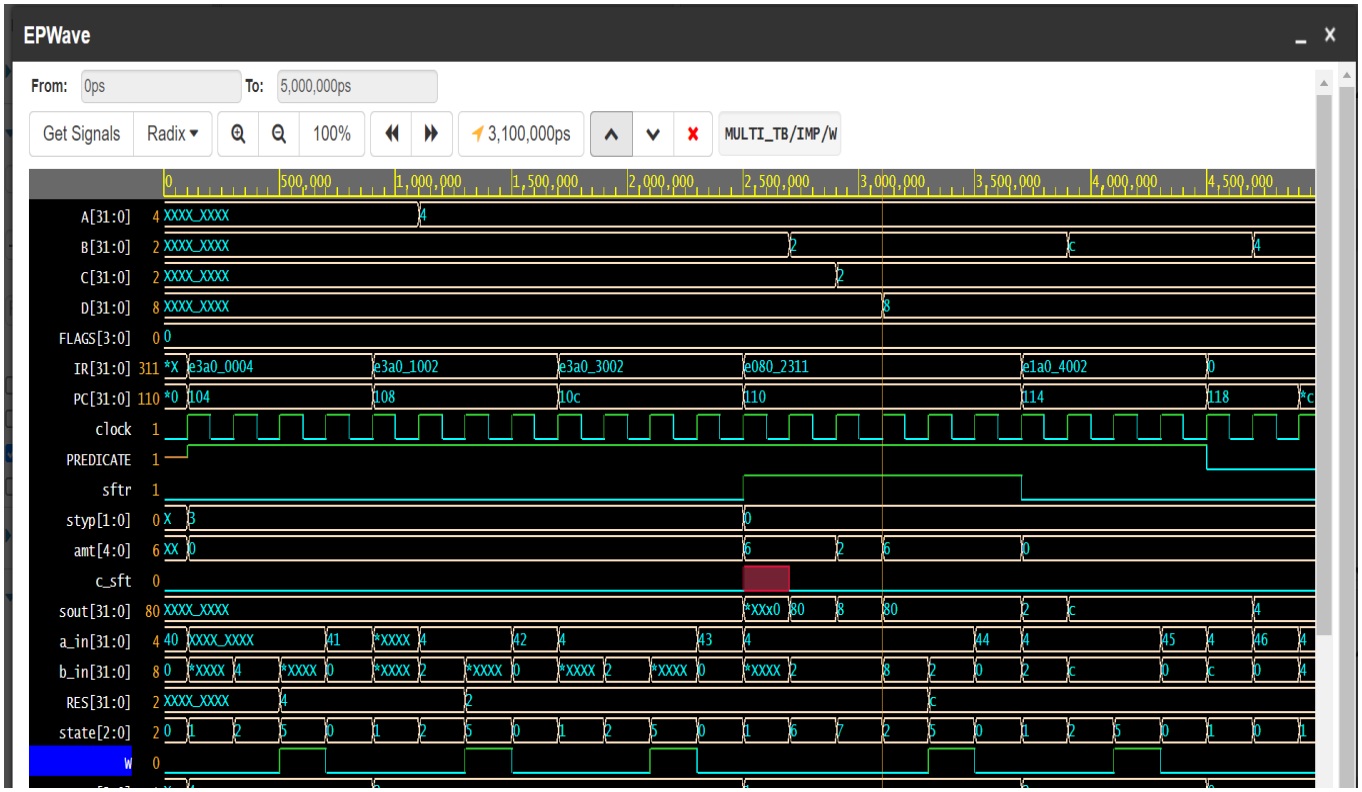
Stestcase1-


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00000004
R1 :00000002
R2 :0000000c
R3 :00000000
R4 :0000000c
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):00011400
CPSR Register
Negative(N):0
Zero(Z) :0
Carry(C) :0
Overflow(V):0
CodeView
stestcase1.o
.text
00001000:E3A00004 mov r0,#4
00001004:E3A01002 mov r1,#2
00001008:E0802101 add r2,r0,r1,LSL #2
0000100C:E1A04002 mov r4,r2
.end...
OutputView
WatchView
Console
stdin/stdout/stderr

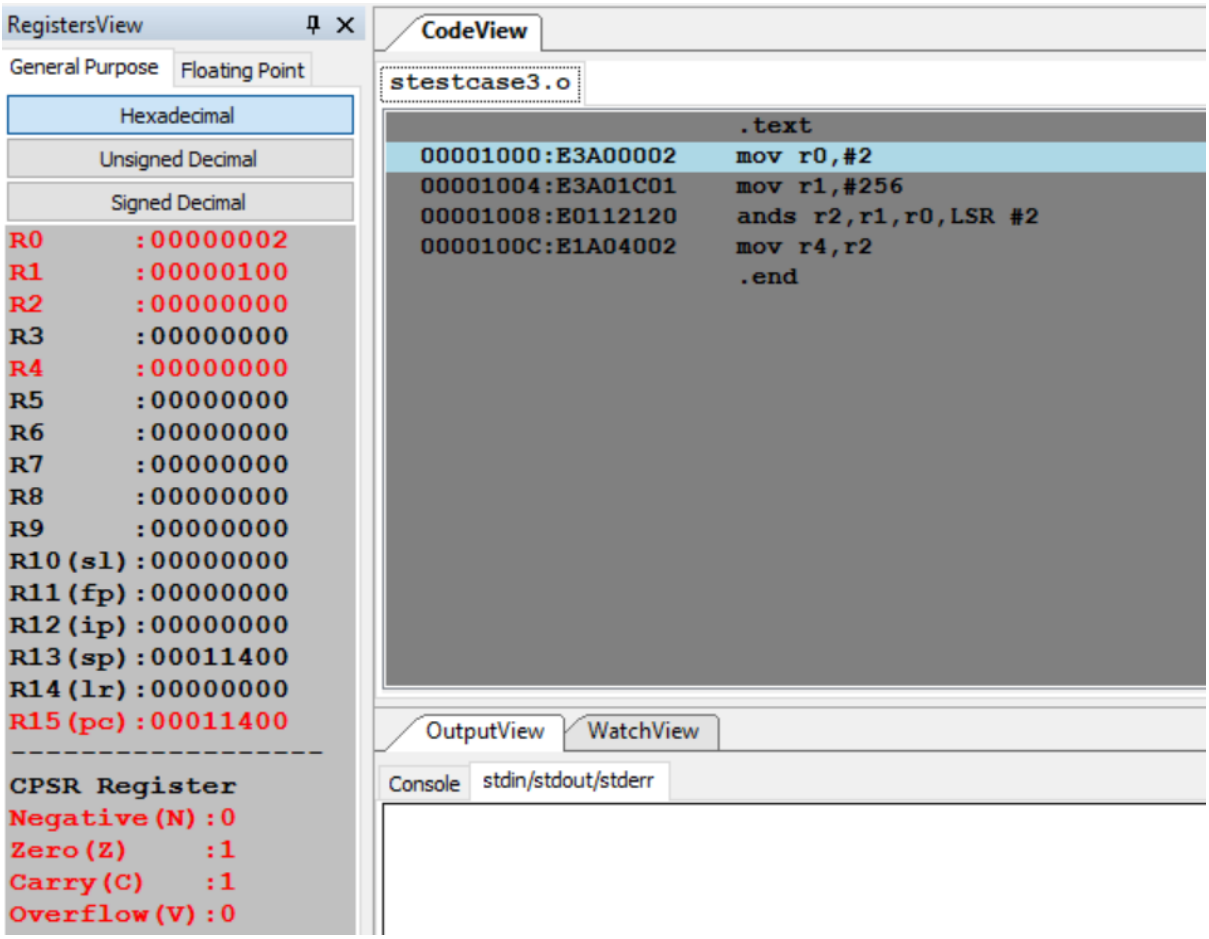

EPWave
From: 0ps
To: 5,000,000ps
Get Signals
Radix
100%
2,100,000ps
MULTI_TB/IMP/styp[1:0]
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
clock
PREDICATE
sftr
styp[1:0]
amt[4:0]
c_sft
sout[31:0]
RES[31:0]
state[2:0]
a_in[31:0]
b_in[31:0]
W
RFin[31:0]

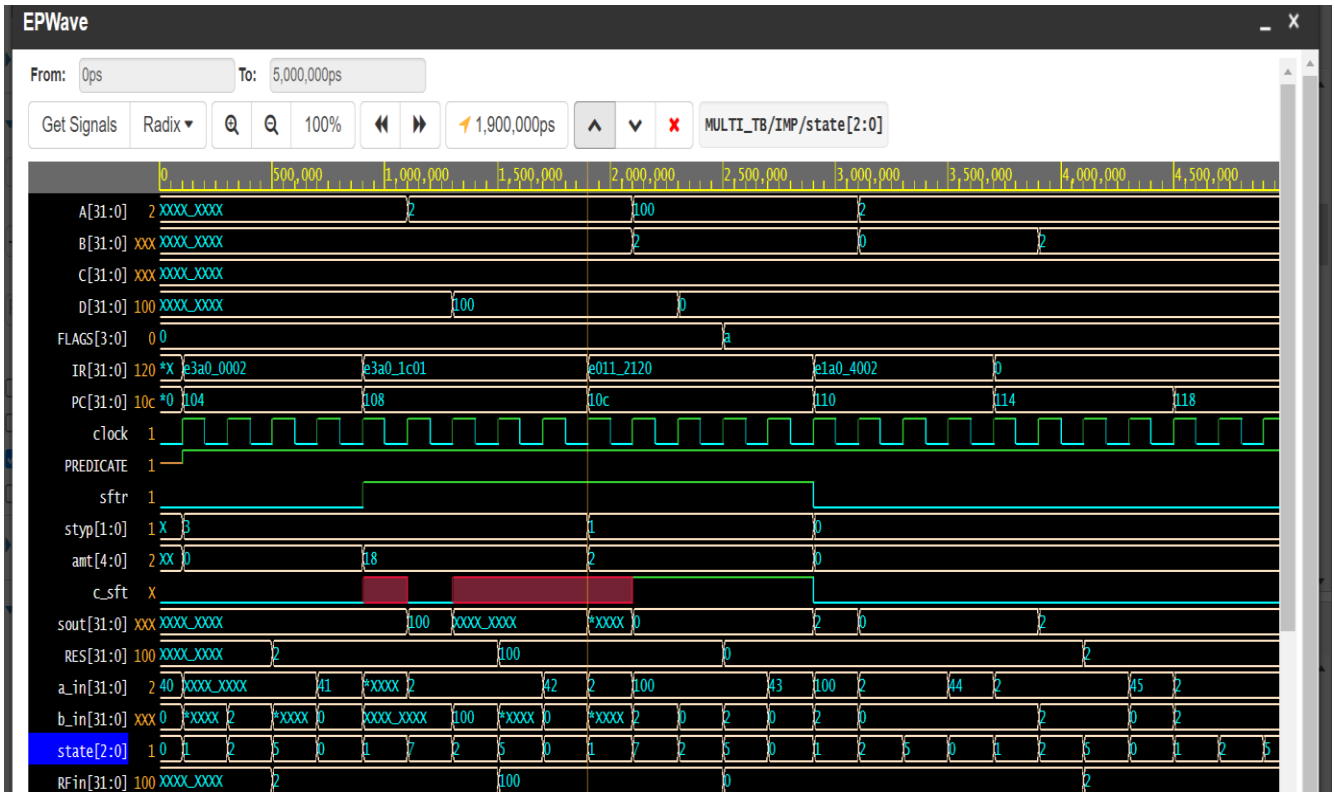
Stestcase2-


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00000004
R1 :00000002
R2 :0000000c
R3 :00000002
R4 :0000000c
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):00011400
CPSR Register
Negative(N):0
Zero(Z) :0
Carry(C) :0
Overflow(V):0
CodeView
stestcase2.o
.text
00001000:E3A00004 mov r0,#4
00001004:E3A01002 mov r1,#2
00001008:E3A03002 mov r3,#2
0000100C:E0802311 add r2,r0,r1,LSL r3
00001010:E1A04002 mov r4,r2
.end...
OutputView
WatchView
Console
stdin/stdout/stderr

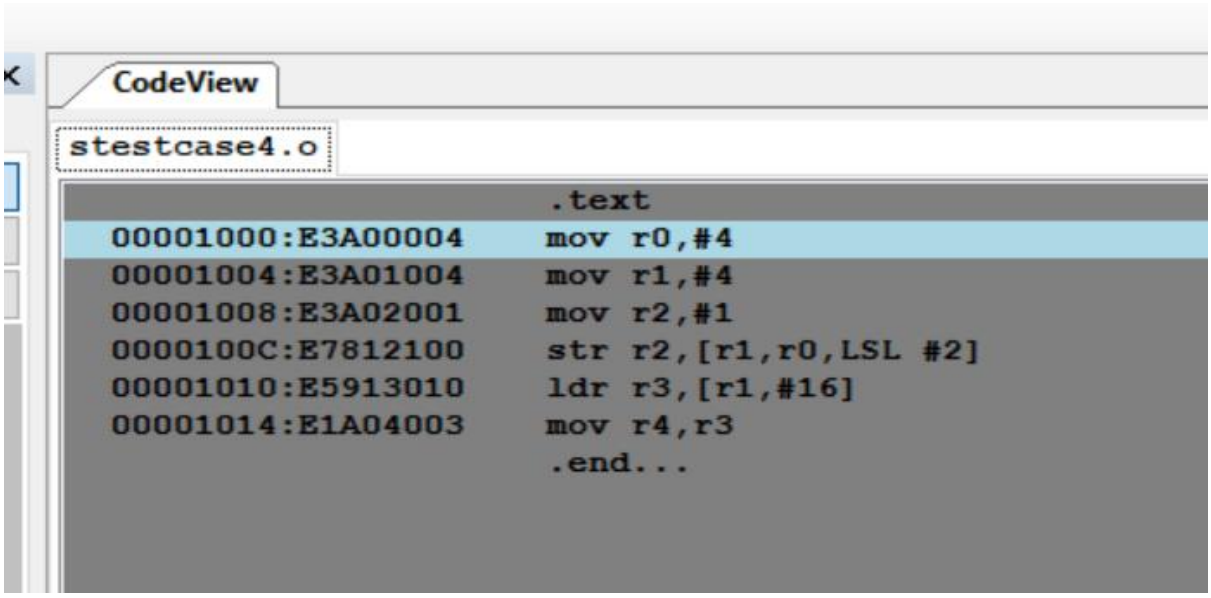

EPWave
From: 0ps
To: 5,000,000ps
Get Signals
Radix
100%
3,100,000ps
MULTI_TB/IMP/W
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
clock
PREDICATE
sftr
styp[1:0]
amt[4:0]
c_sft
sout[31:0]
a_in[31:0]
b_in[31:0]
RES[31:0]
state[2:0]
W

Stestcase3-


RegistersView
General Purpose
Floating Point
Hexadecimal
Unsigned Decimal
Signed Decimal
R0 :00000002
R1 :00000100
R2 :00000000
R3 :00000000
R4 :00000000
R5 :00000000
R6 :00000000
R7 :00000000
R8 :00000000
R9 :00000000
R10(sl) :00000000
R11(fp) :00000000
R12(ip) :00000000
R13(sp) :00011400
R14(lr) :00000000
R15(pc) :00011400
CPSR Register
Negative(N) :0
Zero(Z) :1
Carry(C) :1
Overflow(V) :0
CodeView
stestcase3.o
.text
00001000:E3A00002 mov r0,#2
00001004:E3A01C01 mov r1,#256
00001008:E0112120 ands r2,r1,r0,LSR #2
0000100C:E1A04002 mov r4,r2
.end
OutputView
WatchView
Console
stdin/stdout/stderr


EPWave
From: 0ps
To: 5,000,000ps
Get Signals
Radix
100%
1,900,000ps
MULTI_TB/IMP/state[2:0]
A[31:0]
B[31:0]
C[31:0]
D[31:0]
FLAGS[3:0]
IR[31:0]
PC[31:0]
clock
PREDICATE
sftr
styp[1:0]
amt[4:0]
c_sft
sout[31:0]
RES[31:0]
a_in[31:0]
b_in[31:0]
state[2:0]
RFin[31:0]

Stestcase4.s



Thank you.