Report of Lab 2.4

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.

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

In previous modules ,made almost no changes in them but only made combinational process sensitivity list more riskfree,like added Rm ..

Here in comparison previous stage ,there is only changes happened in testcase related files,like data_mem.vhd.

Zip folder name – L2.4_2019CS50439

Submission folder, **L2.4_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

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

testbench files-

testbench_multicycle.vhd - testbench file for module simple multicycle processor

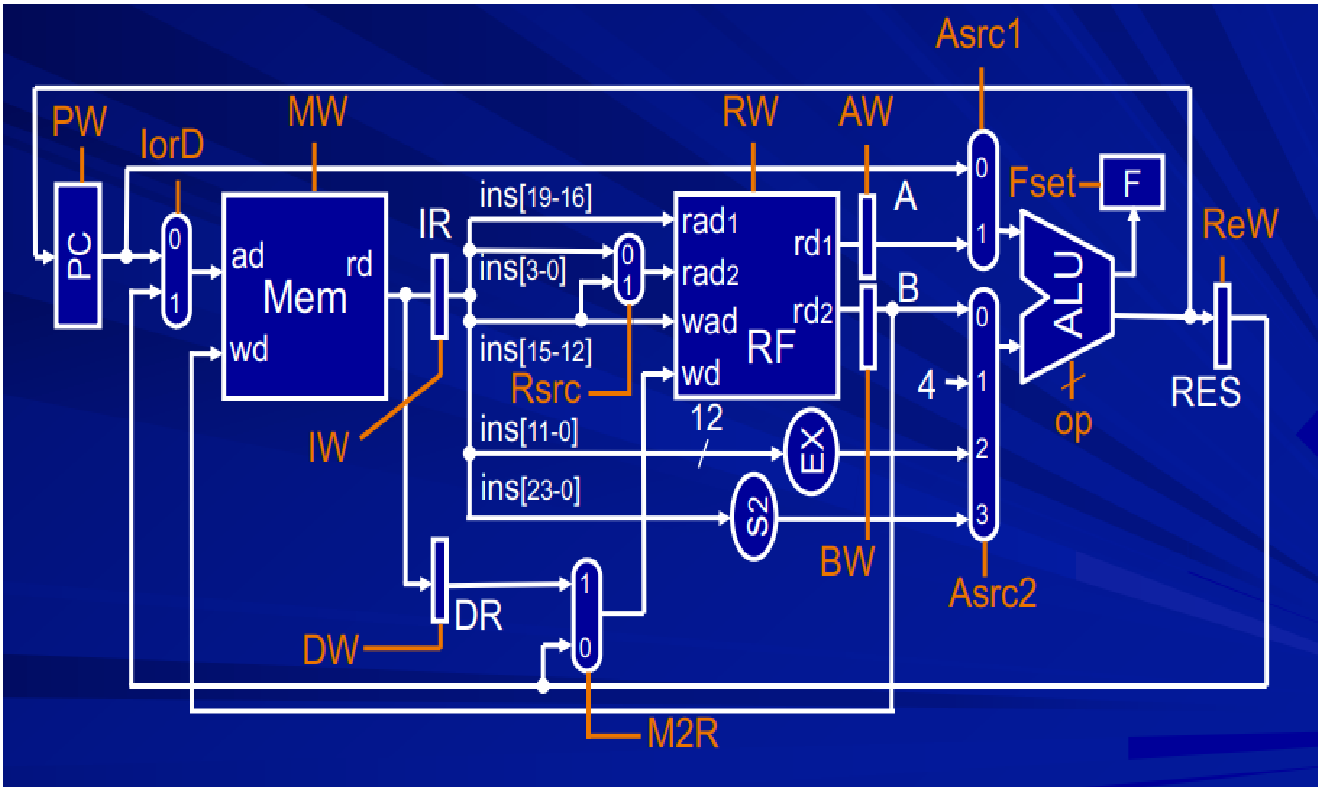
TESTCASES EPWAVE PICS-testcase-1,2,3,4,5,6….

TESTCASES files- testcasen.s, n is 1,2,…..

Report of Lab 2.4-lab 2 stage 4 description and test cases(also screenshots)

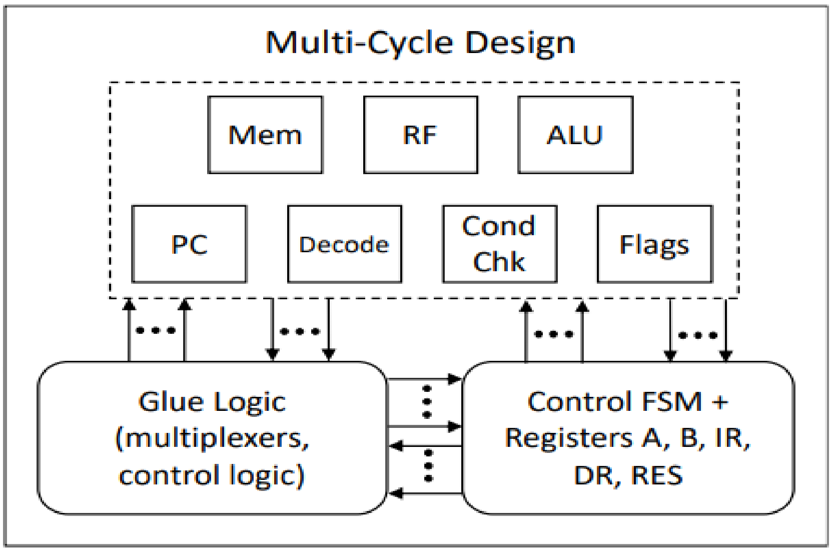
More details are in next pages of each module-

MULTICYCLE DATAPATH WITH CONTROL SIGNALS

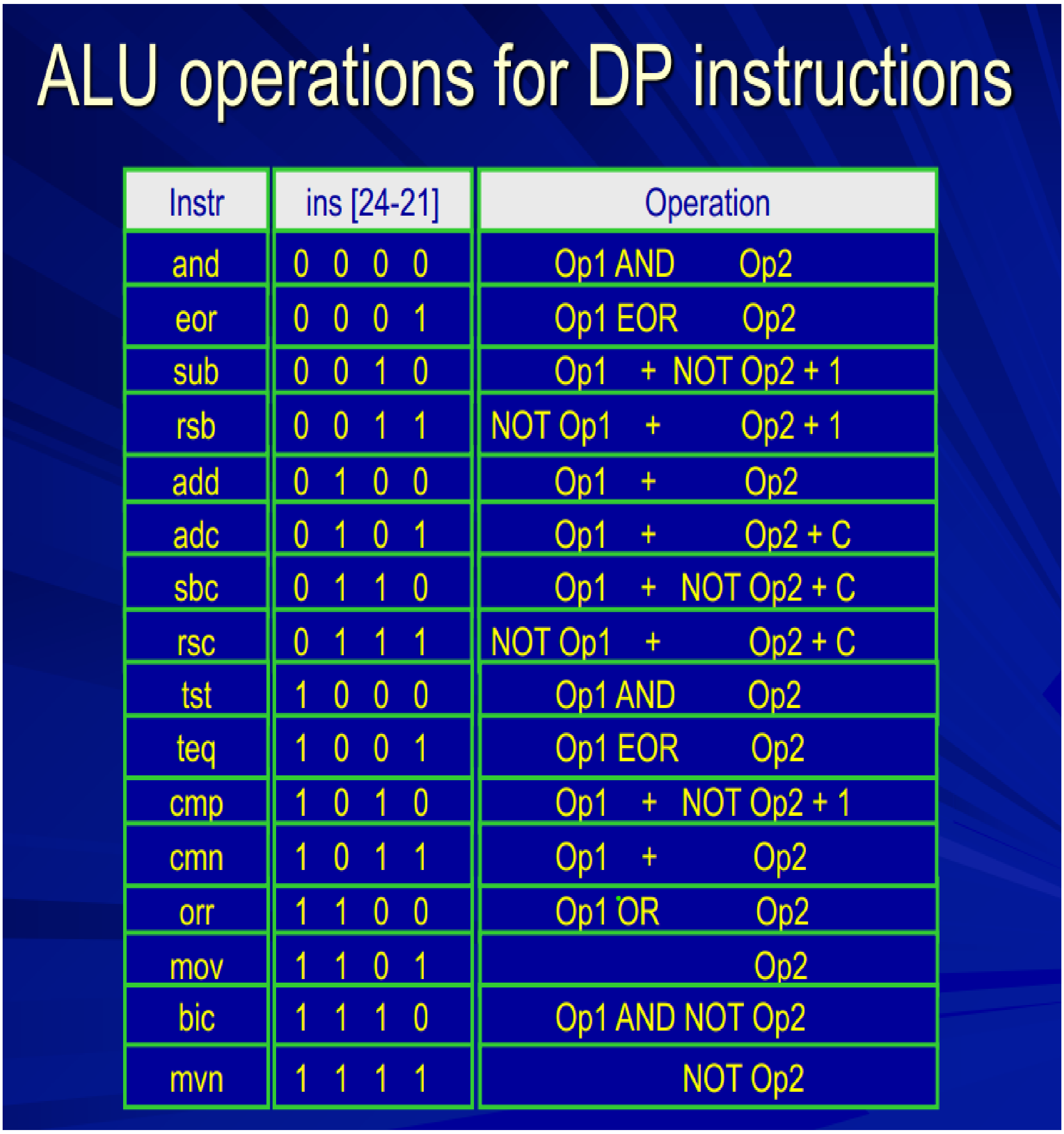


For each state in combinational process we generate glue logic and multiplex signal,write enables etc.

USED Registers A, B, IR, DR and RES as local signals,and modify value directly according to states so not need of write enable on them.


Multi-Cycle Design
Mem
RF
ALU
PC
Decode
Cond Chk
Flags
Glue Logic (multiplexers, control logic)
Control FSM + Registers A, B, IR, DR, RES

DP opcodes-



ALU operations for DP instructions

Instr	ins [24-21]	Operation
and	0 0 0 0	Op1 AND Op2
eor	0 0 0 1	Op1 EOR Op2
sub	0 0 1 0	Op1 + NOT Op2 + 1
rsb	0 0 1 1	NOT Op1 + Op2 + 1
add	0 1 0 0	Op1 + Op2
adc	0 1 0 1	Op1 + Op2 + C
sbc	0 1 1 0	Op1 + NOT Op2 + C
rsc	0 1 1 1	NOT Op1 + Op2 + C
tst	1 0 0 0	Op1 AND Op2
teq	1 0 0 1	Op1 EOR Op2
cmp	1 0 1 0	Op1 + NOT Op2 + 1
cmn	1 0 1 1	Op1 + Op2
orr	1 1 0 0	Op1 OR Op2
mov	1 1 0 1	Op2
bic	1 1 1 0	Op1 AND NOT Op2
mvn	1 1 1 1	NOT Op2

Testing and 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,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,2nd operand.

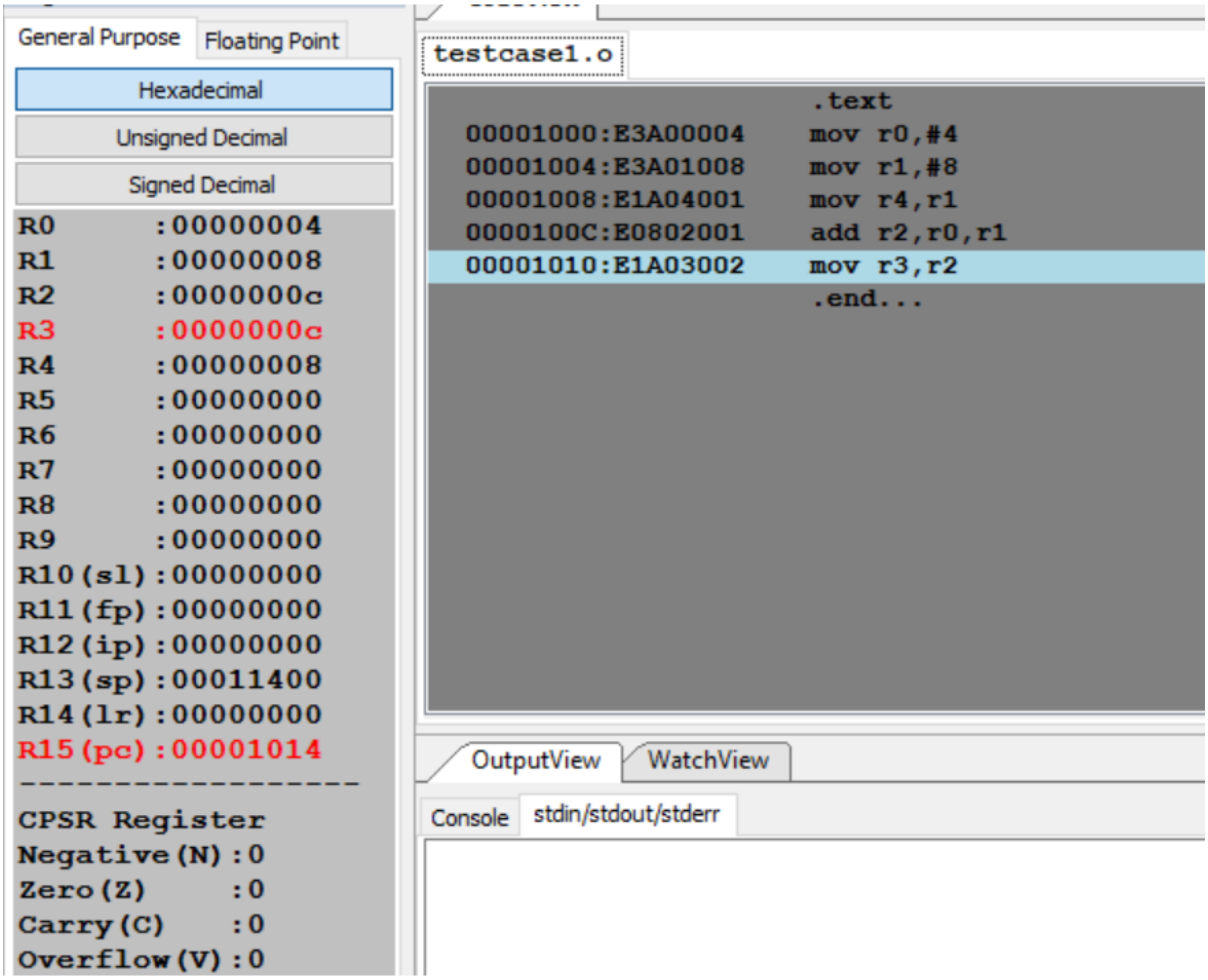
We can observe number of cycles etc,also used state signal in

testbench to show at which state running currently.

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

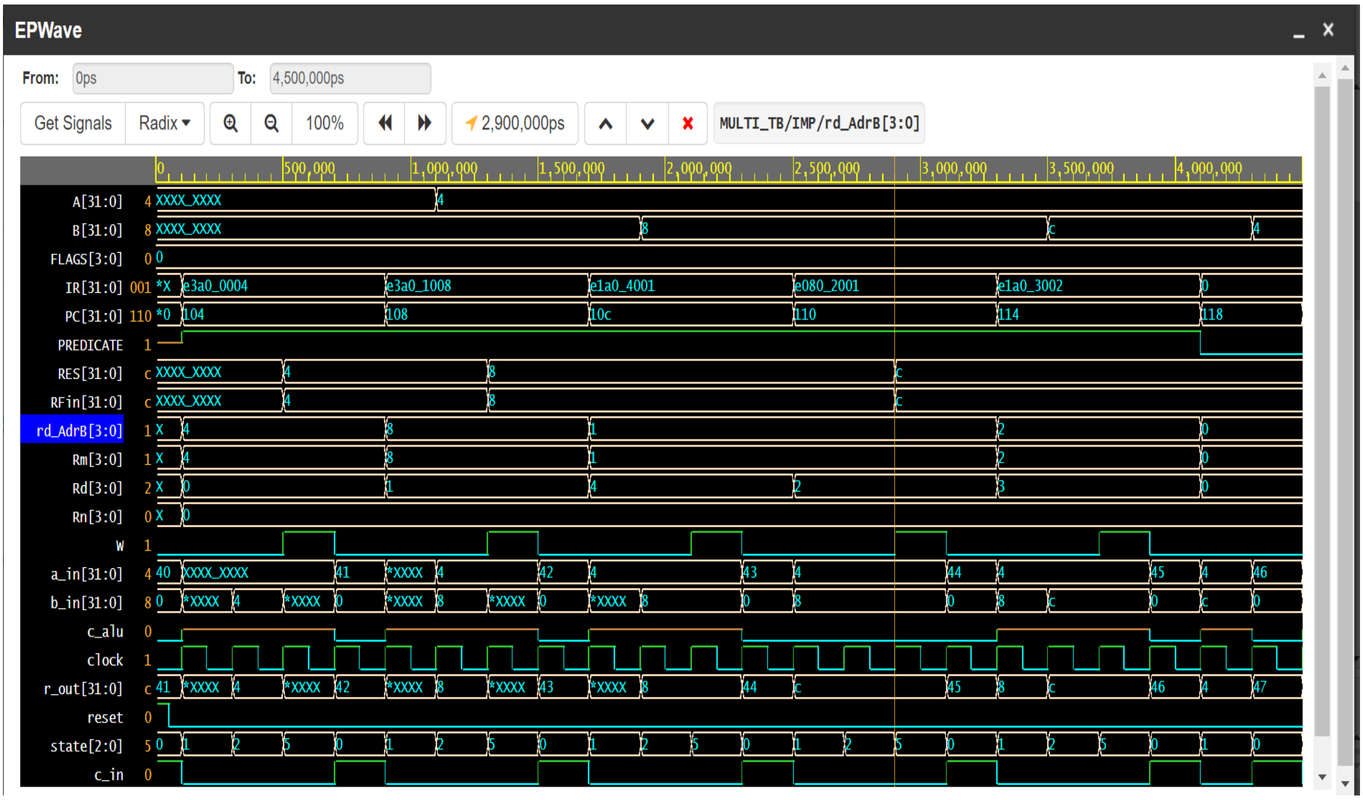
TESTCASES SEPARATE EPWAVE PICS and assembly files CAN also BE FOUND IN SUBMITTED FOLDER.

Example-1

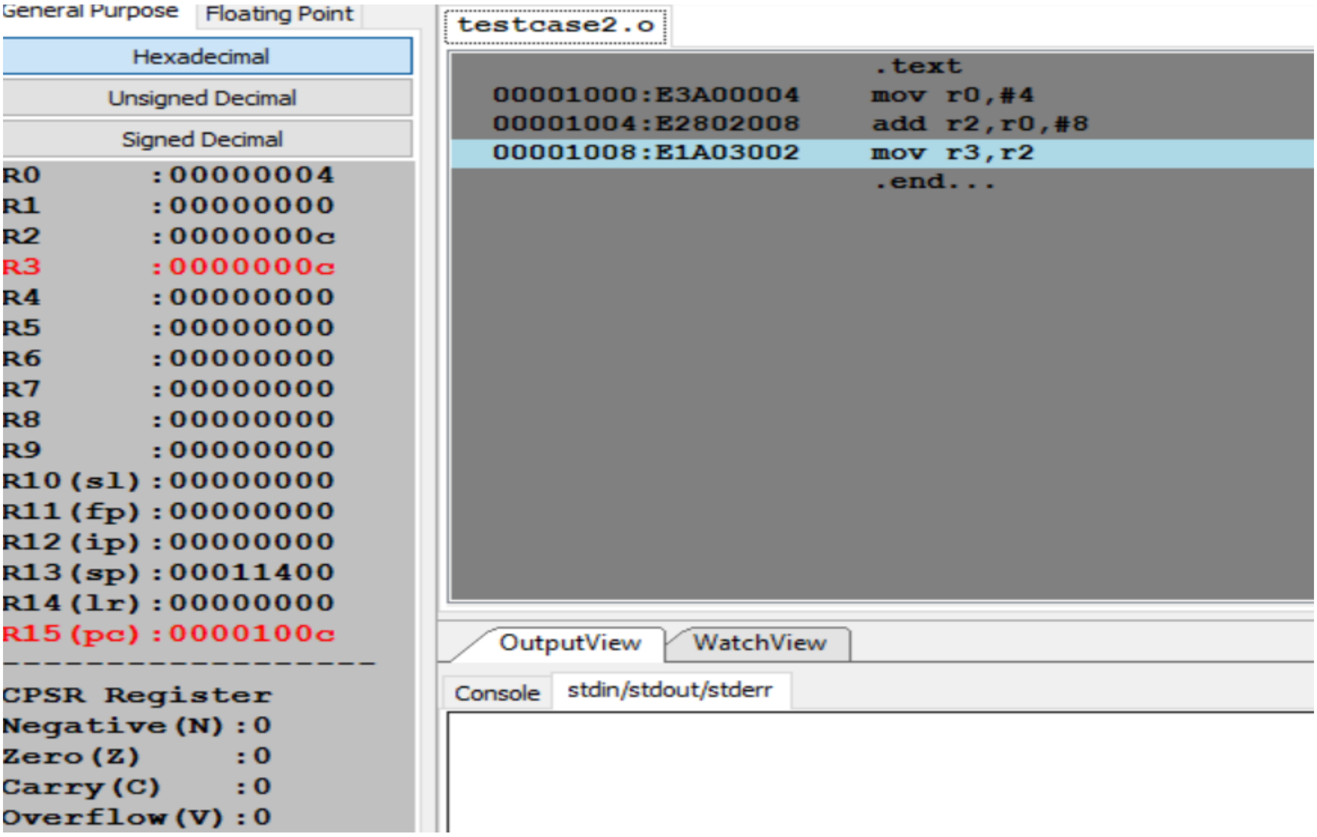


```
R0        :00000004
R1        :00000008
R2        :0000000c
R3        :0000000c
R4        :00000008
R5        :00000000
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl)   :00000000
R11(fp)   :00000000
R12(ip)   :00000000
R13(sp)   :00011400
R14(lr)   :00000000
R15(pc)   :00001014
-------------------
CPSR Register
Negative(N) :0
Zero(Z)     :0
Carry(C)    :0
Overflow(V) :0
```

```
                        .text
00001000:E3A00004       mov r0,#4
00001004:E3A01008       mov r1,#8
00001008:E1A04001       mov r4,r1
0000100C:E0802001       add r2,r0,r1
00001010:E1A03002       mov r3,r2
                        .end...
```

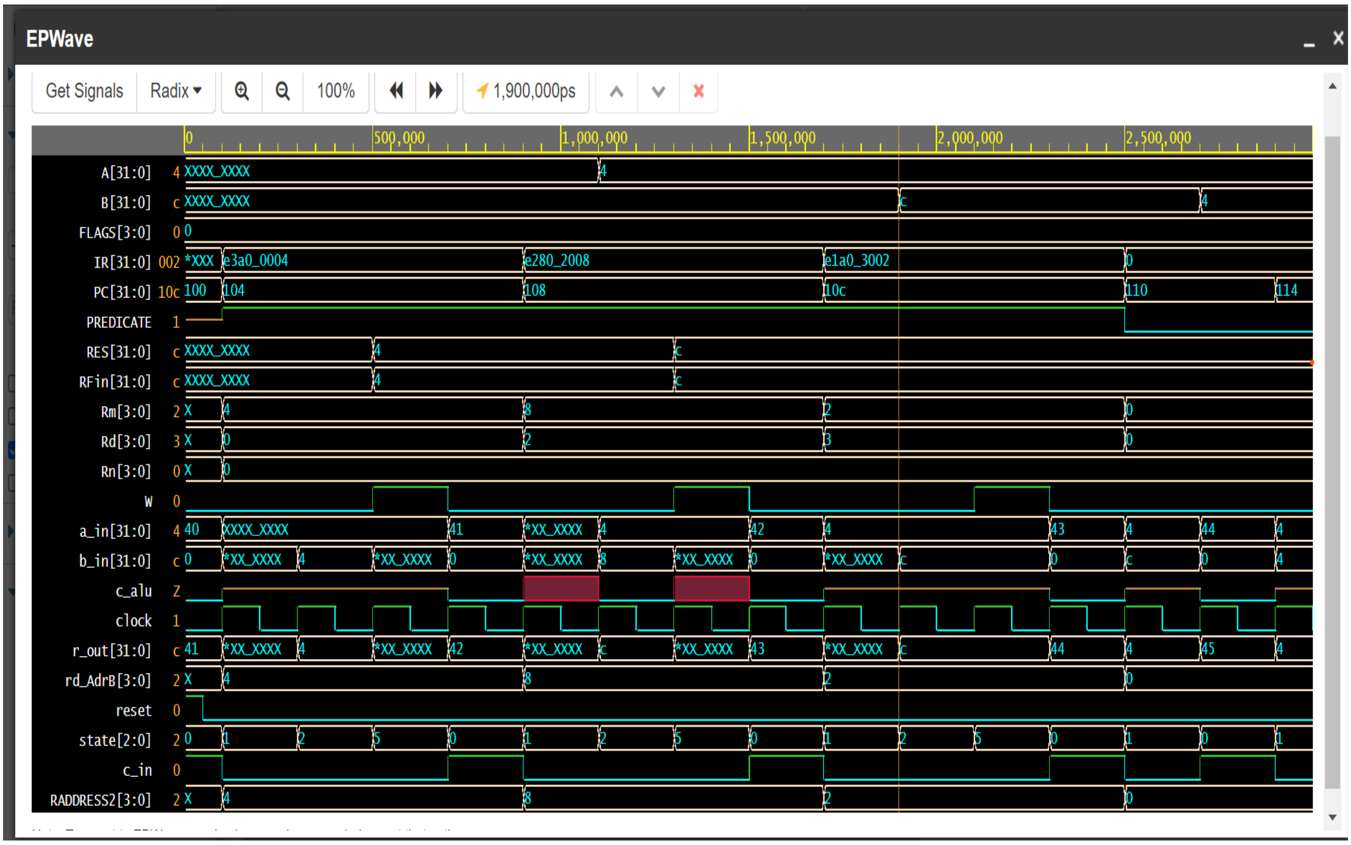


Example-2

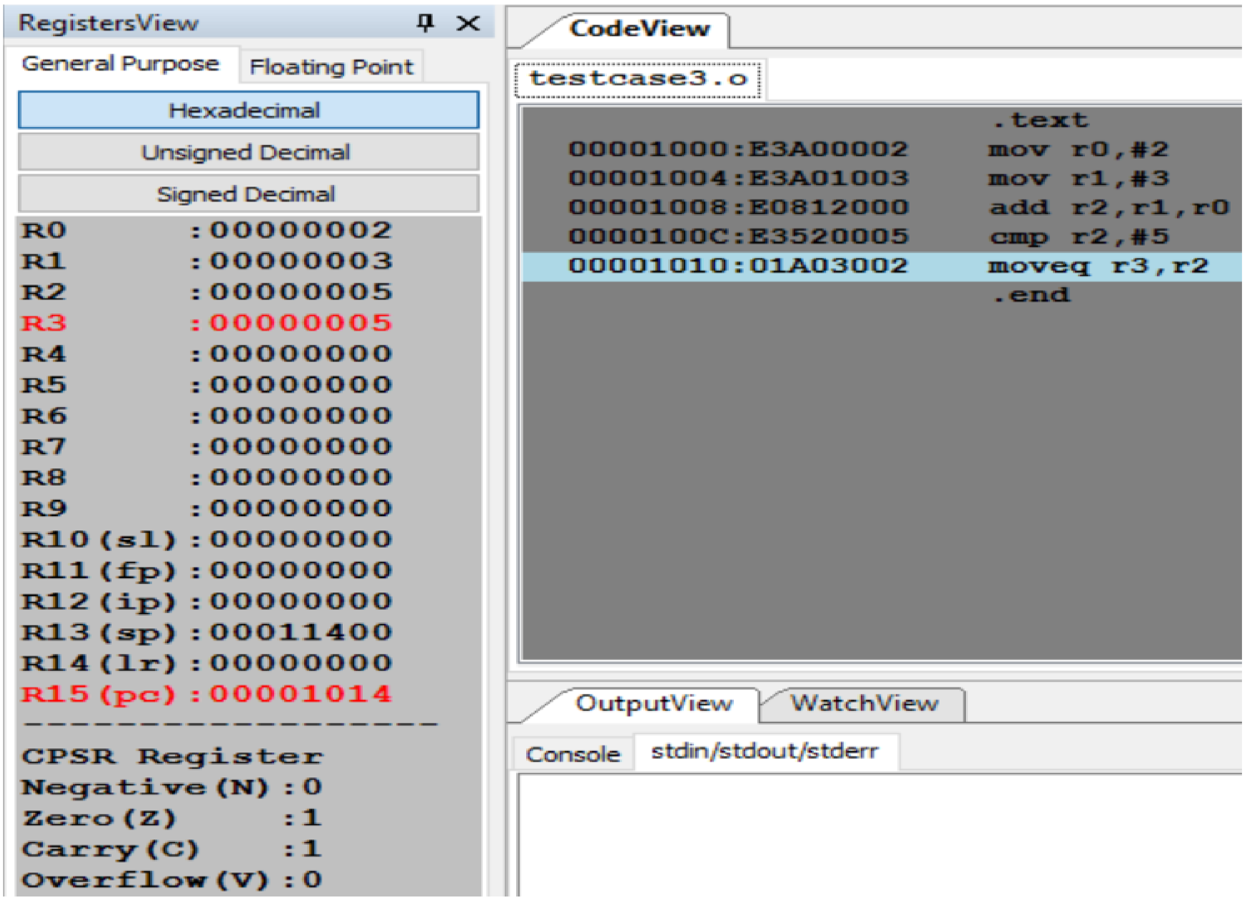


```
R0        :00000004
R1        :00000000
R2        :0000000c
R3        :0000000c
R4        :00000000
R5        :00000000
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl):00000000
R11(fp):00000000
R12(ip):00000000
R13(sp):00011400
R14(lr):00000000
R15(pc):0000100c
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :0
Overflow(V):0
```

```
                         .text
00001000:E3A00004       mov r0,#4
00001004:E2802008       add r2,r0,#8
00001008:E1A03002       mov r3,r2
                         .end...
```

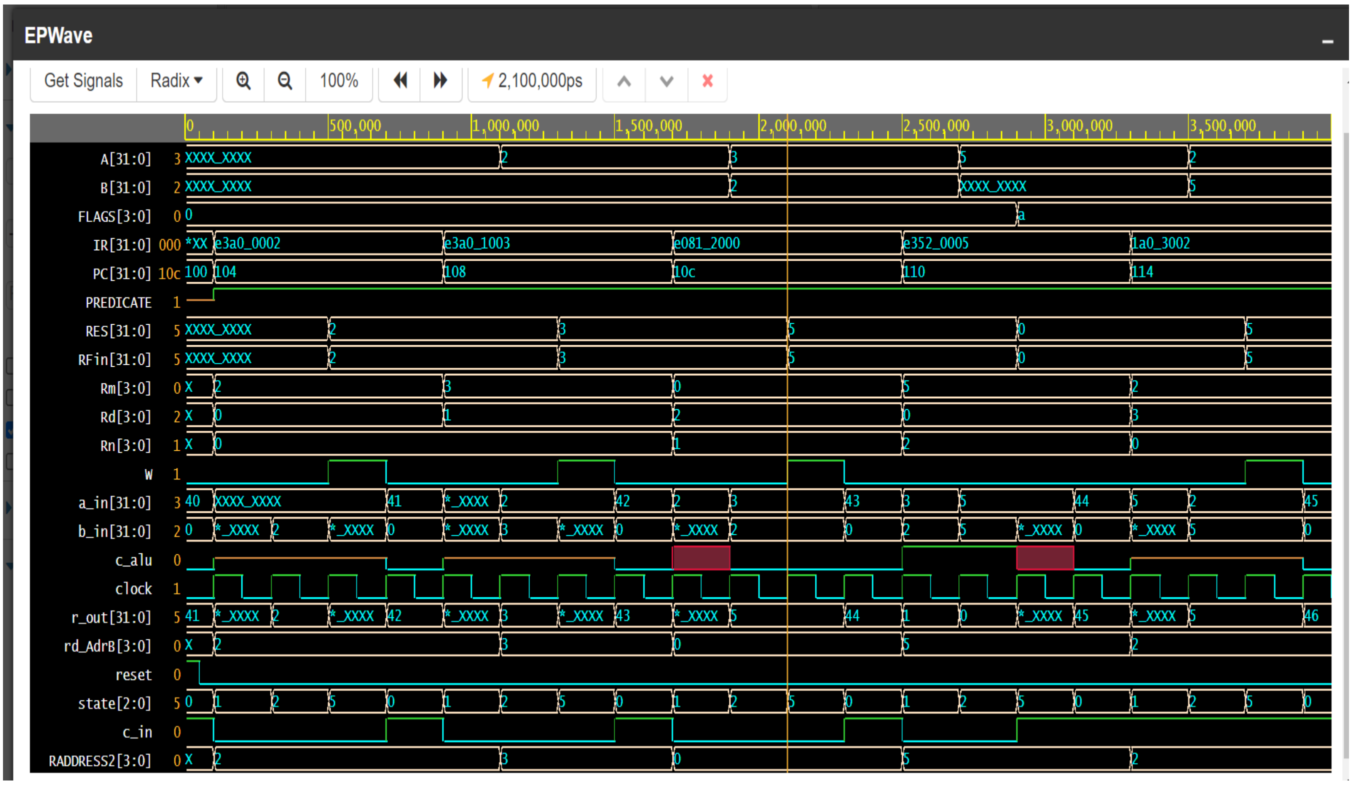


Example-3



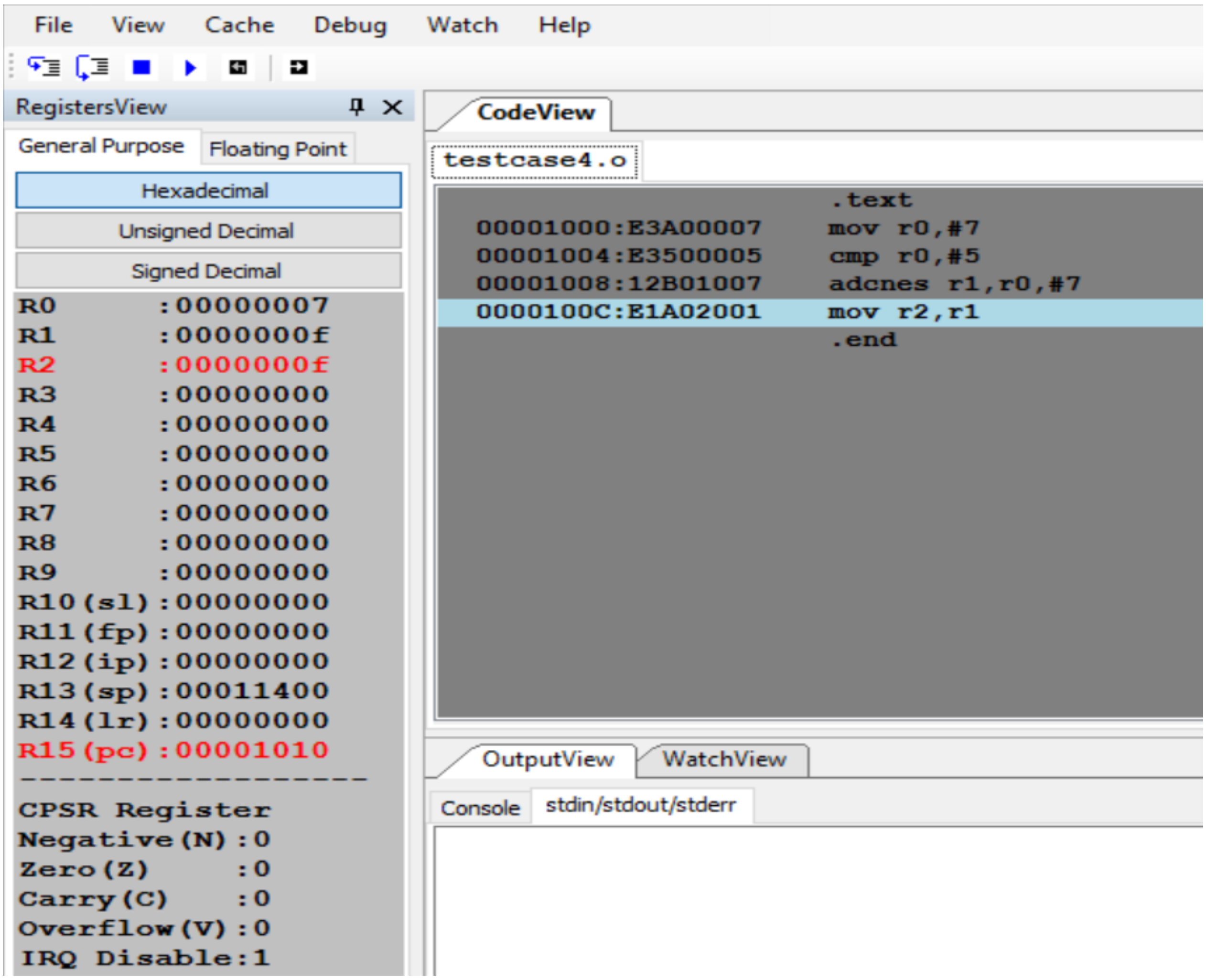
```
R0        :00000002
R1        :00000003
R2        :00000005
R3        :00000005
R4        :00000000
R5        :00000000
R6        :00000000
R7        :00000000
R8        :00000000
R9        :00000000
R10(sl)   :00000000
R11(fp)   :00000000
R12(ip)   :00000000
R13(sp)   :00011400
R14(lr)   :00000000
R15(pc)   :00001014
------------------
CPSR Register
Negative(N) :0
Zero(Z)     :1
Carry(C)    :1
Overflow(V) :0
```

```
                         .text
00001000:E3A00002       mov r0,#2
00001004:E3A01003       mov r1,#3
00001008:E0812000       add r2,r1,r0
0000100C:E3520005       cmp r2,#5
00001010:01A03002       moveq r3,r2
                         .end
```



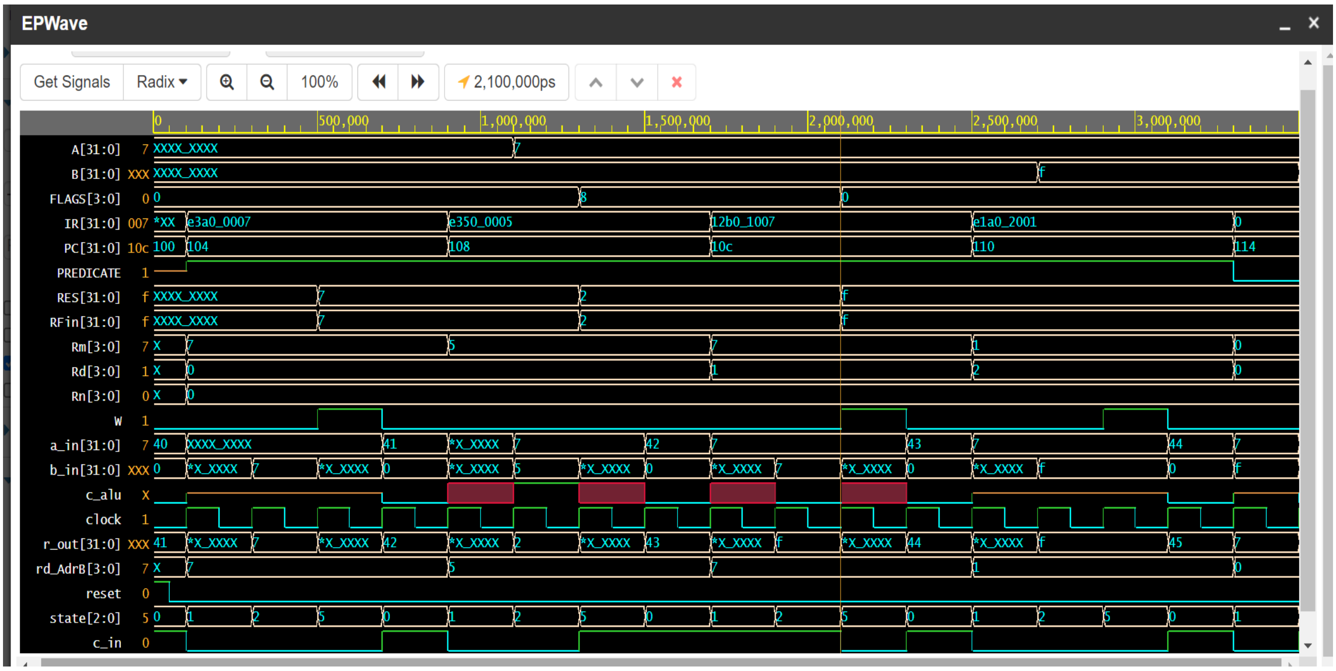
EPWave
Get Signals
Radix
100%
2,100,000ps
0
500,000
1,000,000
1,500,000
2,000,000
2,500,000
3,000,000
3,500,000
A[31:0] 3 XXXX_XXXX 2 3 5 2
B[31:0] 2 XXXX_XXXX 2 XXXX_XXXX 5
FLAGS[3:0] 0 0 a
IR[31:0] 000 *XX e3a0_0002 e3a0_1003 e081_2000 e352_0005 1a0_3002
PC[31:0] 10c 100 104 108 10c 110 114
PREDICATE 1
RES[31:0] 5 XXXX_XXXX 2 3 5 0 5
RFin[31:0] 5 XXXX_XXXX 2 3 5 0 5
Rm[3:0] 0 X 2 3 0 5 2
Rd[3:0] 2 X 0 1 2 0 3
Rn[3:0] 1 X 0 1 2 0
W 1
a_in[31:0] 3 40 XXXX_XXXX 41 *_XXXX 2 42 2 3 43 3 5 44 5 2 45
b_in[31:0] 2 0 *_XXXX 2 *_XXXX 0 *_XXXX 3 *_XXXX 0 *_XXXX 2 0 2 5 *_XXXX 0 *_XXXX 5 0
c_alu 0
clock 1
r_out[31:0] 5 41 *_XXXX 2 *_XXXX 42 *_XXXX 3 *_XXXX 43 *_XXXX 5 44 1 0 *_XXXX 45 *_XXXX 5 46
rd_AdrB[3:0] 0 X 2 3 0 5 2
reset 0
state[2:0] 5 0 1 2 5 0 1 2 5 0 1 2 5 0 1 2 5 0 1 2 5 0
c_in 0
RADDRESS2[3:0] 0 X 2 3 0 5 2

Example-4



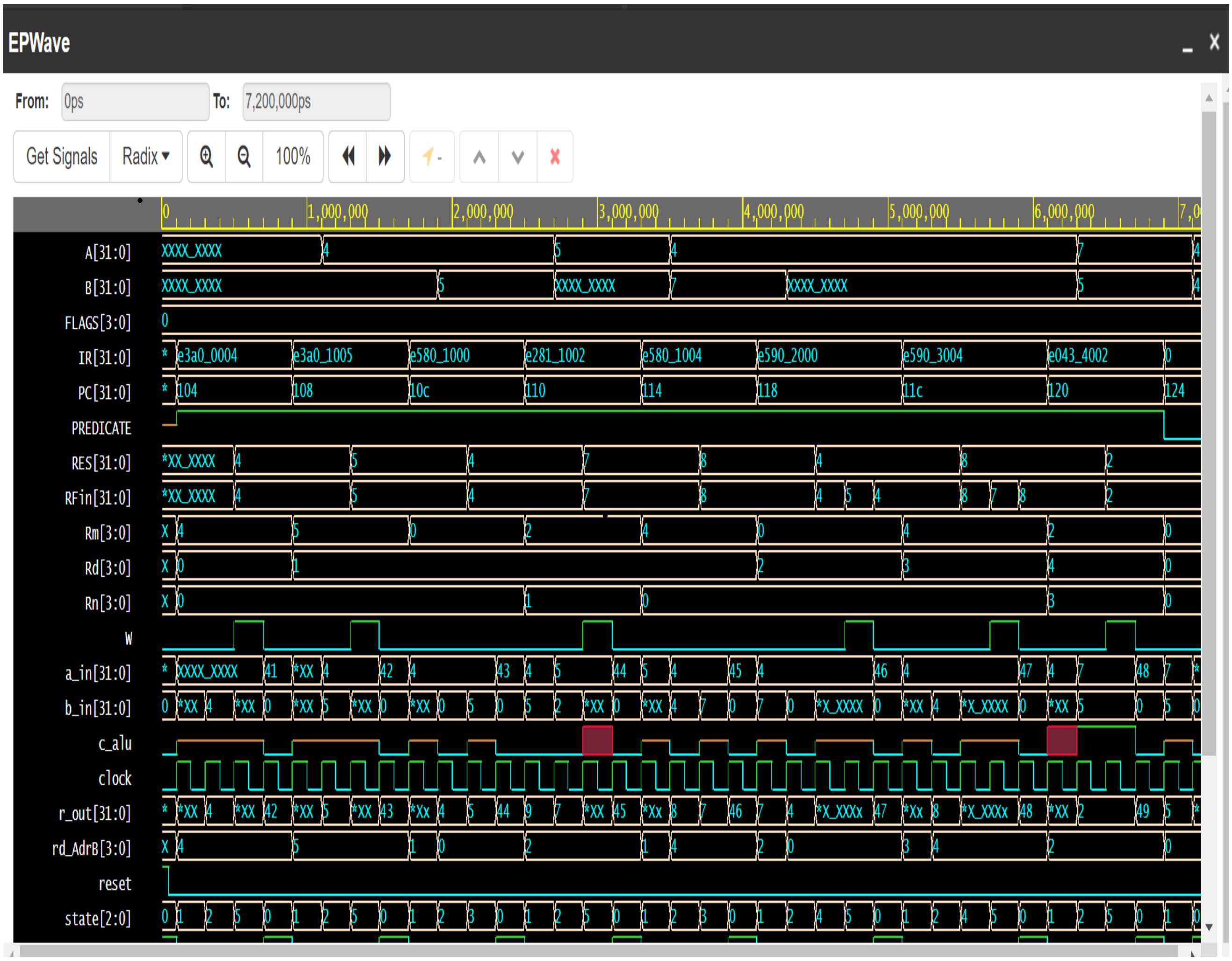
```
R0         :00000007
R1         :0000000f
R2         :0000000f
R3         :00000000
R4         :00000000
R5         :00000000
R6         :00000000
R7         :00000000
R8         :00000000
R9         :00000000
R10(sl)    :00000000
R11(fp)    :00000000
R12(ip)    :00000000
R13(sp)    :00011400
R14(lr)    :00000000
R15(pc)    :00001010
------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :0
Overflow(V):0
IRQ Disable:1
```

```
testcase4.o
                            .text
00001000:E3A00007           mov r0,#7
00001004:E3500005           cmp r0,#5
00001008:12B01007           adcnes r1,r0,#7
0000100C:E1A02001           mov r2,r1
                            .end
```



Example-5

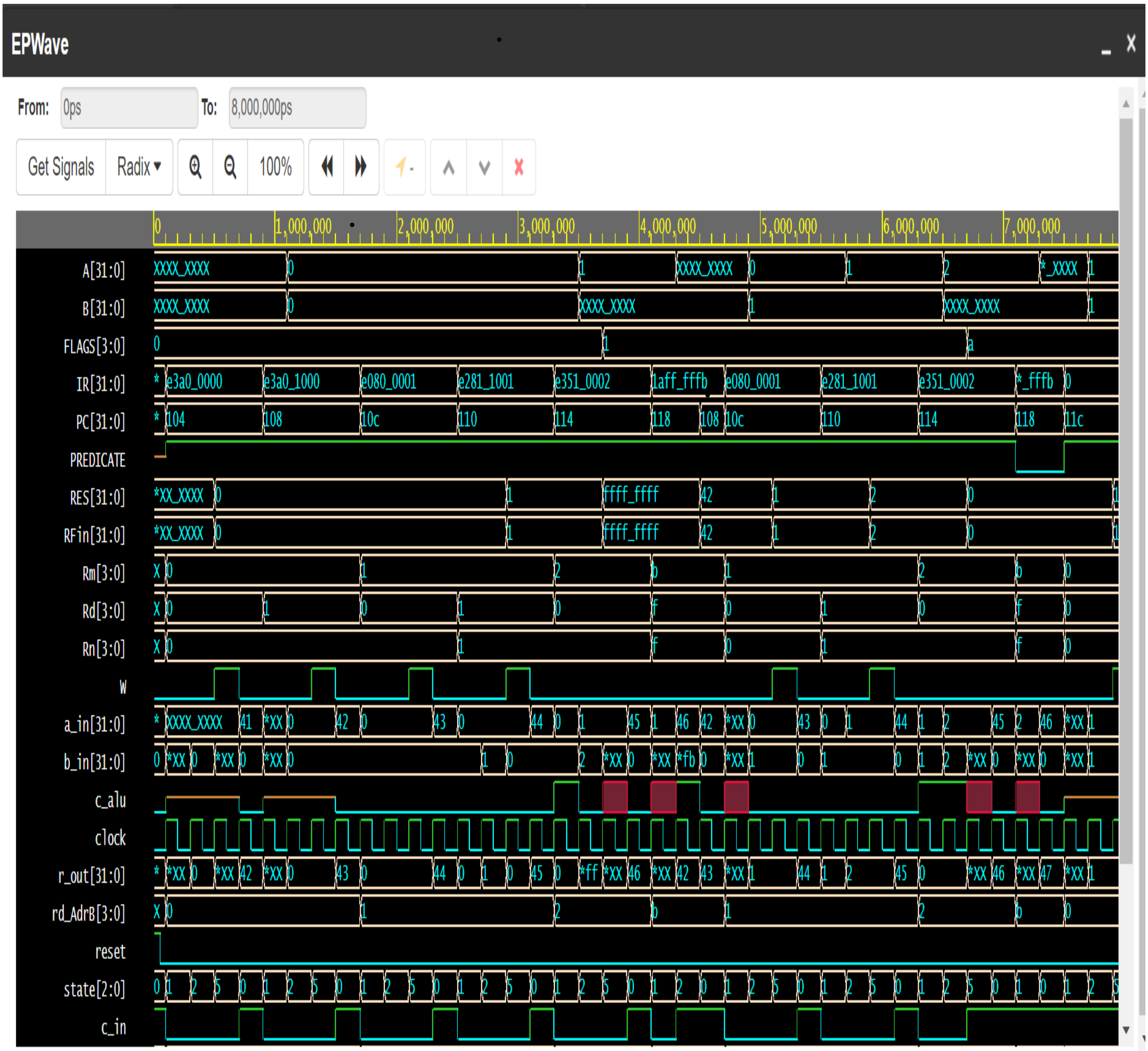
signal mem_array : memory_type := (0 => X"E3A0000~~A~~4", 1 => X"E3A01005", 2 => X"E5801000", 3 => X"E2811002", 4 => X"E5801004", 5 => X"E5902000", 6 => X"E5903004", 7 => X"E0434002", others => X"00000000");	.text mov r0, #~~10~~ 4 mov r1, #5 str r1, [r0] add r1, r1, #2 str r1, [r0, #4] ldr r2, [r0] ldr r3, [r0, #4] sub r4, r3, r2 .end



Example-6

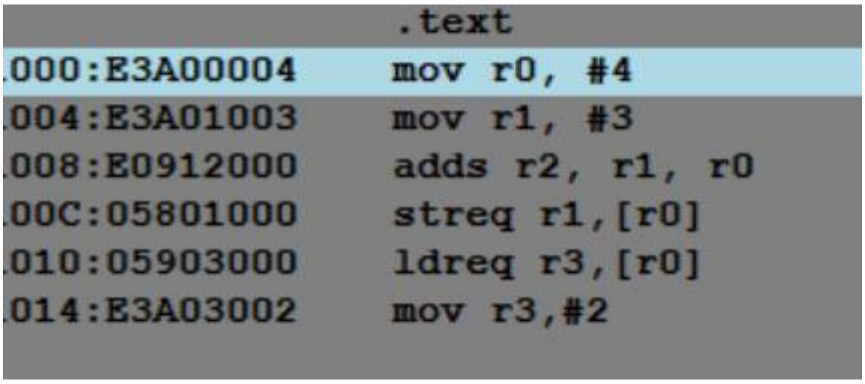
```
signal mem_array : memory_type :=
        (0 => X"E3A00000",
         1 => X"E3A01000",
         2 => X"E0800001",
         3 => X"E2811001",
         4 => X"E351000~~2~~1",
         5 => X"1AFFFFFB",
         others => X"00000000"
        );
```

```
        .text
        mov r0, #0
        mov r1, #0
Loop: add r0, r0, r1
        add r1, r1, #1
        cmp r1, #~~2~~1
        bne Loop
        .end
```

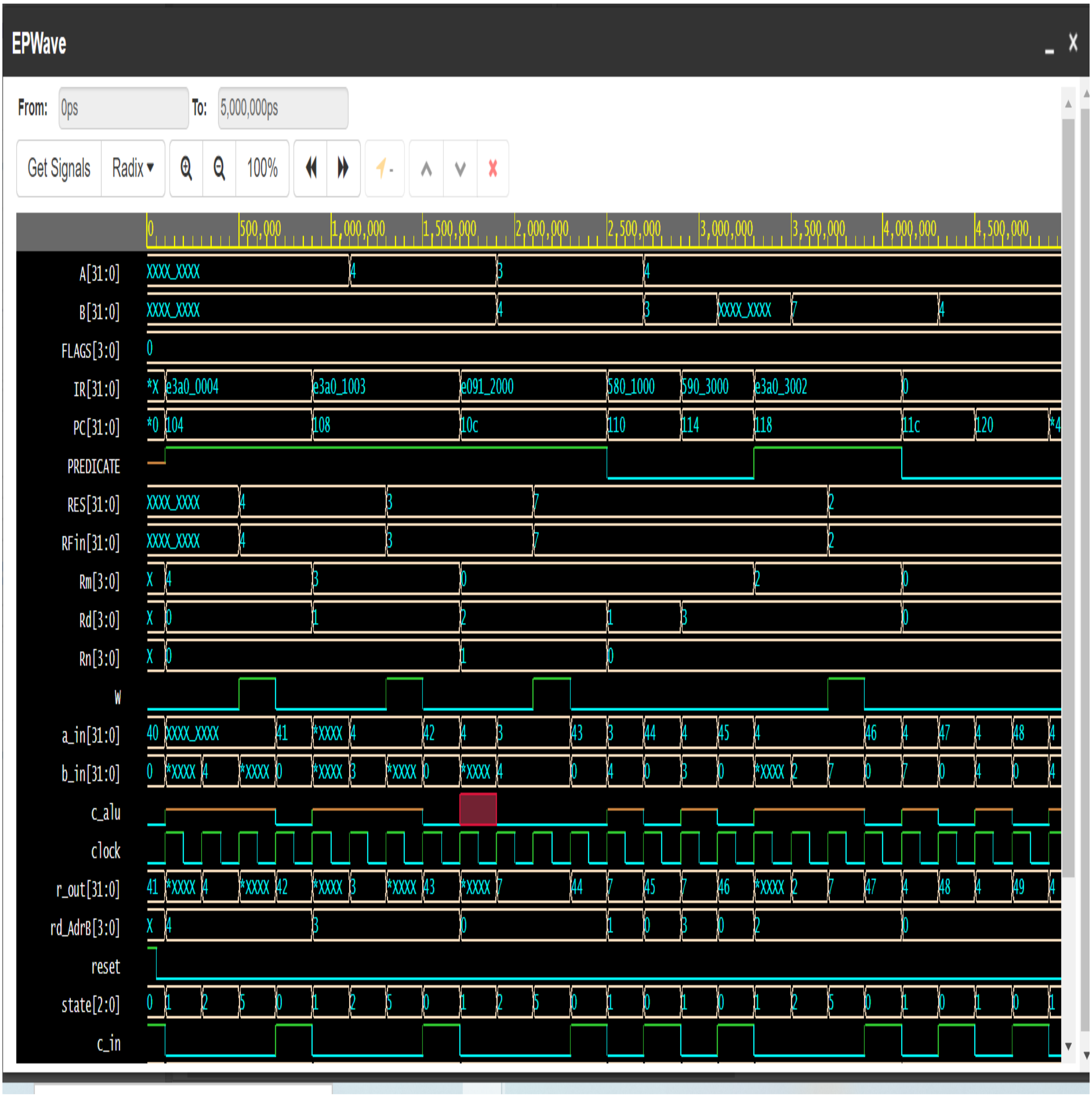


From: 0ps To: 8,000,000ps

Example-7



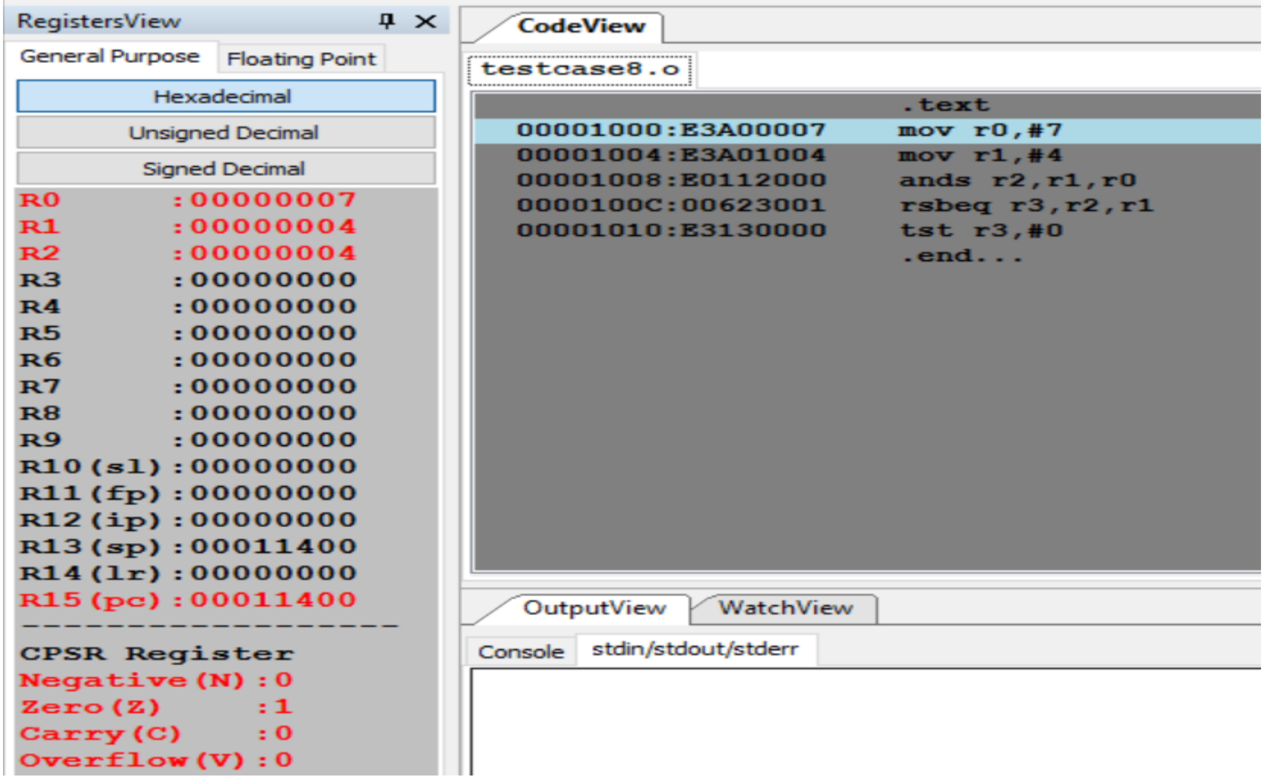
```
                        .text
000:E3A00004        mov r0, #4
004:E3A01003        mov r1, #3
008:E0912000        adds r2, r1, r0
00C:05801000        streq r1,[r0]
010:05903000        ldreq r3,[r0]
014:E3A03002        mov r3,#2
```



From: 0ps To: 5,000,000ps

EPWave
Get Signals
Radix ▾
100%
0
500,000
1,000,000
1,500,000
2,000,000
2,500,000
3,000,000
3,500,000
4,000,000
4,500,000
A[31:0] XXXX_XXXX 4 3 4
B[31:0] XXXX_XXXX 4 3 XXXX_XXXX 7 4
FLAGS[3:0] 0
IR[31:0] *X e3a0_0004 e3a0_1003 e091_2000 580_1000 590_3000 e3a0_3002 0
PC[31:0] *0 104 108 10c 110 114 118 11c 120 *4
PREDICATE
RES[31:0] XXXX_XXXX 4 3 7 2
RFin[31:0] XXXX_XXXX 4 3 7 2
Rm[3:0] X 4 3 0 2 0
Rd[3:0] X 0 1 2 1 3 0
Rn[3:0] X 0 1 0
W
a_in[31:0] 40 XXXX_XXXX 41 *XXXX 4 42 4 3 43 3 44 4 45 4 46 4 47 4 48 4
b_in[31:0] 0 *XXXX 4 *XXXX 0 *XXXX 3 *XXXX 0 *XXXX 4 0 4 0 3 0 *XXXX 2 7 0 7 0 4 0 4
c_alu
clock
r_out[31:0] 41 *XXXX 4 *XXXX 42 *XXXX 3 *XXXX 43 *XXXX 7 44 7 45 7 46 *XXXX 2 7 47 4 48 4 49 4
rd_AdrB[3:0] X 4 3 0 1 0 3 0 2 0
reset
state[2:0] 0 1 2 5 0 1 2 5 0 1 2 5 0 1 0 1 0 1 2 5 0 1 0 1 0 1
c_in

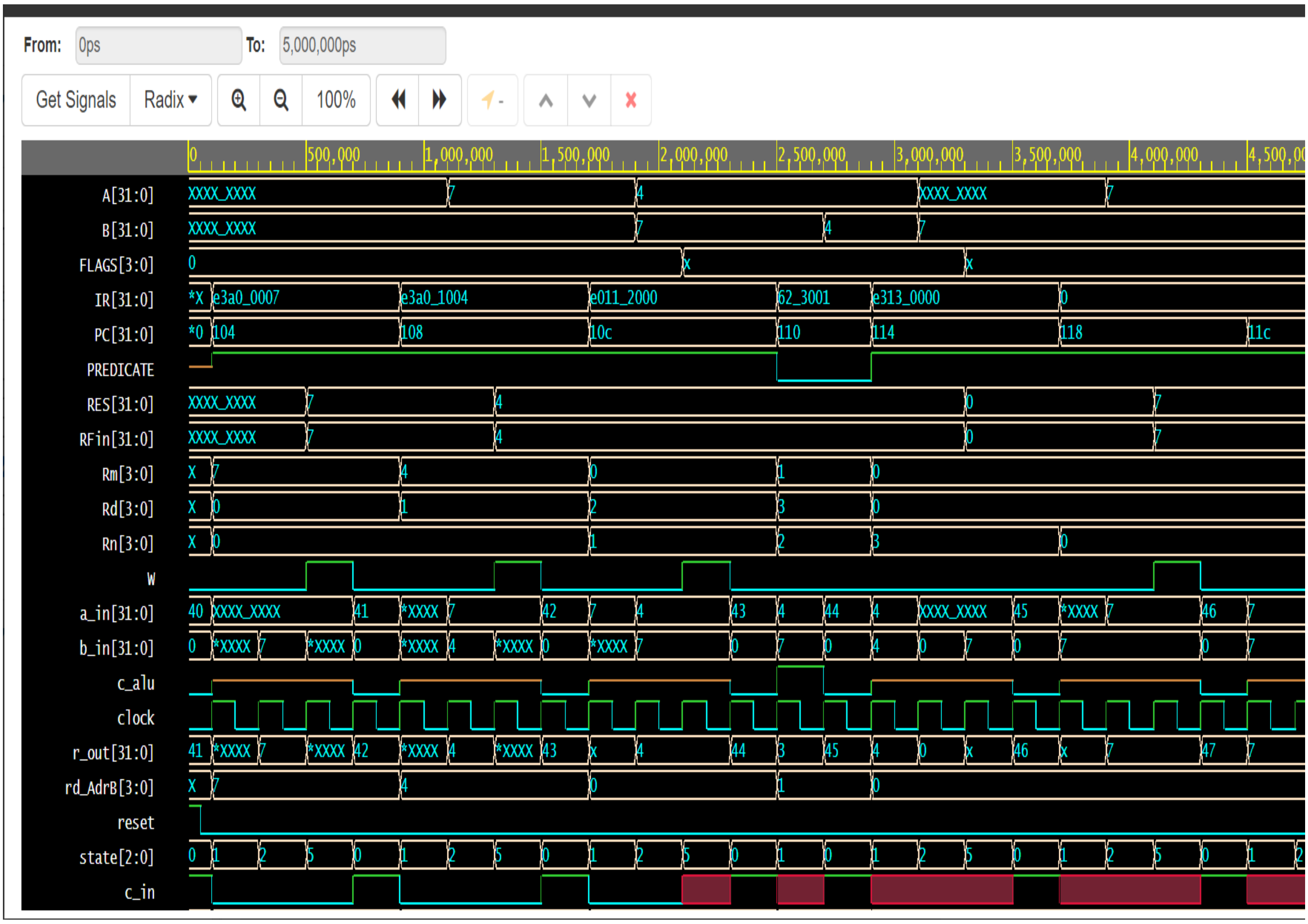
Example8



```
R0       :00000007
R1       :00000004
R2       :00000004
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)    :0
Overflow(V) :0
```

testcase8.o

```
                        .text
00001000:E3A00007       mov r0,#7
00001004:E3A01004       mov r1,#4
00001008:E0112000       ands r2,r1,r0
0000100C:00623001       rsbeq r3,r2,r1
00001010:E3130000       tst r3,#0
                        .end...
```



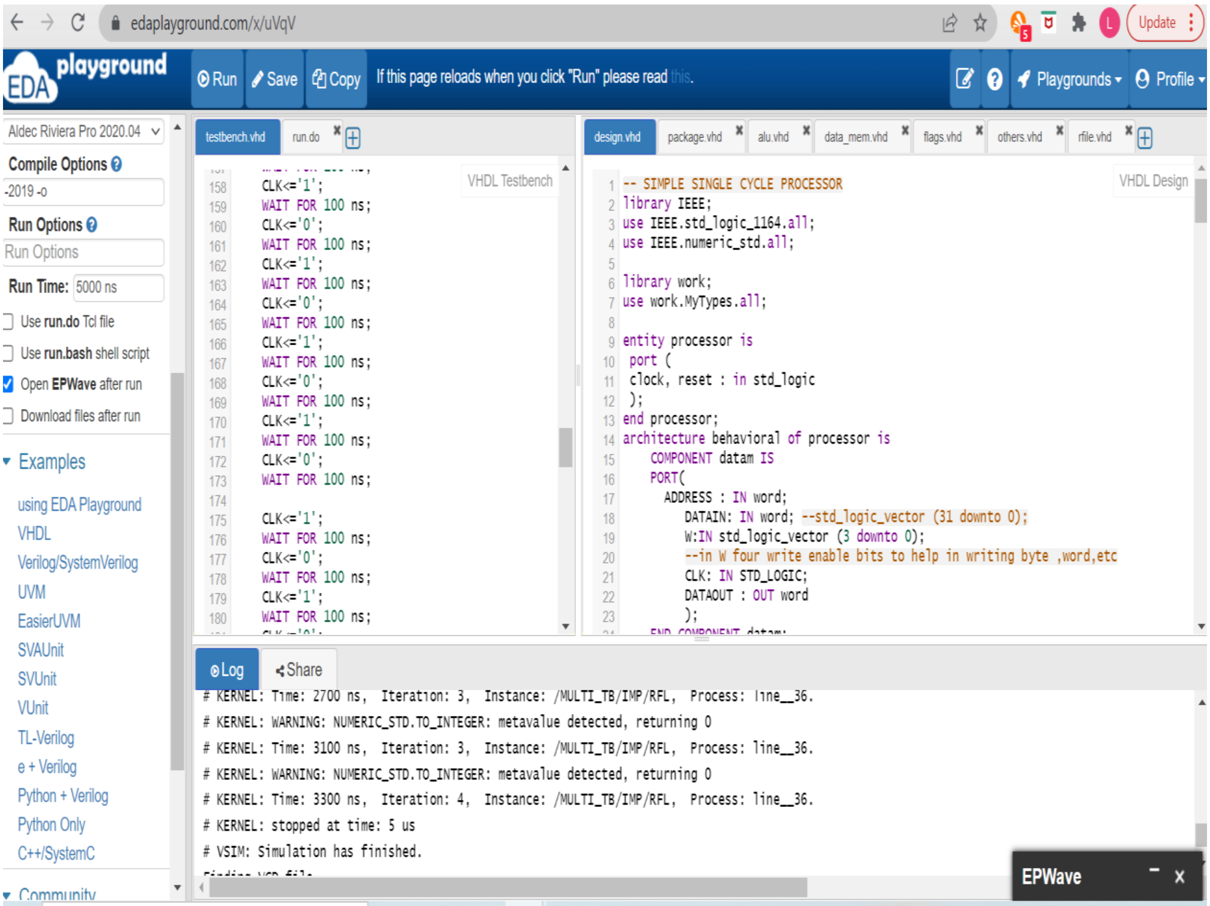
From: 0ps To: 5,000,000ps

For each module,work and analysis I have put files in edaplayground like below-

Used for simulation- Aldec riviera pro 2020.04

Used for synthesis- Mentor Precision 2021.1

simulation settings For eg testcase-2-



Aldec Riviera Pro 2020.04

Compile Options: -2019 -o

Run Options:

Run Time: 5000 ns

□ Use run.do Tcl file

□ Use run.bash shell script

☑ Open EPWave after run

□ Download files after run

```
    CLK<='1';
    WAIT FOR 100 ns;
    CLK<='0';
    WAIT FOR 100 ns;
    CLK<='1';
    WAIT FOR 100 ns;
    CLK<='0';
    WAIT FOR 100 ns;
    CLK<='1';
    WAIT FOR 100 ns;
    CLK<='0';
    WAIT FOR 100 ns;
    CLK<='1';
    WAIT FOR 100 ns;
    CLK<='0';
    WAIT FOR 100 ns;

    CLK<='1';
    WAIT FOR 100 ns;
    CLK<='0';
    WAIT FOR 100 ns;
    CLK<='1';
    WAIT FOR 100 ns;
```

```
-- SIMPLE SINGLE CYCLE PROCESSOR
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

library work;
use work.MyTypes.all;

entity processor is
 port (
 clock, reset : in std_logic
 );
end processor;
architecture behavioral of processor is
    COMPONENT datam IS
    PORT(
      ADDRESS : IN word;
         DATAIN: IN word; --std_logic_vector (31 downto 0);
         W:IN std_logic_vector (3 downto 0);
         --in W four write enable bits to help in writing byte ,word,etc
         CLK: IN STD_LOGIC;
         DATAOUT : OUT word
         );
```

```
# KERNEL: Time: 2700 ns,  Iteration: 3,  Instance: /MULTI_TB/IMP/RFL,  Process: line__36.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 3100 ns,  Iteration: 3,  Instance: /MULTI_TB/IMP/RFL,  Process: line__36.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 3300 ns,  Iteration: 4,  Instance: /MULTI_TB/IMP/RFL,  Process: line__36.
# KERNEL: stopped at time: 5 us
# VSIM: Simulation has finished.
```

For more testcases and live running ,edaplayground can be played in demo for this stage and public link of it can be shared.