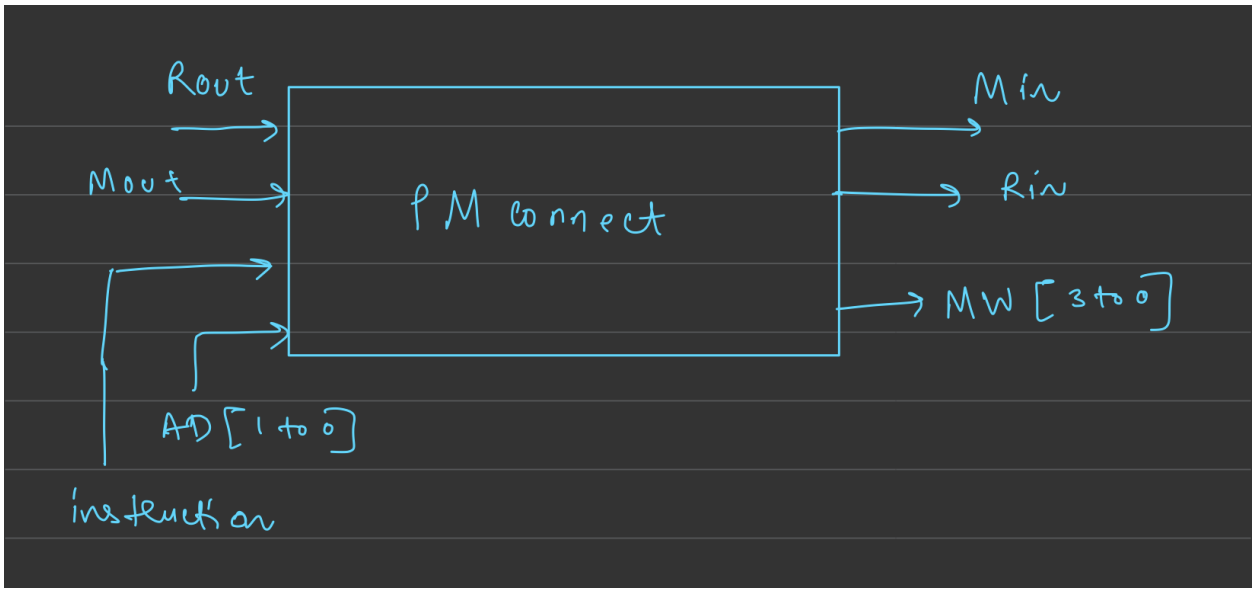
COL 216 Assignment 2.6

Chinmay Mittal (2020CS10336)

In this stage, we implement the remaining types of load-store instructions.

For this, we add a new component PMconnect as described in the assignment, I remove the control state input from this component.
This control state was intended to be used for giving appropriate values to MW (memory write).
To handle this I take logical and of Memory write signals from the FSM and PMconnect. This ensures MW takes correct values (from PMconnect) and are non-zero only in states in which write is expected (as governed by the value from the FSM).



The instruction input specifies the type of the DT instruction (ldr, str, ldrh, strh, ldrsh, ldrb, ldrsb, strb).
To ease the process, I have defined a new enumerated datatype for the same in MyTypes.vhd
I have added additional logic in the Decoder to figure out whether the instruction is DT and what type of DT instruction it is (here I handle the fact that there is a different format for signed and half-word data transfers)

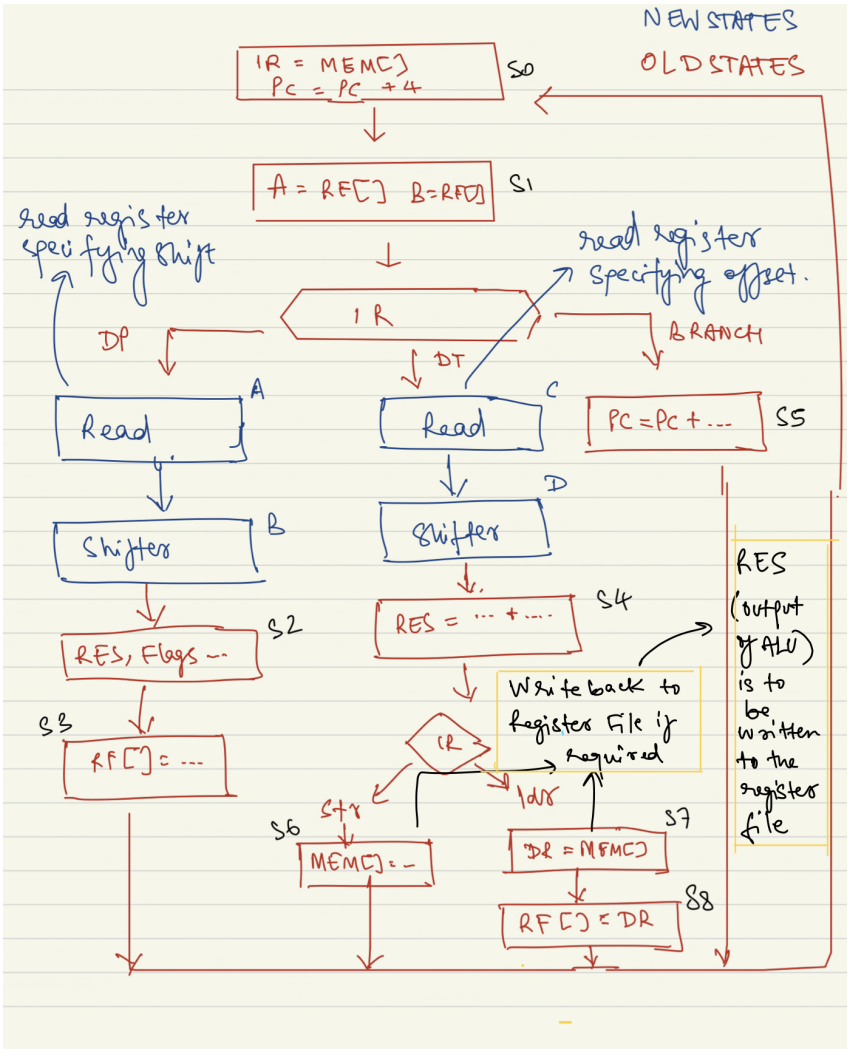
```
DT_instr <= ldr when ( instruction( 27 downto 26) = "01" and instruction(20) = '1' and instruction(22) = '0' ) else
            str when ( instruction( 27 downto 26) = "01" and instruction(20) = '0' and instruction(22) = '0') else
            ldrb when ( instruction( 27 downto 26) = "01" and instruction(20) = '1' and instruction(22) = '1' ) else
            strb when ( instruction( 27 downto 26) = "01" and instruction(20) = '0' and instruction(22) = '1' ) else
            ldrh when ( instruction( 27 downto 25 ) = "000" and instruction(20) = '1' and instruction( 7 downto 4 ) = "1011" ) else
            strh when ( instruction( 27 downto 25 ) = "000" and instruction(20) = '0' and instruction( 7 downto 4 ) = "1011") else
            ldrsh when ( instruction( 27 downto 25 ) = "000" and instruction(20) = '1' and instruction( 7 downto 4 ) = "1111") else
            ldrsb ;
```

```
instr_class <= BRN when ( instruction( 27 downto 26 ) = "10") else
               DT when ( (instruction( 27 downto 26) = "01" ) or ( (instruction( 27 downto 25) = "000" ) and ( instruction(4) = '1') and ( instruction(7) = '1' ) ) )  else
               DP when ( instruction( 27 downto 26) = "00" ) else
               none ;
```

To implement auto-indexing I have added a new register (AddressForDT) that stores the address actually used for data transfers i.e base or base plus offset. This is now fed to the memory instead of the ALU output as before. The value clocked in this register is determined by the PreIndexing signal. It is clocked along with the ALU output (signal is the same for both).

```
process(clk)
begin
if( rising_edge(clk) and ReW = '1') then
RES <= ALU_output ;
    -- to accomodate pre/post indexing
    if(PreIndexing = '1') then
    AddressForDT <= ALU_output ; -- base + offset
    else
    AddressForDT <= A ; -- base
    end if ;
end if ;
```

To implement write back I do not introduce new states and I use the old states themselves to write to the register file (controlled by the WriteBack signal). I add an additional control signal to control the address of the register which is to be written to the register file.(RegisterWriteSrc). WriteBack is always true in the case of Post Indexing irrespective of the W bit.



Connections of PM connect

The type of DT instruction is fed in from the output of the Decoder.
Rout is fed in by B (output of reading from register file)
Mout is fed DR (stores the data read from the memory)
AddressForDT is used to supply the byte address (AD)
Rin and Min are fed back to the Register file and Memory instead of DR and B as before

Second Operand for signed and half-word transfer

The second operand is also in a bit different format in the case of half-word and signed data transfers.
I have a signal which determines if the DT instruction is normal (ldr, ldrb, strb, str) or the other types (signed and half-word transfers). This is used in the multiplexing for the shifter which prepares the second operand.
The shift amount for signed and half-word data transfers are always zero. The data can come as immediate (concatenate bits 11 to 8 with bits 3 to 0)or from registers (specified by bits 3 to 0)

```
HalfWordOrSignedTransfer <= '0' when ( DT_instr = ldr or DT_instr = str or DT_instr = ldrb or DT_instr = strb ) else '1' ;
HlfWdOrSgnedTrnsferSrc <= IR(22) ; -- 1 when offset is immediate else 0 when offset is specified by register
```

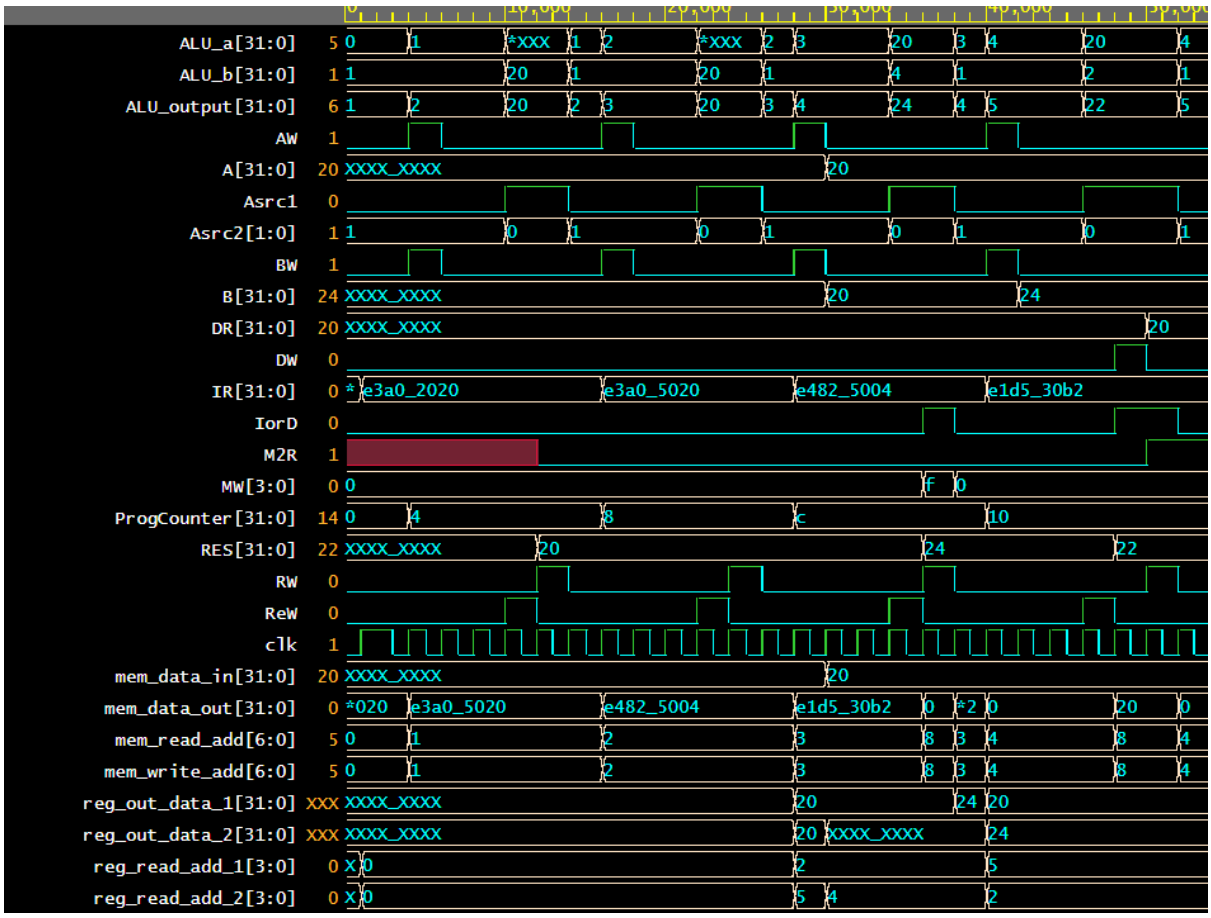
Test Cases

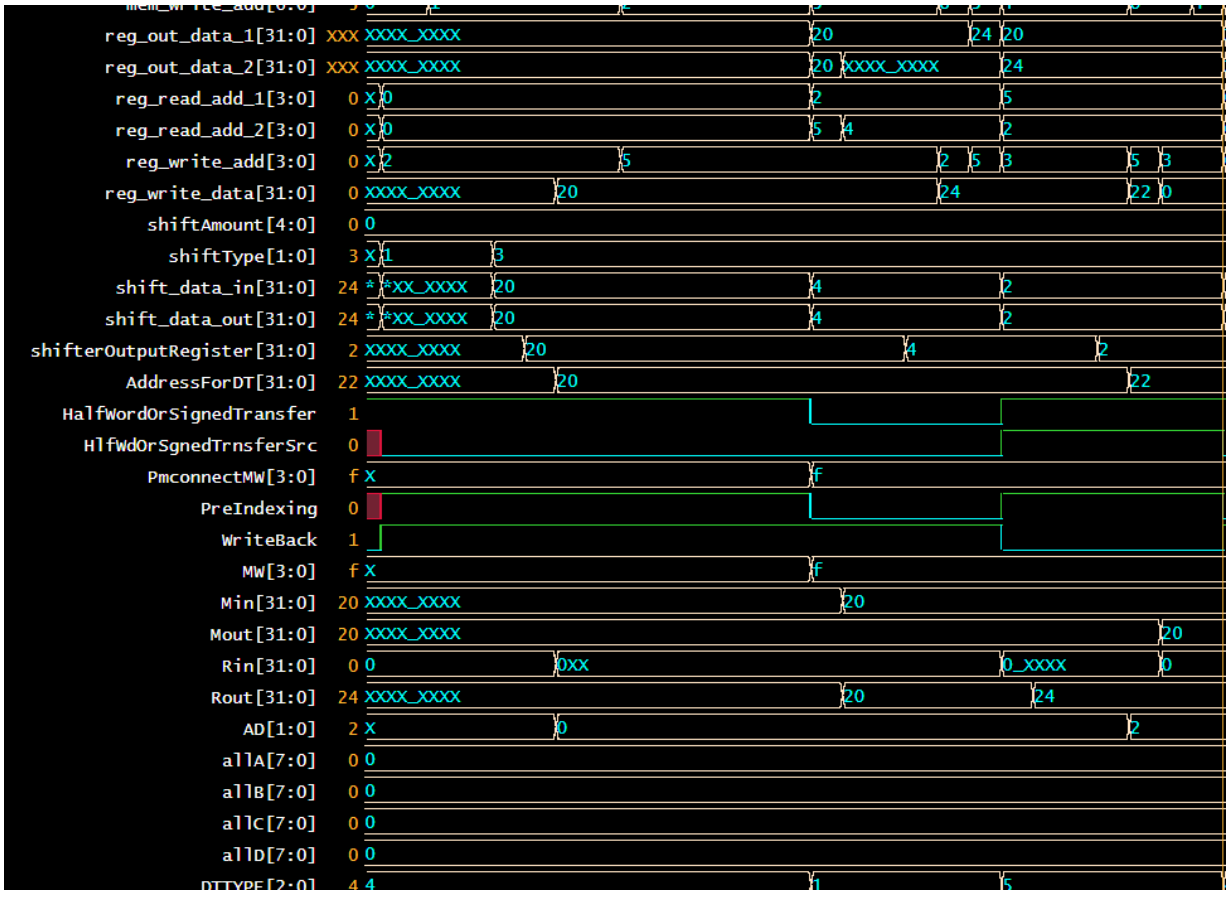
test1


ALU_a[31:0] 5
ALU_b[31:0] 1
ALU_output[31:0] 6
AW 1
A[31:0] 20
BW 1
B[31:0] XXX
DR[31:0] 80
DT_offsetReg[31:0] XXX
DT_offsetW 0
DW 0
IR[31:0] 0
MW[3:0] 0
ProgCounter[31:0] 14
RES[31:0] 20
RW 0
ReW 0
Rs[31:0] XXX
clk 1
mem_data_in[31:0] 80
mem_data_out[31:0] 0
mem_read_add[6:0] 5
mem_write_add[6:0] 5
predicate X
reg_out_data_1[31:0] XXX
reg_out_data_2[31:0] XXX
reg_read_add_1[3:0] 0
reg_read_add_2[3:0] 0
reg_write_add[3:0] 0
e3a0_1080
e3a0_2020
e582_1000
e1d2_10d0

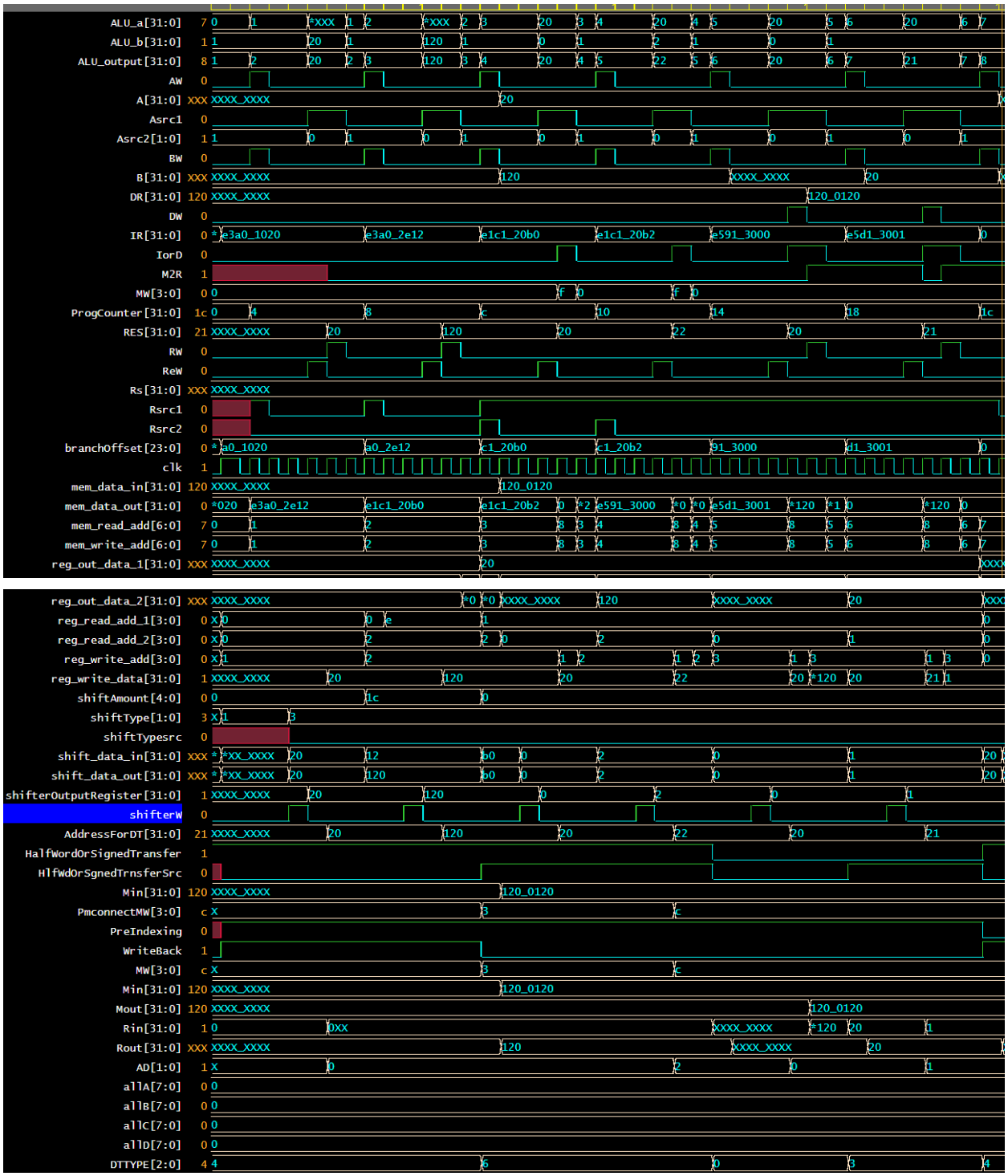

reg_read_add_1[3:0] 0
reg_read_add_2[3:0] 0
reg_write_add[3:0] 0
reg_write_data[31:0] f80
shiftAmount[4:0] 0
shiftType[1:0] 3
shiftTypesrc 0
shift_data_in[31:0] XXX
shift_data_out[31:0] XXX
ifterOutputRegister[31:0] 0
shifterW 0
AddressForDT[31:0] 20
HalfWordOrSignedTransfer 1
HlfWdOrSgnedTrnsferSrc 0
Min[31:0] 80
PmconnectMW[3:0] f
PreIndexing 0
WriteBack 1
MW[3:0] f
Min[31:0] 80
Mout[31:0] 80
Rin[31:0] f80
Rout[31:0] XXX
AD[1:0] 0
allA[7:0] ff
allB[7:0] 0
allC[7:0] 0
allD[7:0] 0
DTTYPE[2:0] 4

Test2

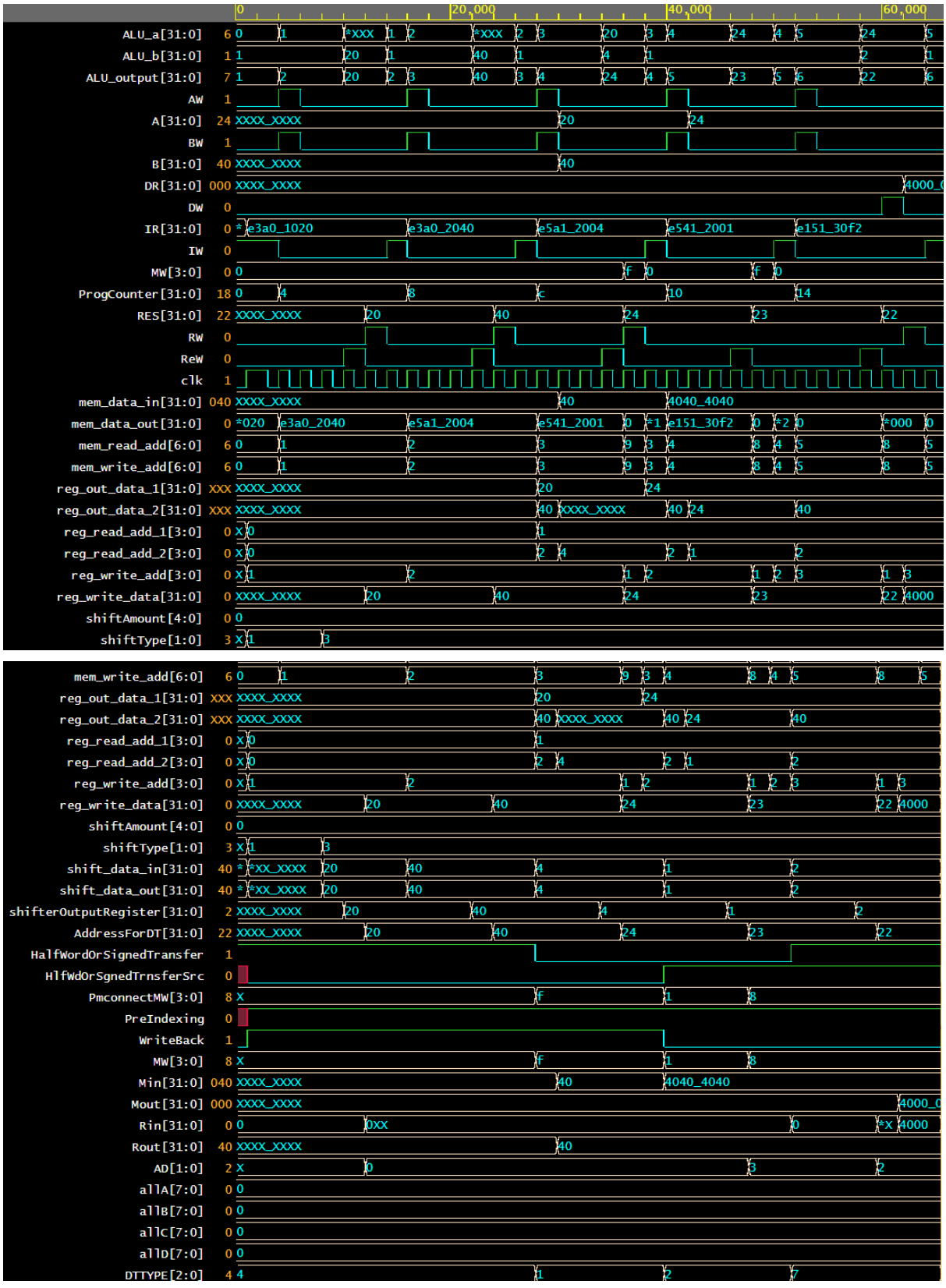

ALU_a[31:0] 5
ALU_b[31:0] 1
ALU_output[31:0] 6
AW 1
A[31:0] 20
Asrc1 0
Asrc2[1:0] 1
BW 1
B[31:0] 24
DR[31:0] 20
DW 0
IR[31:0] 0
IorD 0
M2R 1
MW[3:0] 0
ProgCounter[31:0] 14
RES[31:0] 22
RW 0
ReW 0
clk 1
mem_data_in[31:0] 20
mem_data_out[31:0] 0
mem_read_add[6:0] 5
mem_write_add[6:0] 5
reg_out_data_1[31:0] xxx
reg_out_data_2[31:0] xxx
reg_read_add_1[3:0] 0
reg_read_add_2[3:0] 0
e3a0_2020
e3a0_5020
e482_5004
e1d5_30b2


reg_out_data_1[31:0] xxx
reg_out_data_2[31:0] xxx
reg_read_add_1[3:0] 0
reg_read_add_2[3:0] 0
reg_write_add[3:0] 0
reg_write_data[31:0] 0
shiftAmount[4:0] 0
shiftType[1:0] 3
shift_data_in[31:0] 24
shift_data_out[31:0] 24
shifterOutputRegister[31:0] 2
AddressForDT[31:0] 22
HalfWordOrSignedTransfer 1
HlfWdOrSgnedTrnsferSrc 0
PmconnectMW[3:0] f
PreIndexing 0
WriteBack 1
MW[3:0] f
Min[31:0] 20
Mout[31:0] 20
Rin[31:0] 0
Rout[31:0] 24
AD[1:0] 2
allA[7:0] 0
allB[7:0] 0
allC[7:0] 0
allD[7:0] 0

Test3


ALU_a[31:0]
ALU_b[31:0]
ALU_output[31:0]
AW
A[31:0]
Asrc1
Asrc2[1:0]
BW
B[31:0]
DR[31:0]
DW
IR[31:0]
IorD
M2R
MW[3:0]
ProgCounter[31:0]
RES[31:0]
RW
ReW
Rs[31:0]
Rsrc1
Rsrc2
branchOffset[23:0]
clk
mem_data_in[31:0]
mem_data_out[31:0]
mem_read_add[6:0]
mem_write_add[6:0]
reg_out_data_1[31:0]
reg_out_data_2[31:0]
reg_read_add_1[3:0]
reg_read_add_2[3:0]
reg_write_add[3:0]
reg_write_data[31:0]
shiftAmount[4:0]
shiftType[1:0]
shiftTypesrc
shift_data_in[31:0]
shift_data_out[31:0]
shifterOutputRegister[31:0]
shifterW
AddressForDT[31:0]
HalfWordOrSignedTransfer
HlfWdOrSgnedTrnsferSrc
Min[31:0]
PmconnectMW[3:0]
PreIndexing
WriteBack
MW[3:0]
Min[31:0]
Mout[31:0]
Rin[31:0]
Rout[31:0]
AD[1:0]
allA[7:0]
allB[7:0]
allC[7:0]
allD[7:0]
DTTYPE[2:0]

Test4


0
20,000
40,000
60,000
ALU_a[31:0]
ALU_b[31:0]
ALU_output[31:0]
AW
A[31:0]
BW
B[31:0]
DR[31:0]
DW
IR[31:0]
IW
MW[3:0]
ProgCounter[31:0]
RES[31:0]
RW
ReW
clk
mem_data_in[31:0]
mem_data_out[31:0]
mem_read_add[6:0]
mem_write_add[6:0]
reg_out_data_1[31:0]
reg_out_data_2[31:0]
reg_read_add_1[3:0]
reg_read_add_2[3:0]
reg_write_add[3:0]
reg_write_data[31:0]
shiftAmount[4:0]
shiftType[1:0]
shift_data_in[31:0]
shift_data_out[31:0]
shifterOutputRegister[31:0]
AddressForDT[31:0]
HalfWordOrSignedTransfer
HlfWdOrSgnedTrnsferSrc
PmconnectMW[3:0]
PreIndexing
WriteBack
Min[31:0]
Mout[31:0]
Rin[31:0]
Rout[31:0]
AD[1:0]
allA[7:0]
allB[7:0]
allC[7:0]
allD[7:0]
DTTYPE[2:0]