Lab Assignment 2, Stage 5: Support for shift and rotate features

Kushal Kumar Gupta, 2020CS10355

ARM instruction set provides the following shift/rotate features in DP and DT instructions

Files, entities and Architectures:

Changes from stage 4-

`I.Processor_stage5.vhd`:

FSM now has 11 states to incorporate the shifter module.

From state 2, DP and DT instructions can transition to state 10 or 11.

- state 10: reached if the second operand/offset is a register and the shift amount is specified by a register. The shift amount is stored in register X at the end of this stage. State 10 transitions to state 11.
- state 11: the shift amount is available (either in X or immediate) and the result of the shifter module is stored in register D at the end of this stage. From here, transition to state 3 for DP and state 4 for DT instructions.

Other changes-

1. In state 2, DT instructions now also load register contents of register specified by bits 3-0 of instruction in register B. This is done to get the base offset amount if offset is stored in the register, and apply shift/rotate operations later.
2. For DT instructions, now the data from register file corresponding to bits 15 -12 of instruction (used for storing in memory in `str` instructions) is fetched in register B at the end of state 4.
3. Full predication is now supported. This is done by checking if condition is satisfied at the end of stage 2, and if the condition is not satisifed, then the FSM transitions to state 1.
4. op2 of ALU is now either register D (for DP and DT instructions) or 0 for stage 1 PC update or PC offset for stage 5 PC update.
5. FlagUpdater now takes shifter_carry_out as well for updating flags.
6. Suitable glue logic provided for incorporating the shifter module in the design. Deciding the input, shift type, shift amount for shifter depending on source of operand/offset and source of shift amount done.

For implementation, see the architecture `pro_beh` of `processor`.

`II.FlagUpdater_stage5.vhd`:

```
entity FlagUpdater is
Port (
    CLK: in bit;
   DP_subclass : in DP_subclass_type;  --multiply instructions not included
    Fset : in std_logic; --if this is 1 then at the rising edge of the clock, flags are updated
    carry_ALU, carry_shifter, MSBop1, MSBop2: in std_logic; --carry bit generated by ALU, carry
bit generated by shifter, MSB bits of operands of ALU (after considering + operation for arith
and comp subclass, for calculating V flag)
    res_ALU: in word; --result of ALU
    Z, V, C, N: out std_logic := '0'
);
end FlagUpdater;
```

Takes in another input signal for shift carry. Shift carry is used to update the Carry flag for DP Test and Logical Instructions in case of shift/rotate. Care has been taken so that if there is no shift/rotate, then the Shift carry is unchanged and hence the Carry flag is unchanged.

III.mytypes_stage5.vhd:

Changed name of DP_operand_src_type to src_type and included the following types to augment the decoder-

```
type load_store_type is (load, store);

type DT_offset_sign_type is (plus, minus);

type shift_rotate_op_type is (LSL, LSR, ASR, RORrot); --using RORrot as ROR is a vhdl keyword
```

IV.Decoder_stage5.vhd:

```
entity Decoder is
Port (
    instruction : in word;
    instr_class : out instr_class_type;
    operation : out optype;
    DP_subclass : out DP_subclass_type;
    DP_operand_src : out src_type;
    load_store : out load_store_type;
    DT_offset_sign : out DT_offset_sign_type;
    DT_offset_src : out src_type;
    reg_shift_type : out shift_rotate_op_type;
    reg_shift_src : out src_type
);
end Decoder;
```

Extra output ports
DT_offset_src:whether register specifies the offset or 12 bit immediate
reg_shift_type: if register is the operand of DP/ offset of DT, what is the shift type
reg_shift_src:if register is the operand of DP/ offset of DT, is the shift amount given by a register or is a 5 bit immediate

V. shifter_stage5.vhd: Has entities shifter, shift1, shift2, shift4, shift8, shift16.

```
entity shifter is
Port (
   inp: in word;
    shift_type : in shift_rotate_op_type;
    shift_amount : in std_logic_vector(4 downto 0);
    carry_in: in std_logic;
    outp: out word;
    carry_out : out std_logic
);
end shifter;
```

`shifter` is the Shifter/rotator module which takes in `inp` and performs one of `LSL, LSR, ASR, ROR` specified by shift type, and the shift/rotate amount ranging from 0-31 is specified by `shift_amount`. The result of shift/rotate is ouput in `outp`. If carry is generated by shift/rotate then it is ouput in `carry_out`, otherwise `carry_out` is assigned `carry_in` if there is no shift/rotate performed.

Implementation details-

- Has 5 components `shift1, shift2, shift4, shift8, shift16`. Each module can perform `LSR, ASR, ROR` if select signal given to them is active. Each module can either generate a new carry if it is shifting/rotating or propagate the previous carry.
- These 5 components have been instantiated in the shifter architecture and connected appropriately. The select signal for each component is decided by considering the binary representation of the shift_amount. To handle `LSL`, the output is reversed before supplying to the shifters, `LSR` is applied for the same shift_amount and finally the result is reversed again for output. The entire circuit is combinational.

For implementation, see the architecture `shifter_beh` of `shifter`, and the architectures `shift1_beh` of `shift1`, `shift2_beh` of `shift2`, `shift4_beh` of `shift4`, `shift8_beh` of `shift8` and `shift16_beh` of `shift16`.

How to use:

On edaplayground.com, upload testbench.vhd and run.do in the left column, and ALU_stage3.vhd, RegFile_stage1.vhd, Mem_stage3.vhd, mytypes_stage5.vhd , decoder_stage5.vhd, FlagUpdater_stage5.vhd, conditionChecker_stage4.vhd, programCounter_stage3.vhd, processor_stage5.vhd, shifter_stage5.vhd. Copy contents in design.vhd section as given in the design.vhd file. Select Testbench + Design as VHDL. Then, for-

1.)Simulation

Type testbench in the Top entity. Select Aldec Riviera Pro 2020.04 simulator to simulate the design. Set the run time accordingly and select the EPWave option. Then Save and Run the simulation to get waves of the signals defined in these modules.

2.)Synthesis

Copy the VHDL file of the component you want to synthesise into design.vhd. (Only design.vhd entities are synthesised)

Then, select Mentor Precision 2021.1 to synthesise. Select netlist option to view the Verilog description. Then Save and Run to get the synthesis result. We get the report containing the resource table specifying number of IOs, LUTs, CLB slices, ports, nets, etc. used to implement this module in the given FPGA specified by run.do.

Results of EPWave (Simulation):

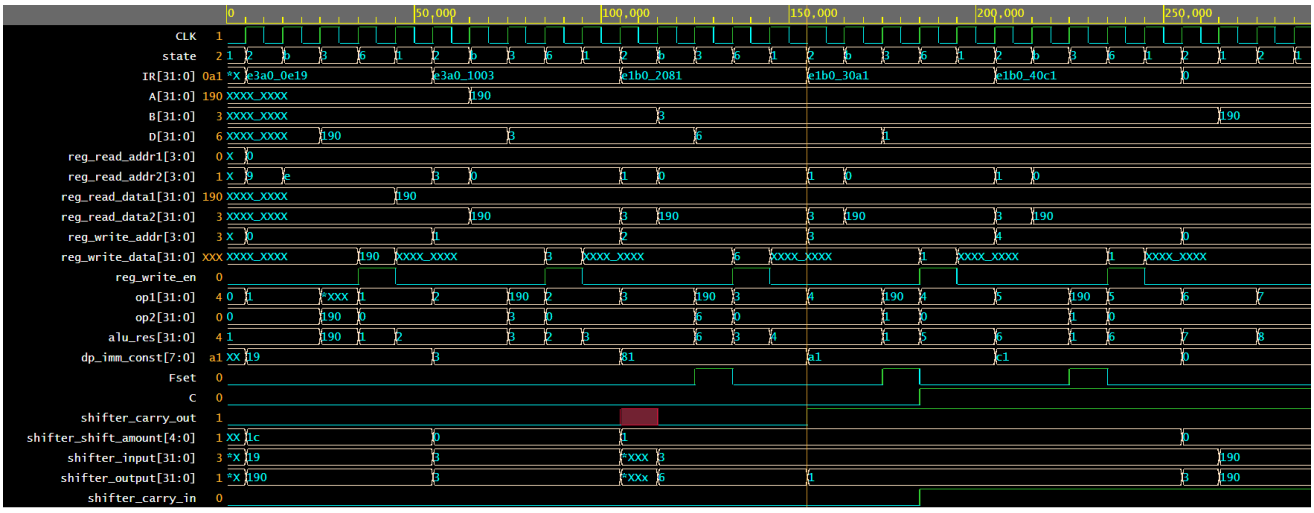
NOTE: Please ensure that Program memory is from address 0 to 63, data memory is from 64 to 127

Can see the input and output signals of processor against the clock.

Memory initialisation with different programs and EPWave-

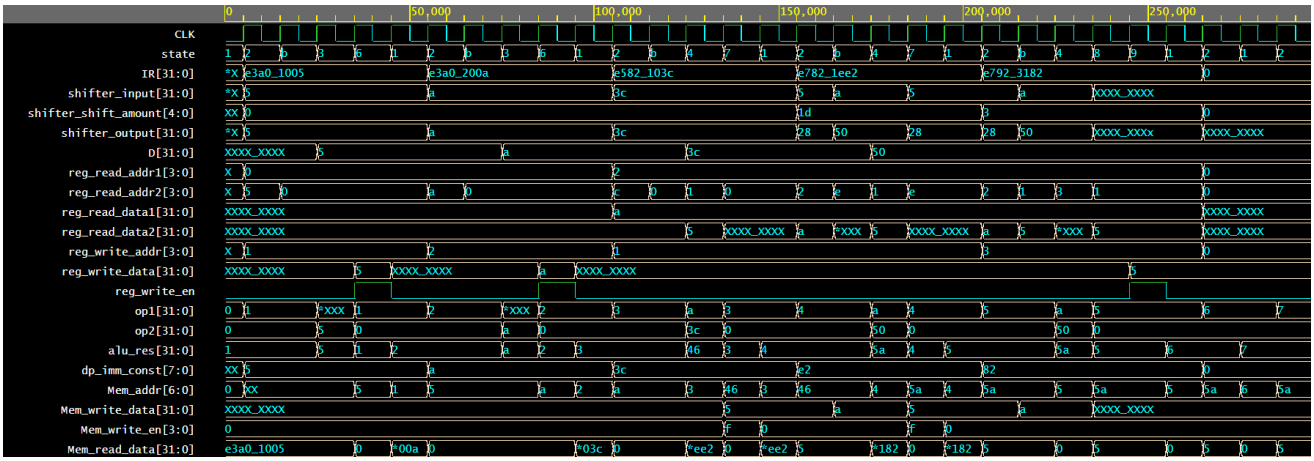
1.

```
    signal Mem_space : type_mem :=
                                (0 => X"E3A00E19",
                                 1 => X"E3A01003",
                                 2 => X"E1B02081",
                                 3 => X"E1B030A1",
                                 4 => X"E1B040C1",
                                 others => X"00000000");

    -- mov r0, #400
    -- mov r1, #3
    -- movs r2, r1, LSL #1
    -- movs r3, r1, LSR #1
    -- movs r4, r1, ASR #1
```



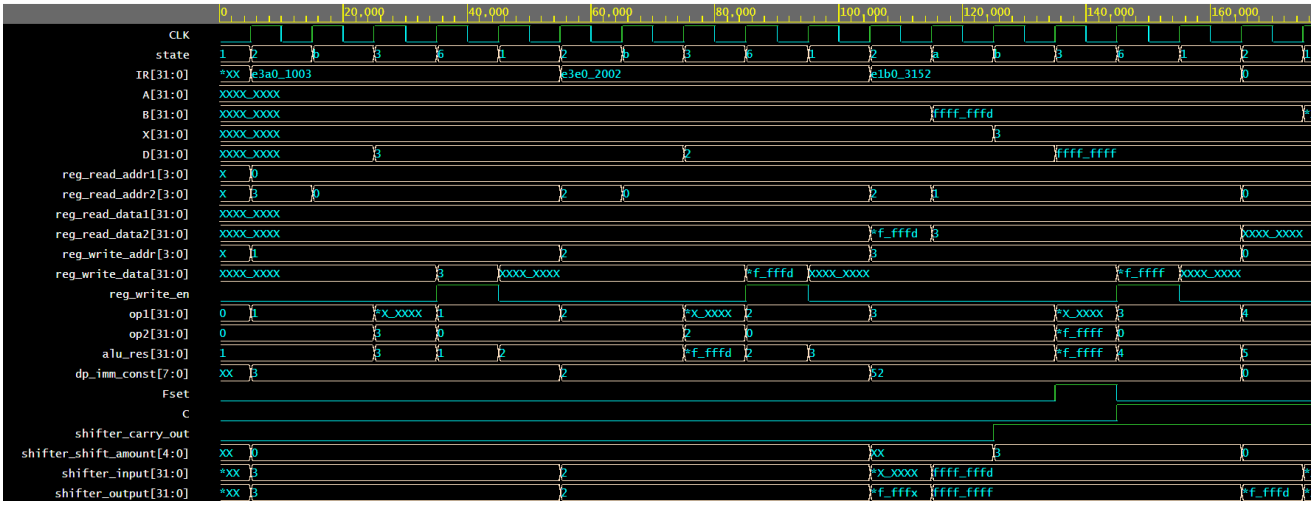
2.

```
    signal Mem_space : type_mem :=
                                (0 => X"E3A01005",
                                 1 => X"E3A0200A",
                                 2 => X"E582103C",
                                 3 => X"E7821EE2",
                                 4 => X"E7923182",
                                 others => X"00000000");

    -- mov r1, #5
    -- mov r2, #10
    -- str r1, [r2, #60] @store at 70
    -- str r1, [r2, r2, ROR #29] @store at 10 + 10 * 8 = 90
    -- ldr r3, [r2, r2, LSL #3] @load from 90
```



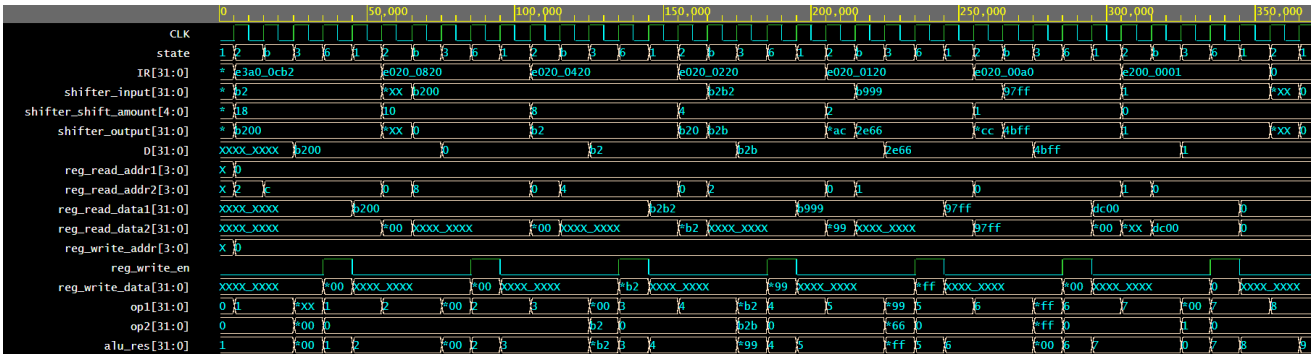
3.

```
signal Mem_space : type_mem :=
                                (0 => X"E3A01003",
                                 1 => X"E3E02002",
                                 2 => X"E1B03152",
                                 others => X"00000000");

--mov r1, #3
--mvn r2, #2
--movs r3, r2, ASR r1 @r3 should have all bits set and carry flag should be 1
```



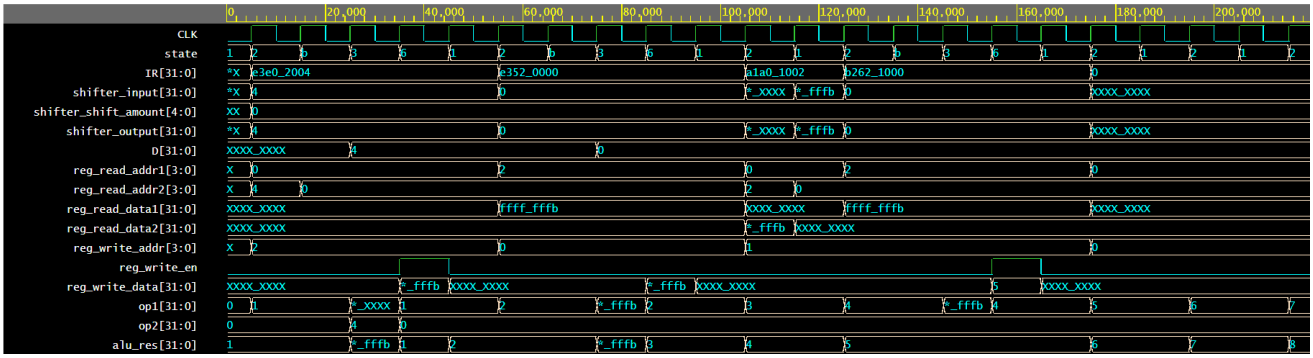
4.

```
signal Mem_space : type_mem :=
                                (0 => X"E3A00CB2",
                                 1 => X"E0200820",
                                 2 => X"E0200420",
                                 3 => X"E0200220",
                                 4 => X"E0200120",
                                 5 => X"E02000A0",
                                 6 => X"E2000001",
                                others => X"00000000");

-- @compute parity of a number
-- mov r0, #00131000 @even parity
-- eor r0, r0, r0, LSR #16
-- eor r0, r0, r0, LSR #8
-- eor r0, r0, r0, LSR #4
-- eor r0, r0, r0, LSR #2
-- eor r0, r0, r0, LSR #1
-- and r0, r0, #1
```



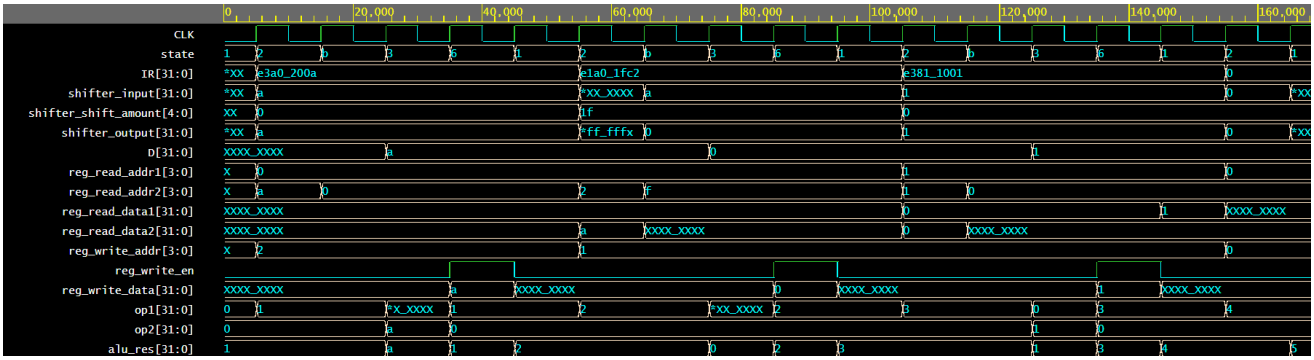
5.

```
signal Mem_space : type_mem :=
                                (0 => X"E3E02004",
                                 1 => X"E3520000",
                                 2 => X"A1A01002",
                                 3 => X"B2621000",
                                 others => X"00000000");

--@absolute of r2 in r1
--mov r2, #-5
--cmp r2, #0
--movge r1, r2
--rsblt r1, r2, #0
```



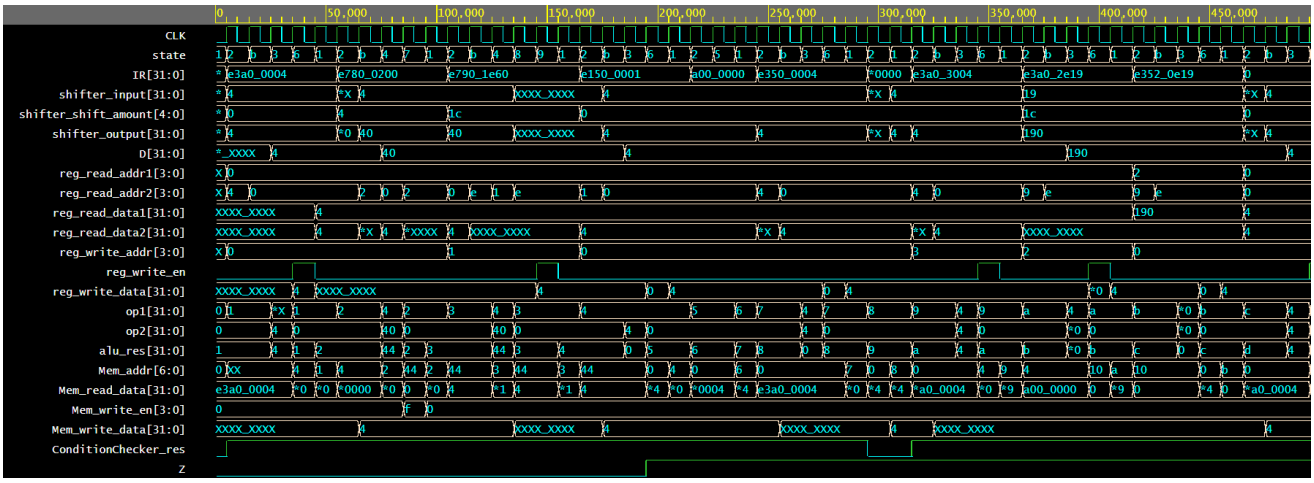
6.

```
signal Mem_space : type_mem :=
                                (0 => X"E3A0200A",
                                 1 => X"E1A01FC2",
                                 2 => X"E3811001",
                                 others => X"00000000");

-- @sign check of r2, r1 has +1 or -1
-- mov r2, #10
-- mov r1, r2, ASR #31
-- orr r1, r1, #1
```



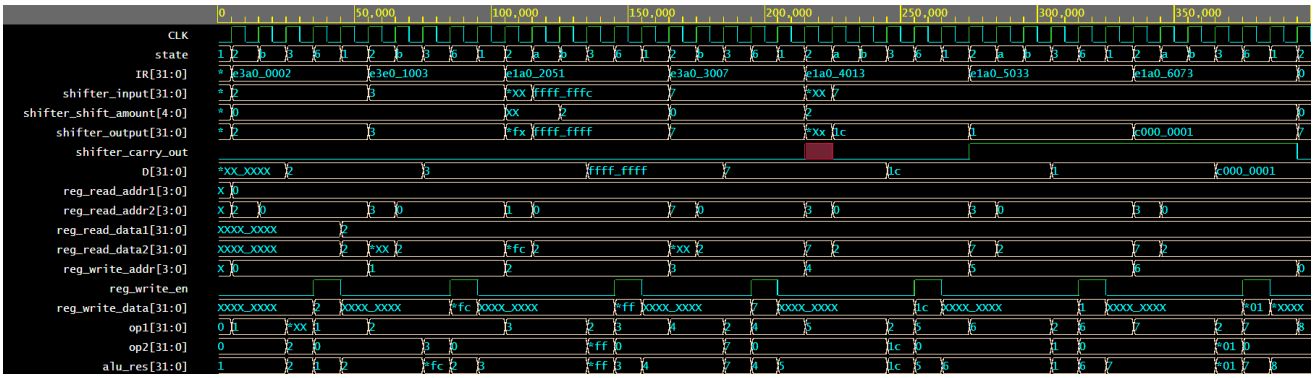
7.

```
    signal Mem_space : type_mem :=
                                ( 0 => X"E3A00004",
                                  1 => X"E7800200",
                                  2 => X"E7901E60",
                                  3 => X"E1500001",
                                  4 => X"0A000000",
                                  5 => X"E3A00004",
                                  6 => X"E3500004",
                                  7 => X"1A000000",
                                  8 => X"E3A03004",
                                  9 => X"E3A02E19",
                                  10 => X"E3520E19",
                                  others => X"00000000"
                                  );

    --      mov r0, #4
    --      str r0, [r0, r0, LSL #4] @store in 4 + 4*16 = 68
    --      ldr r1, [r0, r0, ROR #28] @load from 4 + 4*16 = 68
    --      cmp r0, r1
    --      beq A
    --      mov r0, #4
    -- A:   cmp r0, #4
    --      bne D
    --      mov r3, #4
    -- D:   mov r2, #0x190
    --      cmp r2, #0x190
```



8.

```
signal Mem_space : type_mem :=
                                ( 0 => X"E3A00002",
                                  1 => X"E3E01003",
                                  2 => X"E1A02051",
                                  3 => X"E3A03007",
                                  4 => X"E1A04013",
                                  5 => X"E1A05033",
                                  6 => X"E1A06073",
                                  others => X"00000000"
                                  );

-- mov r0, #2
-- mvn r1, #3
-- mov r2, r1, ASR r0 @r2 should contain 0xFFFFFFFF
-- mov r3, #7
-- mov r4, r3, LSL r0
-- mov r5, r3, LSR r0
-- mov r6, r3, ROR r0
```



2.) Resource Table (Synthesis)

Can see the resources used by the different module implementations on given FPGA.

For example, for shifter -

```
# Info: Device Utilization for 7A100TCSG324
# Info: ******************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ------------------------------------------------------------------
# Info: IOs                               73      210      34.76%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              183     63400     0.29%
# Info: CLB Slices                        24      15850     0.15%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: ------------------------------------------------------------------
# Info: ******************************************************************
# Info: Library: work    Cell: shifter    View: shifter_beh
# Info: ******************************************************************
# Info:  Number of ports :                          73
# Info:  Number of nets :                           325
# Info:  Number of instances :                      285
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of LUTs :                           183
# Info:  Number of Primitive LUTs :                 212
# Info:  Number of LUTs with LUTNM/HLUTNM :          58
# Info:  Number of accumulated instances :          285
# Info: ****************************
# Info:  IO Register Mapping Report
# Info: ****************************
```