Assignment 2
Computer Architecture

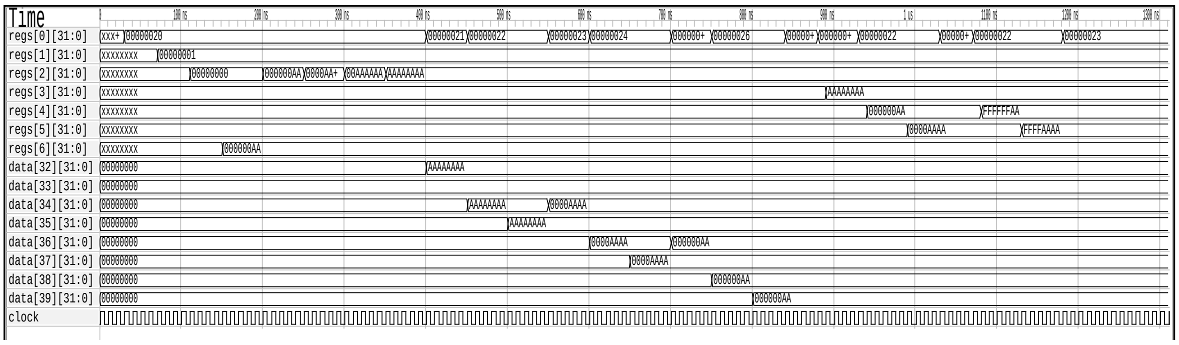
Divyansh Mittal , 2020CS10342
cs1200342@iitd.ac.in

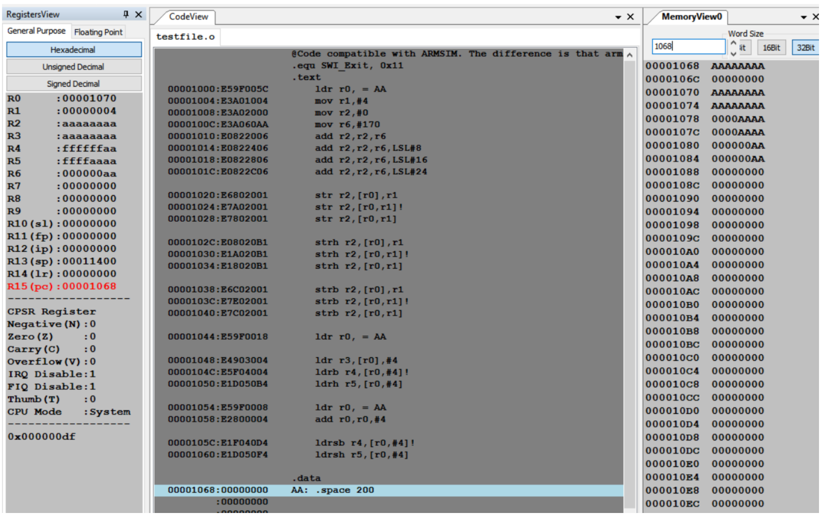
1. Stage 6

Mentor precision for synthesis and quartus for netlist and used EDA Playground to code. gtkWave for the waves output

a) Implementing shift and rotate

i. gtk Waves





```
R0       :00001070
R1       :00000004
R2       :aaaaaaaa
R3       :aaaaaaaa
R4       :ffffffaa
R5       :ffffaaaa
R6       :000000aa
R7       :00000000
R8       :00000000
R9       :00000000
R10(sl)  :00000000
R11(fp)  :00000000
R12(ip)  :00000000
R13(sp)  :00011400
R14(lr)  :00000000
R15(pc)  :00001068
-------------------
CPSR Register
Negative(N):0
Zero(Z)    :0
Carry(C)   :0
Overflow(V):0
IRQ Disable:1
FIQ Disable:1
Thumb(T)   :0
CPU Mode   :System
-------------------
0x000000df
```

```
                        @Code compatible with ARMSIM. The difference is that arm
                        .equ SWI_Exit, 0x11
                        .text
00001000:E59F005C          ldr r0, = AA
00001004:E3A01004          mov r1,#4
00001008:E3A02000          mov r2,#0
0000100C:E3A060AA          mov r6,#170
00001010:E0822006          add r2,r2,r6
00001014:E0822406          add r2,r2,r6,LSL#8
00001018:E0822806          add r2,r2,r6,LSL#16
0000101C:E0822C06          add r2,r2,r6,LSL#24

00001020:E6802001          str r2,[r0],r1
00001024:E7A02001          str r2,[r0,r1]!
00001028:E7802001          str r2,[r0,r1]

0000102C:E08020B1          strh r2,[r0],r1
00001030:E1A020B1          strh r2,[r0,r1]!
00001034:E18020B1          strh r2,[r0,r1]

00001038:E6C02001          strb r2,[r0],r1
0000103C:E7E02001          strb r2,[r0,r1]!
00001040:E7C02001          strb r2,[r0,r1]

00001044:E59F0018          ldr r0, = AA

00001048:E4903004          ldr r3,[r0],#4
0000104C:E5F04004          ldrb r4,[r0,#4]!
00001050:E1D050B4          ldrh r5,[r0,#4]

00001054:E59F0008          ldr r0, = AA
00001058:E2800004          add r0,r0,#4

0000105C:E1F040D4          ldrsb r4,[r0,#4]!
00001060:E1D050F4          ldrsh r5,[r0,#4]

                        .data
00001068:00000000       AA:  .space 200
         :00000000
```

```
1068
00001068  AAAAAAAA
0000106C  00000000
00001070  AAAAAAAA
00001074  AAAAAAAA
00001078  0000AAAA
0000107C  0000AAAA
00001080  000000AA
00001084  000000AA
00001088  00000000
0000108C  00000000
00001090  00000000
00001094  00000000
00001098  00000000
0000109C  00000000
000010A0  00000000
000010A4  00000000
000010A8  00000000
000010AC  00000000
000010B0  00000000
000010B4  00000000
000010B8  00000000
000010BC  00000000
000010C0  00000000
000010C4  00000000
000010C8  00000000
000010CC  00000000
000010D0  00000000
000010D4  00000000
000010D8  00000000
000010DC  00000000
000010E0  00000000
000010E4  00000000
000010E8  00000000
000010EC  00000000
```

Figure 1: gtkWave along with armsim output to verify working

```
        0 => X"E3A00020",
        1 => X"E3A01001",
        2 => X"E3A02000",
        3 => X"E3A060AA",
        4 => X"E0822006",
        5 => X"E0822406",
        6 => X"E0822806",
        7 => X"E0822C06",
        8 => X"E6802001",
        9 => X"E7A02001",
        10 => X"E7802001",
        11 => X"E08020B1",
        12 => X"E1A020B1",
        13 => X"E18020B1",
        14 => X"E6C02001",
        15 => X"E7E02001",
        16 => X"E7C02001",
        17 => X"E3A00020",
        18 => X"E4903001",
        19 => X"E5F04001",
        20 => X"E1D050B1",
        21 => X"E3A00021",
        22 => X"E1F040D1",
        23 => X"E1D050F1",
        others => X"00000000"
```

```
    mov r0,#32
    mov r1,#1
    mov r2,#0
    mov r6,#170
    add r2,r2,r6
    add r2,r2,r6,LSL#8
    add r2,r2,r6,LSL#16
    add r2,r2,r6,LSL#24

    str r2,[r0],r1
    str r2,[r0,r1]!
    str r2,[r0,r1]

    strh r2,[r0],r1
    strh r2,[r0,r1]!
    strh r2,[r0,r1]

    strb r2,[r0],r1
    strb r2,[r0,r1]!
    strb r2,[r0,r1]

    mov r0,#32

    ldr r3,[r0],#1
    ldrb r4,[r0,#1]!
    ldrh r5,[r0,#1]

    mov r0,#33

    ldrsb r4,[r0,#1]!
    ldrsh r5,[r0,#1]
```

b) How to run the code

Simply run the makefile by the command "make" to view the outputs in gtkwave.[1]

c) Basic working

To implement the new DT instructions, the instruction decoder has two more types. Namely DTHI and DTHR for Data transfer half word immediate and data transfer half word register. No new state in FSM is needed. Simply iorD mux is expanded. The output from A goes into this mux too. For post indexing, we use this output for address. Write back is also simple as after ReS has been calculated along with storing in data, we just need to store in reg file as well. Since the store address is different here, we need a mux for this to and FSM controls it too. Shift is also very different for these, as immediate has the bits split in 2 halves, and register load has no rotation specified. This is also taken care by appropriately setting ALU input. For up down, FSM sets ALU opcode accordingly. The write enable for the "D" register is 4 bits to take care of byte and halfword. Another signDT signal is used to do signed loading.

d) Sample test code explained

The sample test code basically checks all the instrucions. I am using locn 32 and beyond for storing purposes. In register r2, I store 0xAAAAAAAA [2]. Now I simply call all the types of str instructions. The output is attatched via gtkWave and verified via ARMSim. Then I call the ldr instructions variant, both signed and unsigned. There output is also verified via armsim. Since the 15^{th} and 7^{th} bit is 1, signed load makes the rest of bits 1 for halfword and byte load. [3]

e) Netlist

i. Log Output

```
[2022-03-12 13:58:00 UTC] vlib work && vcom '-2019' '-o' ALU.vhd Cond.vhd data_mem.vhd Flags.vhd FSM.
    vhd IDAB_Reg.vhd Instr.vhd MyTypes.vhd PC.vhd Register.vhd Rotator.vhd Shifter.vhd design.vhd
    testbench.vhd && vsim -c -do "vsim TB; vcd file dump.vcd; vcd add -r sim:/*; run -all; exit"
VSIMSA: Configuration file changed: '/home/runner/library.cfg'
ALIB: Library "work" attached.
work = /home/runner/work/work.lib
Aldec, Inc. VHDL Compiler, build 2020.04.130
VLM Initialized with path: "/home/runner/library.cfg".
```

[1]For this we do need gtkWave, which can easily be installed on a linux system with the command "sudo apt install gtkwave"
[2]This in binary is "10101010101010101010101010101010" and thus easy to check all instructions
[3]View the testfile.s in submission files to get the ARMSim runnable code.

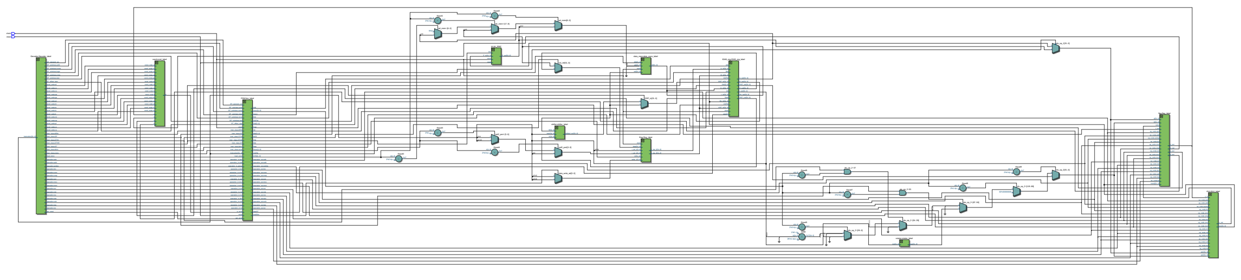


Figure 2: Net List

```
DAGGEN WARNING DAGGEN_0523: "The source is compiled without the -dbg switch. Line breakpoints and
    assertion debug will not be available."
COMP96 File: ALU.vhd
COMP96 Compile Entity "ALU"
COMP96 Compile Architecture "alu_arch" of Entity "ALU"
COMP96 File: Cond.vhd
COMP96 Compile Entity "cond"
COMP96 Compile Architecture "arch" of Entity "cond"
COMP96 File: data_mem.vhd
COMP96 Compile Entity "data_mem"
COMP96 Compile Architecture "arch" of Entity "data_mem"
COMP96 File: Flags.vhd
COMP96 Compile Entity "flag"
COMP96 Compile Architecture "arch" of Entity "flag"
COMP96 File: FSM.vhd
COMP96 Compile Entity "FSM"
COMP96 Compile Architecture "behaviour" of Entity "FSM"
COMP96 File: IDAB_Reg.vhd
COMP96 Compile Entity "IDAB_reg"
COMP96 Compile Architecture "IDAB_reg_arch" of Entity "IDAB_reg"
COMP96 File: Instr.vhd
COMP96 Compile Entity "Decoder"
COMP96 Compile Architecture "Behavioral" of Entity "Decoder"
COMP96 File: MyTypes.vhd
COMP96 Compile Package "MyTypes"
COMP96 Compile Package Body "MyTypes"
COMP96 File: PC.vhd
COMP96 Compile Entity "pc"
COMP96 Compile Architecture "arch" of Entity "pc"
COMP96 File: Register.vhd
COMP96 Compile Entity "Reg"
COMP96 Compile Architecture "reg_arch" of Entity "Reg"
COMP96 File: Rotator.vhd
COMP96 Compile Entity "rotator"
COMP96 Compile Architecture "arch" of Entity "rotator"
COMP96 File: Shifter.vhd
COMP96 Compile Entity "shifter"
COMP96 Compile Architecture "arch" of Entity "shifter"
COMP96 File: design.vhd
COMP96 Compile Entity "processor"
COMP96 Compile Architecture "arch" of Entity "processor"
COMP96 File: testbench.vhd
```

```
COMP96 Compile Entity "TB"
COMP96 Compile Architecture "behavior" of Entity "TB"
COMP96 Incorrect order of units detected.
COMP96 Automatic reorder and incremental recompilation of required units in progress.
COMP96 File: ALU.vhd
COMP96 Compile Entity "ALU"
COMP96 Compile Architecture "alu_arch" of Entity "ALU"
COMP96 File: Cond.vhd
COMP96 Compile Entity "cond"
COMP96 Compile Architecture "arch" of Entity "cond"
COMP96 File: FSM.vhd
COMP96 Compile Entity "FSM"
COMP96 Compile Architecture "behaviour" of Entity "FSM"
COMP96 File: Instr.vhd
COMP96 Compile Entity "Decoder"
COMP96 Compile Architecture "Behavioral" of Entity "Decoder"
COMP96 File: Flags.vhd
COMP96 Compile Entity "flag"
COMP96 Compile Architecture "arch" of Entity "flag"
COMP96 File: design.vhd
COMP96 Compile Architecture "arch" of Entity "processor"
COMP96 Top-level unit(s) detected:
COMP96 Entity => TB
COMP96 Compile success 0 Errors 0 Warnings Analysis time : 0.1 [s]
dmesg: read kernel buffer failed: Operation not permitted
dmesg: read kernel buffer failed: Operation not permitted
# Aldec, Inc. Riviera-PRO version 2020.04.130.7729 built for Linux64 on June 10, 2020.
# HDL, SystemC, and Assertions simulator, debugger, and design environment.
# (c) 1999-2020 Aldec, Inc. All rights reserved.
# ELBREAD: Elaboration process.
# ELBREAD: Elaboration time 0.0 [s].
# KERNEL: Main thread initiated.
# KERNEL: Kernel process initialization phase.
# ELAB2: Elaboration final pass...
# ELAB2: Create instances ...
# KERNEL: Time resolution set to 1ps.
# ELAB2: Create instances complete.
# SLP: Started
# SLP: Elaboration phase ...
# SLP: Elaboration phase ... skipped, nothing to simulate in SLP mode : 0.0 [s]
# SLP: Finished : 0.0 [s]
# ELAB2: You do not have a license to run VHDL performance optimized simulation. Contact Aldec for
    ordering information - sales@aldec.com.
# ELAB2: Elaboration final pass complete - time: 0.0 [s].
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance of simulation is reduced
    .
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 8080 kB (elbread=427 elab2=7508 kernel=144 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library '/usr/share/Riviera-PRO/bin/libsystf.so'
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/Decoder_label, Process: line__34.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/Decoder_label, Process: line__48.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/data_mem_label, Process: line__34.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/Reg_label, Process: line__23.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/Reg_label, Process: line__24.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/rotator_label, Process: line__15.
# KERNEL: WARNING: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
# KERNEL: Time: 0 ps, Iteration: 0, Instance: /TB/uut/shifter_label, Process: shift_by_set.
# KERNEL: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2022-03-12 13:58:03 UTC] Opening EPWave...
Done
```