Anthony De La Rosa 77187664
Nikita Tsvetkov 53373524
Viet Ly 92029965
Kim Truong 85614378

CSE 132L Lab 1 Report

Design Architecture

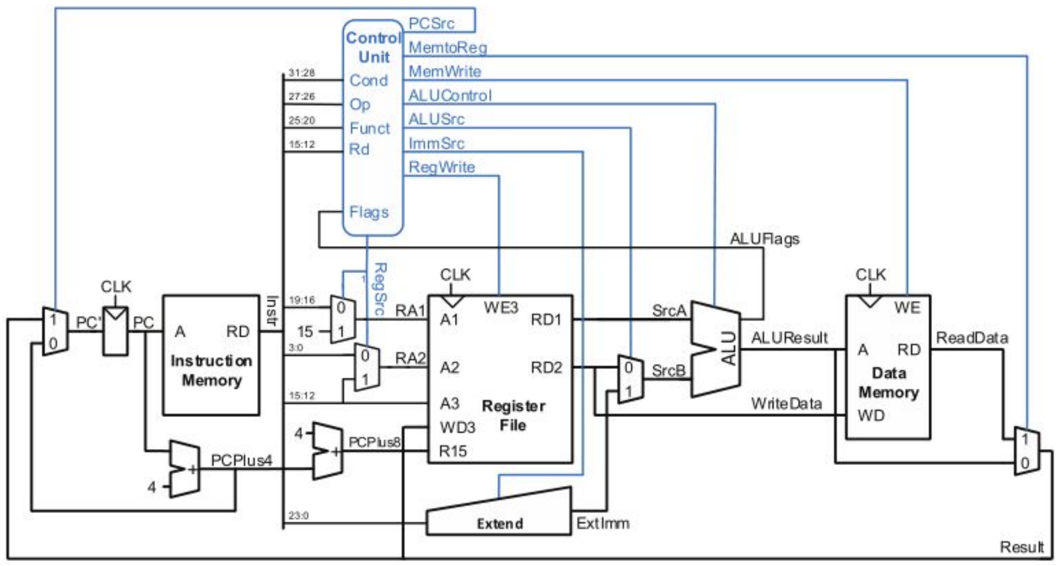


Figure 1: Simplified Single-cycle ARM processor architecture

The architecture was designed according to the specification in Chapter 7 of Digital Design and Computer Architecture, like the figure above. Our group first started by creating basic components such as a flip flop or multiplexer, using the code provided in Digital Design and Computer Architecture. These basic components were then used to create and link the data path between our ALU, adder, decoder, extender, and etc. For a complete list of the components designed, please refer to the `./design/` folder attached with this report. The `arm` module connects both the control and data path together. The `top` module connects `arm` with the `dmem`, data memory, and the `imem`, instruction memory.

Testbench Architecture

We have testbenches for:

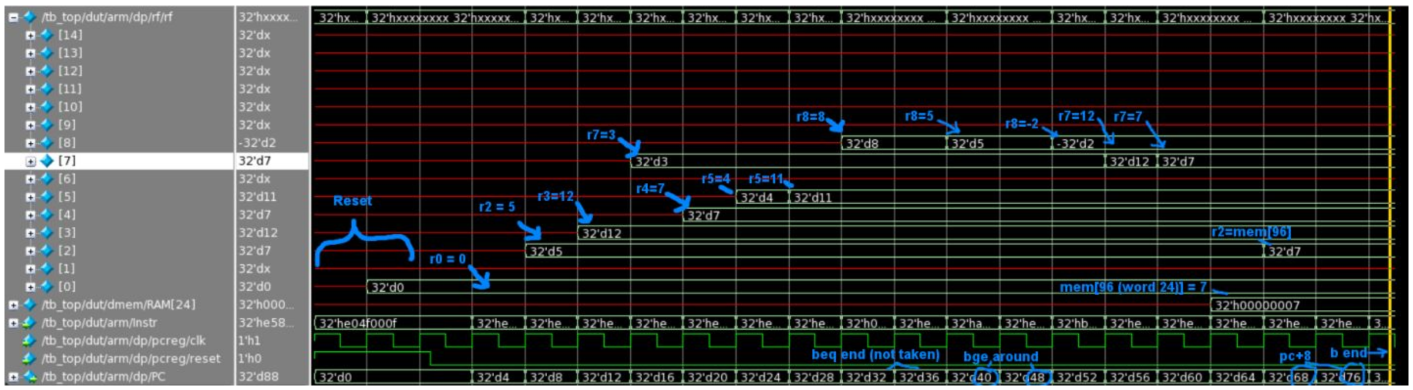
Top Module/Processor: We generate a clock and pass a value (#7) as write data and check if that value gets passed to the correct address

ALU module: Our group test basic operations - add for example, with 2 random values and check that it returns correct values, overflow, etc.

Testing the Processor

Our design is tested with the testbench that is included in the chapter 7 notes. The benchmark code has been pasted below for reference. These instructions are embedded inside of the imem verilog code. The annotated waveform is shown below. The blue text refers to the instruction/comments of the instruction that is being ran, and then the blue lines point to the data which shows that the blue instruction had a valid effect on the processor. We know that the add,sub,and,orr,str,ldr and branch instructions work, because these instructions are included in the testbench, and their sequential execution leads to valid results. We also see the "Simulation succeeded" message at the end of the simulation. The code has been pasted below for reference.

```
MAIN    SUB R0, R15, R15    ; R0 = 0
        ADD R2, R0, #5      ; R2 = 5
        ADD R3, R0, #12     ; R3 = 12
        SUB R7, R3, #9      ; R7 = 3
        ORR R4, R7, R2      ; R4 = 3 OR 5 = 7
        AND R5, R3, R4      ; R5 = 12 AND 7 = 4
        ADD R5, R5, R4      ; R5 = 4 + 7 = 11
        SUBS R8, R5, R7     ; R8 = 11 - 3 = 8, set Flags
        BEQ END             ; shouldn't be taken
        SUBS R8, R3, R4     ; R8 = 12 - 7  = 5
        BGE AROUND          ; should be taken
        ADD R5, R0, #0      ; should be ski pped
AROUND  SUBS R8, R7, R2     ; R8 = 3 - 5 = -2, set Flags
        ADDLT R7, R5, #1    ; R7 = 11 + 1 = 12
        SUB R7, R7, R2      ; R7 = 12 - 5 = 7
        STR R7, [R3, #84]   ; mem[12+84] = 7
        LDR R2, [R0, #96]   ; R2 = mem[96] = 7
        ADD R15, R15, R0    ; PC = PC+8 (skips next)
        ADD R2, R0, #14     ; shouldn't happen
        B END               ; always taken
        ADD R2, R0, #13     ; shouldn't happen
        ADD R2, R0, #10     ; shouldn't happen
END     STR R2, [R0, #100]  ; mem[100] = 7
```


/tb_top/dut/arm/dp/rf/rf 32'hxxxx...
[14] 32'dx
[13] 32'dx
[12] 32'dx
[11] 32'dx
[10] 32'dx
[9] 32'dx
[8] -32'd2
[7] 32'd7
[6] 32'dx
[5] 32'd11
[4] 32'd7
[3] 32'd12
[2] 32'd7
[1] 32'dx
[0] 32'd0
/tb_top/dut/dmem/RAM[24] 32'h000...
/tb_top/dut/arm/Instr 32'he58...
/tb_top/dut/arm/dp/pcreg/clk 1'h1
/tb_top/dut/arm/dp/pcreg/reset 1'h0
/tb_top/dut/arm/dp/PC 32'd88
Reset
r0 = 0
r2 = 5
r3=12
r4=7
r7=3
r5=4
r5=11
r8=8
r8=5
r8=-2
r7=12
r7=7
r2=mem[96]
mem[96 (word 24)] = 7
32'h00000007
32'he04f000f
beq end (not taken)
bge around
pc+8
b end
32'd0 32'd4 32'd8 32'd12 32'd16 32'd20 32'd24 32'd28 32'd32 32'd36 32'd40 32'd48 32'd52 32'd56 32'd60 32'd64 32'd68 32'd76

Bugs/Issues encountered

When writing and synthesizing the modules, multiple issues were present. There was initial confusion about what we had to implement versus what we could take and use from Chapter 7. The difference between sequential and combinational logic had at first prevented us from synthesizing our code, but once that was remedied there no major issues. Minor problems were typical, run-of-the-mill errors such as missing spaces, colons, or mislabeled variables.

Tasks and contributions

Anthony De La Rosa: Formatting and contributing to the lab report, setting up the bitbucket repository for our code allowing everyone to contribute to the project, added in different component files into the repository.

Nikita:Waveform annotation, contributing elements, shell script configuration,bug fixing.

Viet: Design, test and debug ALU file, and also debug several testbench files

Kim: Design circuits elements, Responsible for writing the report