Lab5 Final Lab Description

Jiahao Chen tobychen@bu.edu
Xinran Zhang zhangxr@bu.edu

In this final lab, we modified the components we created in Lab5 milestone and managed to complete the full single cycle datapath. It has the ability to complete add, and, or, cbz, sub, movk, stur, ldur and b(unconditional branch) operation.

program_counter: We changed the offset and PC mux we used in first milestone. Now we use Uncondbranch flag and Branch flag generated by Control Unit and the Zero flag generated by ALU to replace the PC mux we used before. Also we directly use instruction[20:0] or instruction[20:5] and extend to 64 bit, and regard this as offset we used in first milestone. In order to be able to start PC from 0, we use PCmid to calculate the value of next PC at posedge clk and give this value to PC at negedge clk.

instruction_emory: We didn’ t change much in instruction memory. The only place we changed is the section where we fill the instruction memory. Now we have these instructions shown below. The idea is that we can use these instructions to test every operation in order to find out every single mistake that we may make.

```
memory_file[0] = 32'b11110010100_0000000000000001_00010; //movk x2, 1
memory_file[1] = 32'b11110010100_0000000000000111_00001; //moxk x1, 7
memory_file[2] = 32'b10001011000_00001_000000_00010_01010; //add x10, x2, x1
memory_file[3] = 32'b10001010000_01010_000000_00001_00011; //and x3, x1, x10
memory_file[4] = 32'b10001011000_00010_000000_00001_00100; //add x4, x1, x2
memory_file[5] = 32'b10101010000_00010_000000_00001_00101; //orr x5, x1, x2
memory_file[6] = 32'b11001011000_00010_000000_00001_00110; //sub x6, x1, x2
memory_file[7] = 32'b11111000000_00000000011_00010_00001; //stur x1, [x2, 3]
memory_file[8] = 32'b11111000010_00000000011_00010_00111; //ldur x7, [x2, 3]
memory_file[9] = 32'b00010100000_000000000000000000000010; //b 2
memory_file[11] = 32'b11110010100_0000000000000000_01000; //movk x8, 0
memory_file[12] = 32'b101101000000_000000000000011_00111; //cbz x7, 3
memory_file[13] = 32'b101101000000_000000000000010_01000; //cbz x8, 2
memory_file[15] = 32'b11110010100_0000000000000000_01000; //movk x8, 0
```

control_unit: Our control unit followed the idea on the datapath ppt that our professor gave us. This component get the instruction and output a variety of control unit based on the instruction opcode, which is the first 11 bits.

reg2_mux: In this component, we use the different part of instruction as the name of registers. However, our component is kind of unique because we assign all of the three names of registers in the same time. We do this because we do not want one more block to do the other two assign operation.

register_file: We basically didn’ t change anything in register file.

alu_mux: This mux is the same as the one on ppt. It decides whether the output is the signed extension of instruction[20:10] or the data read from Register[ReadSelect2]. The instruction[20:10] is usually an immediate or offset from the base register. But one thing that is different is that we integrated the sign extend in this operation.

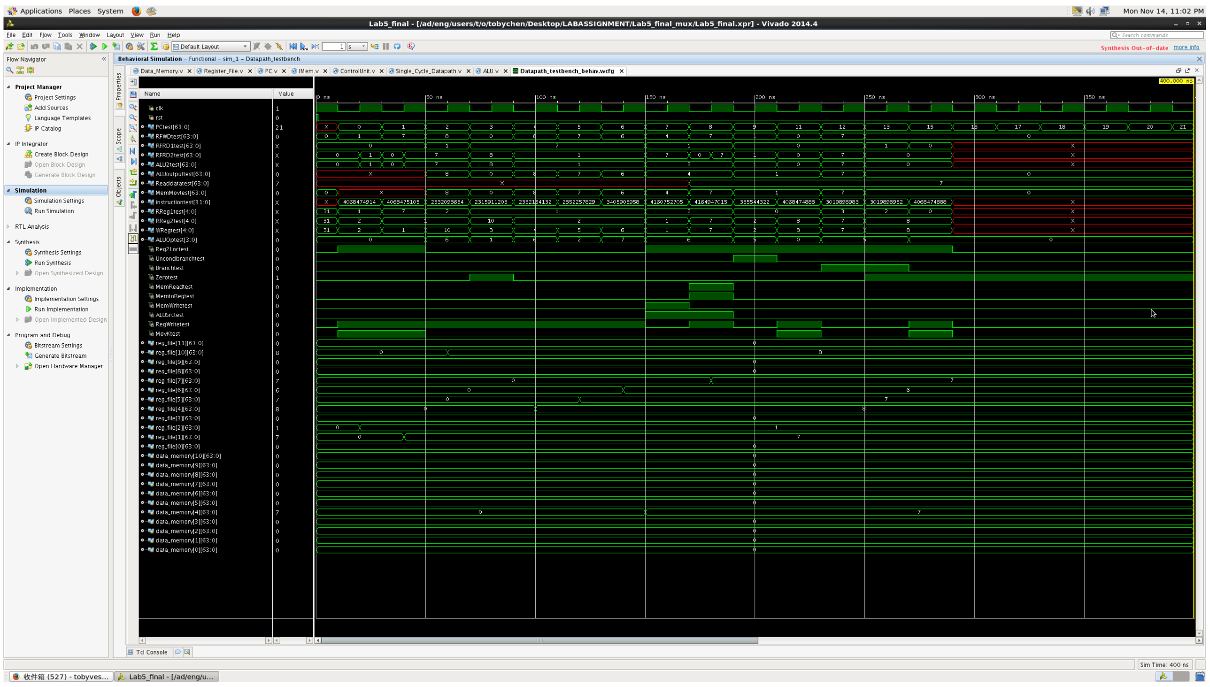
alu_unit: We didn’ t change much in this section. One place we changed is that we used a different always block to determine the Zero flag.

data_memory: We didn’ t change anything in data memory. It’ s the same as the last one.

data_mux: This mux is the same as the one on ppt. It decides whether the output is ALU’ s output or the data read from data memory.

movk: We used a very simple way to complete movk operation. Since it loads a 16 bits number and keep the rest of the bits in Register, we simply used concave to do this. The first 48 bits of the value read from Register[ReadSelect2] concaved with the 16 bits number given from instruction. We then write this number to Register[WriteSelect] and in this case, WriteSelect=ReadSelect2. This is the result of the movk operation. However, since we use the Register[ReadSelect2] and write to Register[WriteSelect], we need a mux to determine if we should use the output from the data_mux or the movk. And we added this mux in this section using if-else if.

The waveform that indicates our datapath successfully operated all the instruction shown above is here:


Behavioral Simulation - Functional - sim_1 - Datapath_testbench

Name	Value
clk	1
rst	0
PCtest[63:0]	21
RFWDtest[63:0]	0
RFRD1test[63:0]	X
RFRD2test[63:0]	X
ALU2test[63:0]	X
ALUoutputtest[63:0]	0
Readdatatest[63:0]	7
MemMovtest[63:0]	0
instructiontest[31:0]	X
RReg1test[4:0]	X
RReg2test[4:0]	X
WRegtest[4:0]	X
ALUOptest[3:0]	0
Reg2Loctest	0
Uncondbranchtest	0
Branchtest	0
Zerotest	1
MemReadtest	0
MemtoRegtest	0
MemWritetest	0
ALUSrctest	0
RegWritetest	0
MovKtest	0
reg_file[11][63:0]	0
reg_file[10][63:0]	8
reg_file[9][63:0]	0
reg_file[8][63:0]	0
reg_file[7][63:0]	7
reg_file[6][63:0]	6
reg_file[5][63:0]	7
reg_file[4][63:0]	8
reg_file[3][63:0]	0
reg_file[2][63:0]	1
reg_file[1][63:0]	7
reg_file[0][63:0]	0
data_memory[10][63:0]	0
data_memory[9][63:0]	0
data_memory[8][63:0]	0
data_memory[7][63:0]	0
data_memory[6][63:0]	0
data_memory[5][63:0]	0
data_memory[4][63:0]	7
data_memory[3][63:0]	0
data_memory[2][63:0]	0
data_memory[1][63:0]	0
data_memory[0][63:0]	0