EECS 112L/CSE 132L

Assignment 1 Deliverable

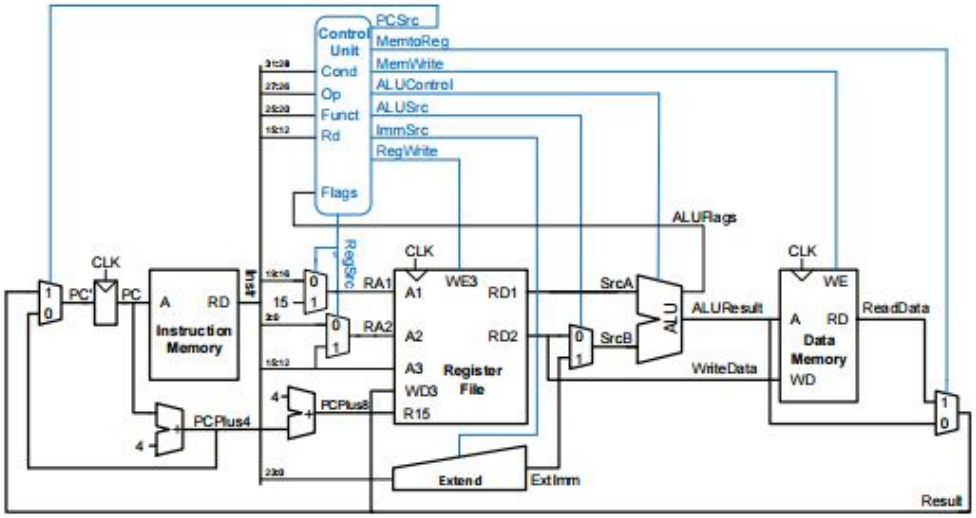
Single-cycle ARM Datapath and Control

Group ID: Kikei

Submission Date: 2/5/2017

Prepared by:
Jeffrey Han, 51999492
Aaron Liao, 90811748
Bryan Ou, 11074596
Kelvin Phan, 51197373

Block Diagram of Processor Design:



Regarding the overall architecture of our processor, the physical components consist of the Data Memory, Instruction Memory, ALU, Adder, Register File, Mux, Flip-Flop, and a Sign Extender. Other design components we have include the Top-level, Conditional Logic, Control Unit, Datapath, and the Decoder. All components but the ALU were provided to us in the textbook. In our Single-cycle Processor, each instruction is read from the Instruction Memory, followed by the Control Unit which decides what the given instruction is designed to do. For register operations, data is obtained from the Register File and processed, and then written back to the Register File. As for memory operations, the memory address is computed, the data is read from the memory, and then it is written back to the Register File. When processing branch operations, the branch target address is calculated using the Sign Extender.

As for the Control Unit on the processor, the decoder takes the Op, Funct, and Rd segments of the incoming bit string, while the Conditional Logic unit takes the Cond and also receives the ALUFlags from the ALU. The decoder sends MemtoReg, ALUSrc, $ImmSrc_{1:0,}RegSrc_{1:0}$, and $ALUControl_{1:0}$ directly to their respective parts on the datapath. $FlagW_{1:0}$, PCS, RegW, and MemW are sent from the decoder to the Conditional Logic unit. The Conditional Logic unit outputs PCSrc, RegWrite, and MemWrite which control the state of the PC, register file or memory. This unit checks if an instruction should execute, and if not it writes its outputs to 0.

Simulation Results:

ADDR	PROGRAM	; COMMENTS	BINARY MACHINE CODE	HEX CODE
00 MAIN	SUB R0, R15, R15	; R0 = 0	1110 000 0010 0 1111 0000 0000 0000 1111	E04F000F
04	ADD R2, R0, #5	; R2 = 5	1110 001 0100 0 0000 0010 0000 0000 0101	E2802005
08	ADD R3, R0, #12	; R3 = 12	1110 001 0100 0 0000 0011 0000 0000 1100	E280300C
0C	SUB R7, R3, #9	; R7 = 3	1110 001 0010 0 0011 0111 0000 0000 1001	E2437009
10	ORR R4, R7, R2	; R4 = 3 OR 5 = 7	1110 000 1100 0 0111 0100 0000 0000 0010	E1874002
14	AND R5, R3, R4	; R5 = 12 AND 7 = 4	1110 000 0000 0 0011 0101 0000 0000 0100	E0035004
18	ADD R5, R5, R4	; R5 = 4 + 7 = 11	1110 000 0100 0 0101 0101 0000 0000 0100	E0855004
1C	SUBS R8, R5, R7	; R8 = 11 - 3 = 8, set Flags	1110 000 0010 1 0101 1000 0000 0000 0111	E0558007
20	BEQ END	; shouldn't be taken	0000 1010 0000 0000 0000 0000 0000 1100	0A00000C
24	SUBS R8, R3, R4	; R8 = 12 - 7 = 5	1110 000 0010 1 0011 1000 0000 0000 0100	E0538004
28	BGE AROUND	; should be taken	1010 1010 0000 0000 0000 0000 0000 0000	AA000000
2C	ADD R5, R0, #0	; should be skipped	1110 001 0100 0 0000 0101 0000 0000 0000	E2805000
30 AROUND	SUBS R8, R7, R2	; R8 = 3 - 5 = -2, set Flags	1110 000 0010 1 0111 1000 0000 0000 0010	E0578002
34	ADDLT R7, R5, #1	; R7 = 11 + 1 = 12	1011 001 0100 0 0101 0111 0000 0000 0001	B2857001
38	SUB R7, R7, R2	; R7 = 12 - 5 = 7	1110 000 0010 0 0111 0111 0000 0000 0010	E0477002
3C	STR R7, [R3, #84]	; mem[12+84] = 7	1110 010 1100 0 0011 0111 0000 0101 0100	E5837054
40	LDR R2, [R0, #96]	; R2 = mem[96] = 7	1110 010 1100 1 0000 0010 0000 0110 0000	E5902060
44	ADD R15, R15, R0	; PC = PC+8 (skips next)	1110 000 0100 0 1111 1111 0000 0000 0000	E08FF000
48	ADD R2, R0, #14	; shouldn't happen	1110 001 0100 0 0000 0010 0000 0000 0001	E280200E
4C	B END	; always taken	1110 1010 0000 0000 0000 0000 0000 0001	EA000001
50	ADD R2, R0, #13	; shouldn't happen	1110 001 0100 0 0000 0010 0000 0000 0001	E280200D
54	ADD R2, R0, #10	; shouldn't happen	1110 001 0100 0 0000 0010 0000 0000 0001	E280200A
58 END	STR R2, [R0, #100]	; mem[100] = 7	1110 010 1100 0 0000 0010 0000 0101 0100	E5802064

Figure 7.60 Assembly and machine code for test program

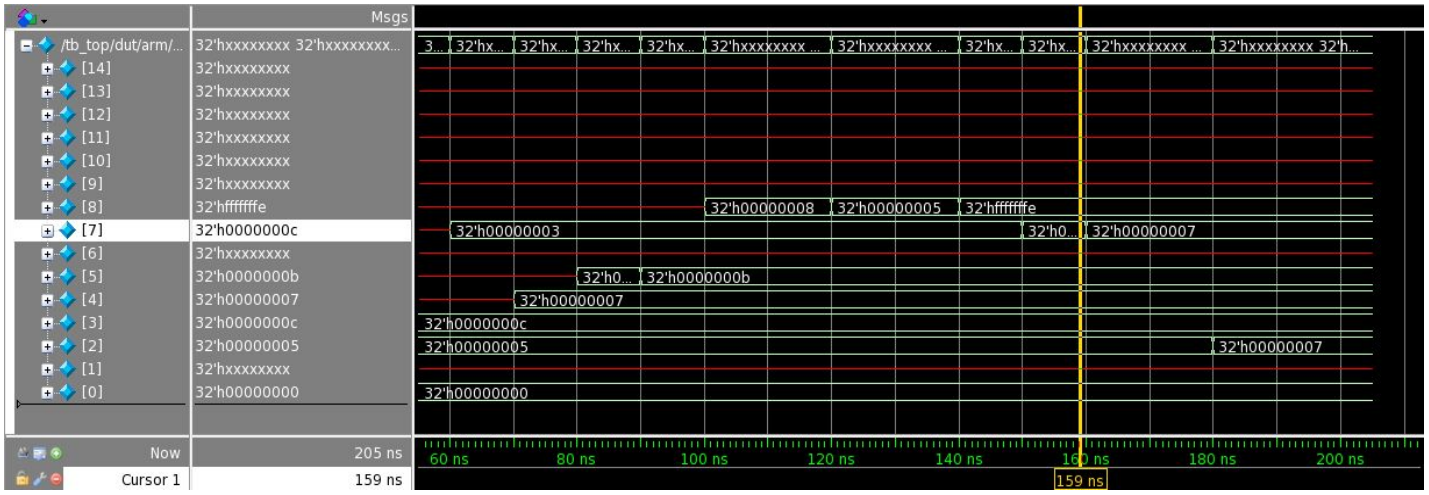


***Figure 1*: Register File waveform, (0-150 ns)**

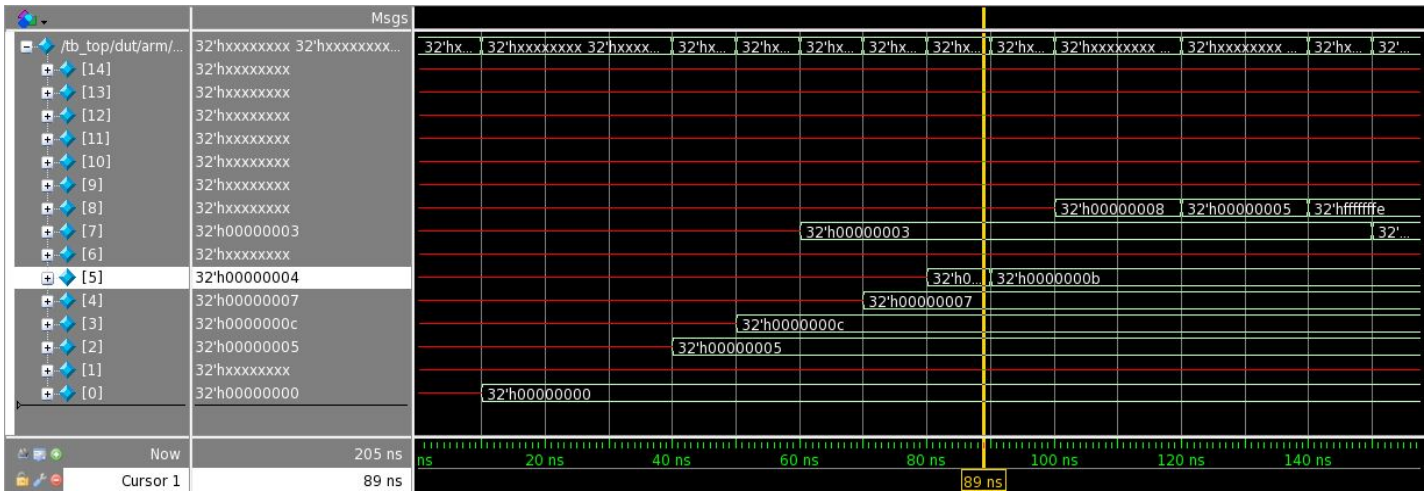


***Figure 2*: Register File waveform, (60-200 ns)**

Simulation Analysis:

- We can verify the functionality of of our processors design by comparing the waveforms we acquired with the given testbench values (Figure 7.60).
- From the testbench, R0 = 0, R2 = 5, R3 = 12, R7 = 3, R4 = 7, and R5 = 4 for initial values.
- Chronologically:
    - R5 changes to 11, which can be seen in Figure 1 at 100ns.
    - R8 is then set to 8, and since R8 changes to 5 at 120ns, it is apparent that the BEQ instruction was skipped.
    - The next branch to the “around” label is taken, as the value in R5 does not change.
    - In Figure 2, R7 changes to 12, then subsequently 7 at 150ns.
    - To verify LDR and STR, mem[96] is set to 7, and the value in R2 is then set equal to mem[96], resulting in R2 = 7.
    - R2 remains unchanged through the rest of the waveform, as the branch instruction to the “end” label is taken, skipping further ADD instructions.
- End register value results:
    - R0 = 0
    - R2 = 7
    - R3 = C = 12
    - R4 = 7
    - R5 = B = 11
    - R7 = 7
    - R8 = -2
- We were unable to detect any visible bugs in our design and simulation. All of the included components in our ALU compiled successfully without any errors or warnings, and the resulting values in our registers matched the expected values given in the testbench.

Designated Tasks/Contributions:

- **Jeff Han** - Component design implementation, debugging, assignment report.
- **Aaron Liao** - Overall ALU synthesis, core debugging and troubleshooting.
- **Bryan Ou** - Component design implementation, debugging, assignment report.
- **Kelvin Phan** - Project optimization, Bash script for compilation, debugging.