Digital Design and Computer Architecture

Lab 10: Multicycle ARM Processor (Part 1)

Introduction

In this lab and the next, you will design and build your own multicycle ARM processor. You will be much more on your own to complete these labs than you have been in the past, but you may reuse any of your hardware (SystemVerilog modules) from previous labs.

Your multicycle processor should match the design from the text, which is reprinted in Figure 1 (at the end of the lab) for your convenience. It should handle the following instructions: `ADD`, `SUB`, `AND`, and `ORR` (with register and immediate operands, but no shifts), `LDR` and `STR` (with positive immediate offset), and `B`.

The multicycle processor is divided into three units: the `controller`, `datapath`, and `mem` (memory) units. Note that the `mem` unit contains the shared memory used to hold both data and instructions. Also note that the `controller` unit comprises both the `Decode` and `Conditional Logic` units. We've repeated the control unit diagram in Figure 2 (at the end of the lab) for your convenience.

In this lab you will design and test the `controller`.

Overall Design

Now you will begin the hardware implementation of your multicycle ARM processor. First, copy the provided `arm_multi.sv` to your own directory and rename it `arm_multi_xx.sv`.

The `arm` module instantiates both the `datapath` and the control unit (called the `controller` module). You will design the `controller` module (and all of its submodules) in this lab. In the next lab, you will design the `datapath`. The memory is essentially identical to the data memory from Lab 9 and will be provided for you.

Control Unit Design

The control unit (`controller`) is the most complex part of the multicycle processor. It consists of two modules: Decode (`decode`) and Conditional Logic (`condlogic`). `decode` instantiates the Main FSM (`mainfsm`) and includes logic for the ALU Decoder, PC Logic, and Instruction Decoder. On reset, the Main FSM should start at

© 2014 David Money Harris and Sarah L. Harris

State 0 (DECODE). The state transition diagram is given in Figure 3 at the end of this handout.

The controller, decode, condlogic, and mainfsm headers are given in arm_multi.sv showing the inputs and outputs for each module. A portion of the SystemVerilog code for the control units has been given to you. Complete the SystemVerilog code to completely design the hardware of the controller and its submodules. Remember that you can reuse code from the single-cycle processor of Lab 9. Please see Figure 2 at the end of this handout for a change to the Condition Logic module.

Generating Control Signals

Before you begin developing the hardware for your ARM multicycle processor, you'll need to determine the correct control signals for each state in the multicycle processor's state transition diagram. This state transition diagram is shown in Figure 7.41 in the book and in Figure 3 in these instructions. Complete the output table of the Main FSM in Table 1 at the end of this handout. Give the FSM control word in hexadecimal for each state. The first two rows are filled in as examples. Be careful with this step. It takes much longer to debug an erroneous circuit than to design it correctly the first time.

Testing

Create a controllertest_xx.sv testbench for the controller module. Test each of the instructions that the processor should support: ADD, SUB, AND, and ORR (with register and immediate operands, but no shifts), LDR and STR (with positive immediate offset), and B. Be sure to test both taken and nontaken branches. From Figure 2, the controller inputs are: CLK, reset, $Cond_{3:0}$, $Op_{1:0}$, $Funct_{5:0}$, $Rd_{3:0}$, and $ALUFlags_{3:0}$. The SystemVerilog header for controller lists clk, reset, Instr[31:12], and ALUFlags[3:0] as inputs. Recall from the machine code formats that Instr[31:12] includes the Cond, Op, Funct, and Rd fields (as well as Rn, which is not used).

Your test bench should apply the inputs to controller (clk, reset, Instr[31:12], and ALUFlags[3:0]). Visually inspect the states and outputs to verify that they match your expectations from Table 1. If you find any errors, debug your circuit and correct the errors. Save a copy of your waveforms showing the inputs, state, and control outputs at each state.

© 2014 David Money Harris and Sarah L. Harris

What to Turn In

Submit the following elements **in the following order**. Clearly label each part by number. Poorly organized submissions will lose points.

1. **Please indicate how many hours you spent on this lab.** This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. A completed Main FSM output table (Table 1).
3. Your arm_multi_xx.sv file highlighting your `controller`, `decode`, `condlogic`, and `mainfsm` modules.
4. Your `controllertest_xx.sv` testbench module.
5. Simulation waveforms of the controller module showing (in the given order): CLK, Reset, Cond, OP, Funct, Rd, ALUFlags, ALUControl, ImmSrc, RegSrc, RegWrite, MemWrite, PCWrite, state, and the entire control word (i.e. the 4-nibble word you entered in Table 1) demonstrating each instruction (including taken and non-taken branches). Display all signals in hexadecimal. Does it match your expectations?

© 2014 David Money Harris and Sarah L. Harris

State (Name)	NextPC	Branch	MemW	RegW	IRWrite	AdrSrc	$ResultSrc_{1:0}$	$ALUSrcA_{1:0}$	$ALUSrcB_{1:0}$	ALUOp	FSM Control Word
0 (Fetch)	1	0	0	0	1	0	10	0	1	10	0	0x114C
1 (Decode)	0	0	0	0	0	0	10	0	1	10	0	0x004C
2 (MemAdr)												
3 (MemRead)												
4 (MemWB)												
5 (MemWrite)												
6 (ExecuteR)												
7 (ExecuteI)												
8 (ALUWB)												
9 (Branch)												

Table 1. Main FSM output

© 2014 David Money Harris and Sarah L. Harris

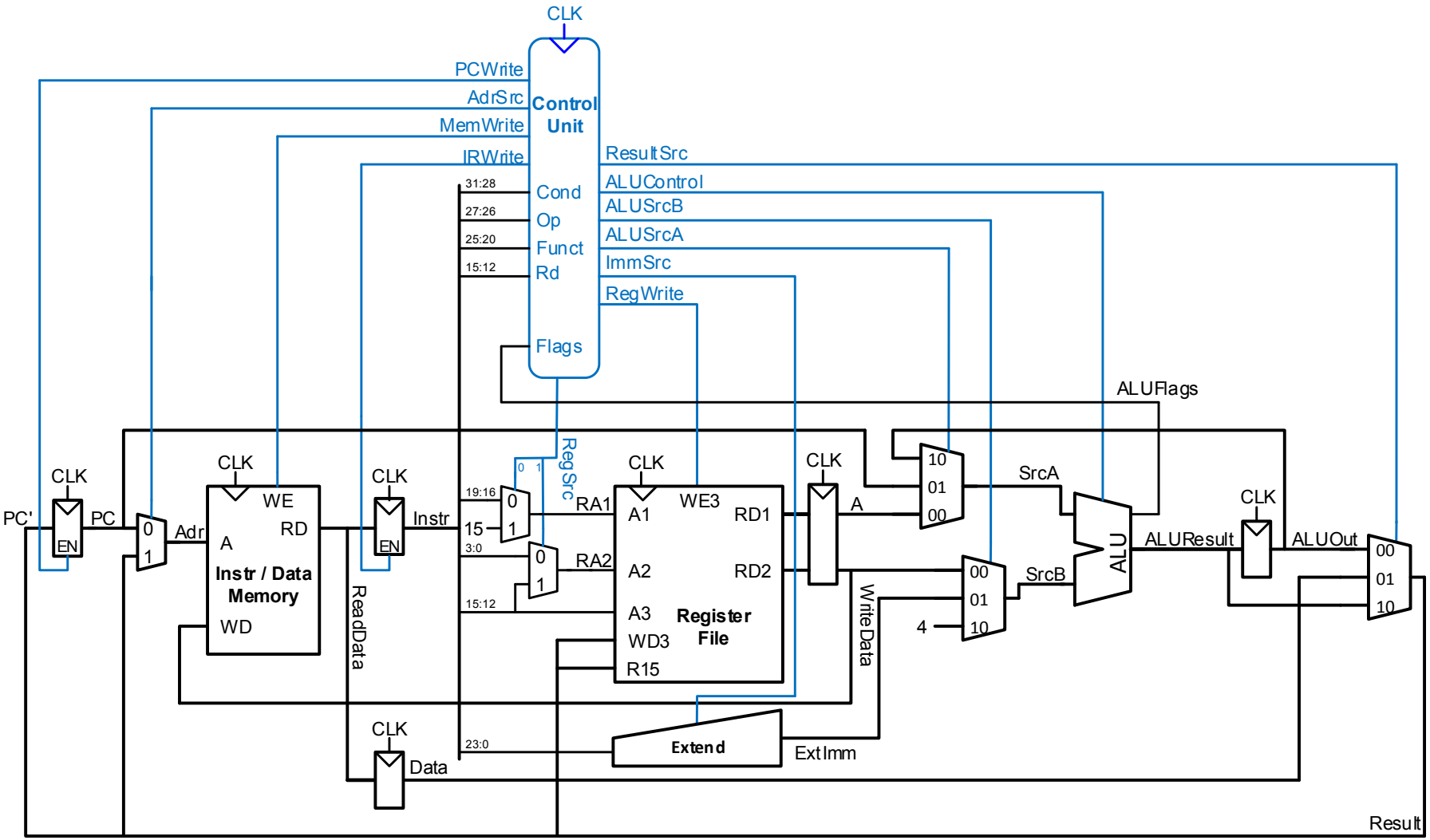


Figure 1. ARM Multicycle Processor

© 2014 David Money Harris and Sarah L. Harris

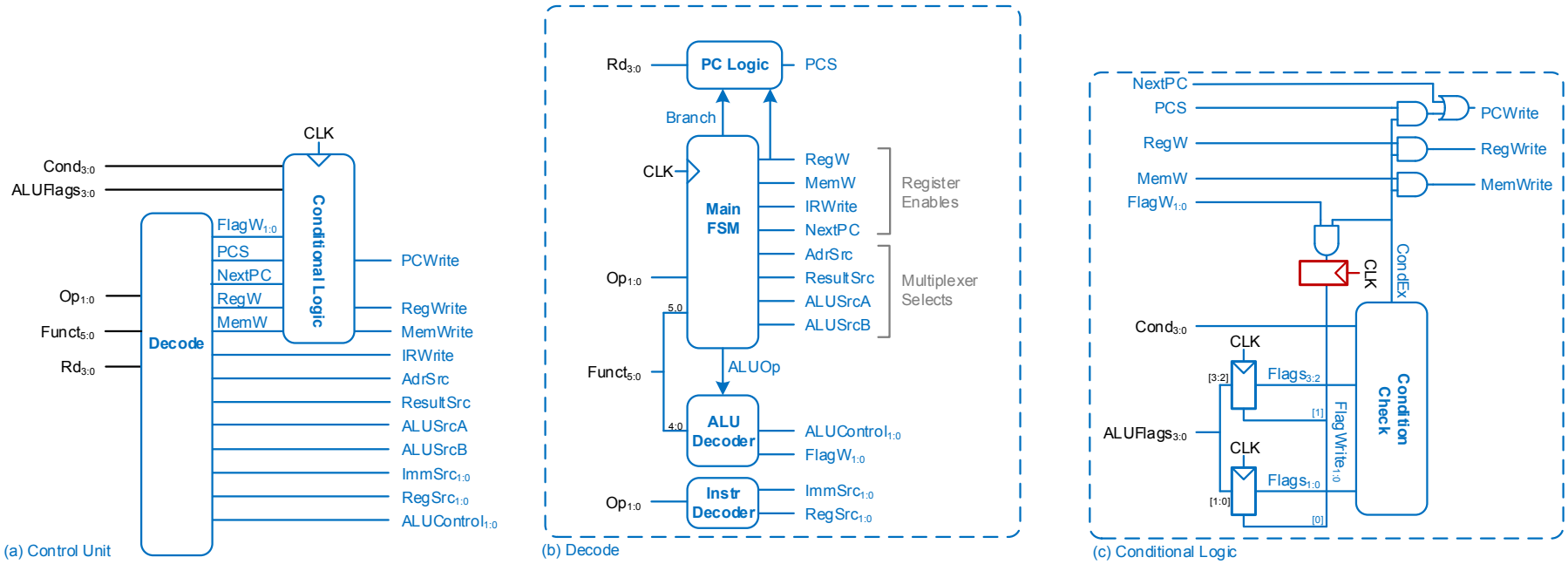


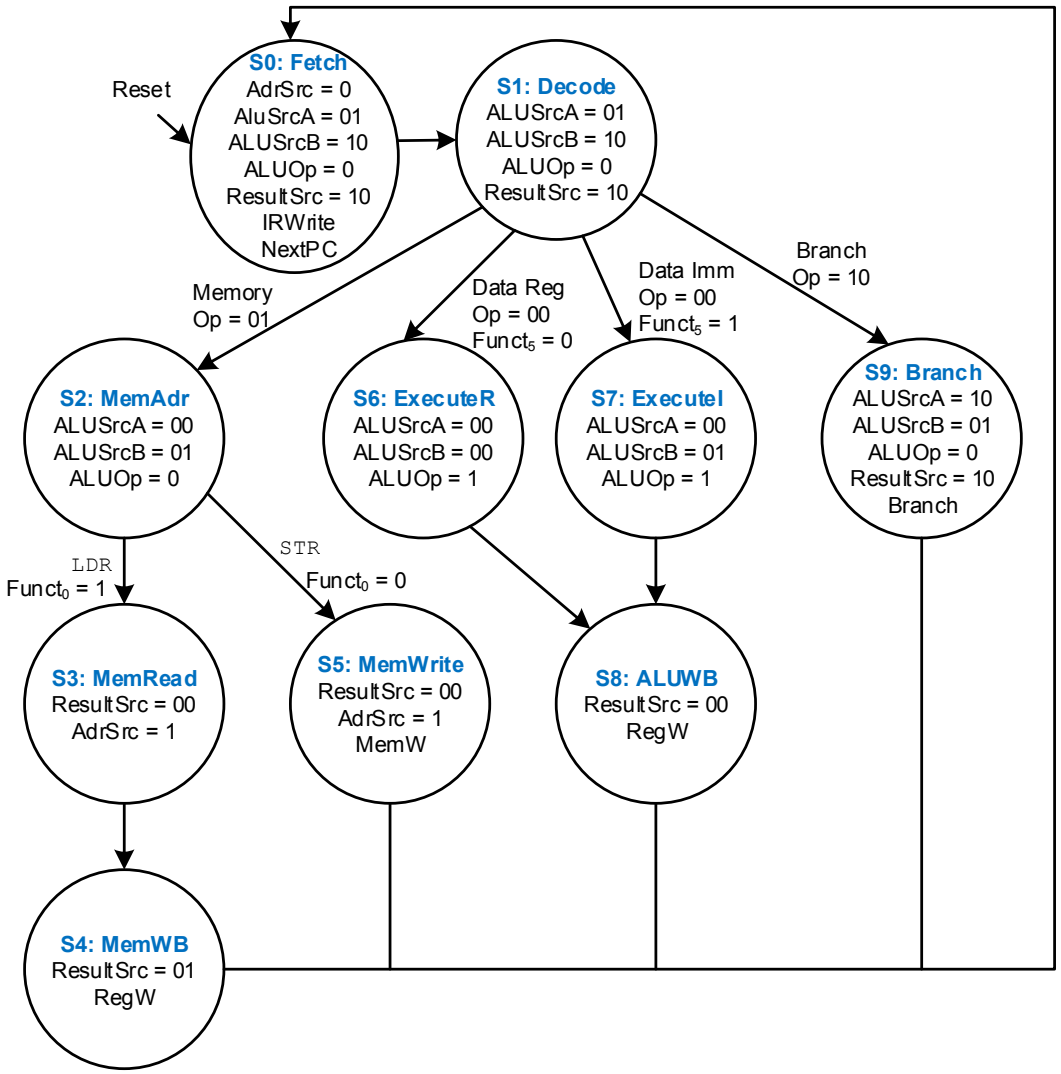
Figure 1. ARM Multicycle Control: (a) controller, (b) Decode unit, (c) Conditional Logic unit

Important: Notice the flip-flop before the $\text{FlagWrite}_{1:0}$ signal.

Explanation: This flip-flop delays updating the Flags until the end of the instruction (in the ALUWB state).

This fixes an error from the Condition Logic module given in the draft text. The error occurs when an instruction is both conditionally executed *and* sets the condition flags (Cond != 1110 AND S=1). Without this flip-flop, the Flags are updated in the ExecuteR or ExecuteI state (due to S=1). Then, in the ALUWB state, writing the destination register (RegWrite) depends on Flags set by the *current* instruction instead of a *prior* instruction. This flip-flop is missing in the figure in draft ARM chapter 7.

© 2014 David Money Harris and Sarah L. Harris



State	Datapath µOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC+4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← ALUOut + offset

Figure 2. ARM Main FSM state transition diagram

© 2014 David Money Harris and Sarah L. Harris