COSC 3406: COMPUTER ORGANIZATION

Home-Work 4

Due Date: Friday, November 12 by 2.00 pm

Instructions for submitting: Upload on D2L

Problems from:
Computer Organization and Design: The hardware/Software Interface ARM Edition, David Patterson and John Hennessy, Morgan Kaufmann, 2016

To receive full marks show your calculations.

Total: 410 points

Chapter 4

4.1 Consider the following instruction:
Instruction: AND Rd, Rn, Rm
Interpretation: Reg[Rd] = Reg[Rn] AND Reg[Rm]
4.1.1 [5] <§4.3>What are the values of control signals generated by the control in Figure 4.10 for this instruction?
4.1.2 [5] <§4.3>Which resources (blocks) perform a useful function for this instruction?
4.1.3 [10] <§4.3>Which resources (blocks) produce no output for this instruction? Which resources produce output that is not used?

4.3 Consider the following instruction mix:

R-type	I-Type	LDUR	STUR	CBZ	B
24%	28%	25%	10%	11%	2%

4.3.1 [5] <§4.4>What fraction of all instructions use data memory?
4.3.2 [5] <§4.4>What fraction of all instructions use instruction memory?
4.3.3 [5] <§4.4>What fraction of all instructions use the sign extend?
4.3.4 [5] <§4.4>What is the sign extend doing during cycles in which its output is not needed?

4.5 In this exercise, we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word: 0xf8014062.
4.5.1 [5] <§4.4>What are the outputs of the sign-extend and the "shift left 2" unit (near the top of Figure 4.23) for this instruction word?
4.5.2 [10] <§4.4>What are the values of the ALU control unit's inputs for this instruction?

4.5.3 [5] <§4.4>What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.
4.5.4 [10] <§4.4> For each mux, show the values of its inputs and outputs during the execution of this instruction. List values that are register outputs at Reg [Xn].
4.5.5 [10] <§4.4> What are the input values for the ALU and the two add units?
4.5.6 [10] <§4.4> What are the values of all inputs for the registers unit?

4.7 Problems in this exercise assume that the logic blocks used to implement a processor's datapath have the following latencies:

I-Mem/ D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

"Register read" is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only. "Register setup" is the amount of time a register's data input must be stable before the rising edge of the clock. This value applies to both the PC and Register File.
4.7.1 [20] <§4.4> Although the control unit as a whole requires 50 ps, it so happens that we can extract the correct value of the Reg2Loc control wire directly from the instruction. Thus, the value of this control wire is available at the same time as the instruction. Explain how we can extract this value directly from the instruction. Hints: Carefully examine the opcodes shown in Figure 2.20. Also, remember that LSR and LSL do not use the Rm field. Finally, ignore STXR.
4.7.2 [5] <§4.4> What is the latency of an R-type instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?
4.7.3 [10] <§4.4> What is the latency of LDUR? (Check your answer carefully. Many students place extra muxes on the critical path.)
4.7.4 [10] <§4.4> What is the latency of STUR? (Check your answer carefully. Many students place extra muxes on the critical path.)
4.7.5 [5] <§4.4> What is the latency of CBZ?
4.7.6 [5] <§4.4> What is the latency of B?
4.7.7 [5] <§4.4> What is the latency of an I-type instruction?
4.7.8 [5] <§4.4> What is the minimum clock period for this CPU?

4.19 [10] <§4.5> Assume that X1 is initialized to 11 and X2 is initialized to 22. Suppose you executed the code below on a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register X5 be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle. See Section 4.7 and Figure 4.51 for details.

```
ADDI X1, X2, #5
ADD X3, X1, X2
ADDI X4, X1, #15
ADD X5, X1, X1
```

4.21 Consider a version of the pipeline from Section 4.5 that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical n-instruction program requires an additional .4*n NOP instructions to correctly handle data hazards.
4.21.1 [5] <§4.5> Suppose that the cycle time of this pipeline without forwarding is 250 ps. Suppose also that adding forwarding hardware will reduce the number of NOPs from .4*n to .05*n, but increase the cycle time to 300 ps. What is the speedup of this new pipeline compared to the one without forwarding?
4.21.2 [10] <§4.5> Different programs will require different amounts of NOPs. How many NOPs (as a percentage of code instructions) can remain in the typical program before that program runs slower on the pipeline with forwarding?
4.21.3 [10] <§4.5> Repeat 4.21.2; however, this time let x represent the number of NOP instructions relative to n. (In 4.21.2, x was equal to .4.) Your answer will be with respect to x.
4.21.4 [10] <§4.5> Can a program with only .075*n NOPs possibly run faster on the pipeline with forwarding? Explain why or why not.
4.21.5 [10] <§4.5> At minimum, how many NOPs (as a percentage of code instructions) must a program have before it can possibly run faster on the pipeline with forwarding?

4.25 Consider the following loop.

```
 LOOP: LDUR X10, [X1, #0]
   LDUR X11, [X1, #8]
   ADD X12, X10, X11
   SUBI X1, X1, #16
   CBNZ X12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.
4.25.1 [10] <§4.7> Show a pipeline execution diagram for the first two iterations of this loop.
4.25.2 [10] <§4.7> Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the SUBI is in the IF stage. End with the cycle during which the CBNZ is in the IF stage.)

4.31 In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
 for(i=0;i!=j;i+=2)
  b[i]=a[i]–a[i+1];
```

A compiler doing little or no optimization might produce the following LEGv8 assembly code:

```
MOV X5, XZR
        B ENT
 TOP: LSL X10, X5, #3
        ADD X11, X1, X10
         LDUR X12, [X11, #0]
        LDUR X13, [X11, #8]
        SUB X14, X12, X13
        ADD X15, X2, X10
        STUR X14, [X15, #0]
        ADDI X5, X5, #2
        ENT: CMP X5, X6
        B.NE TOP
```

The code above uses the following registers:

i	j	a	b	Temporary values
X5	X6	X1	X2	X10–X15

Assume the two-issue, statically scheduled processor for this exercise has the following properties:
1. One instruction must be a memory operation; the other must be an arithmetic/logic instruction or a branch.
2. The processor has all possible forwarding paths between stages (including paths to the ID stage for branch resolution).
3. The processor has perfect branch prediction.
4. Two instruction may not issue together in a packet if one depends on the other. (See page 345.)
5. If a stall is necessary, both instructions in the issue packet must stall. (See page 345.)

As you complete these exercises, notice how much effort goes into generating code that will produce a near-optimal speedup.
4.31.1 [30] <§4.10> Draw a pipeline diagram showing how LEGv8 code given above executes on the two-issue processor. Assume that the loop exits after two iterations.
4.31.2 [10] <§4.10> What is the speedup of going from a one-issue to a two-issue processor? (Assume the loop runs thousands of iterations.)
4.31.3 [10] <§4.10> Rearrange/rewrite the LEGv8 code given above to achieve better performance on the one-issue processor. Hint: Use the instruction "CBZ X6, XZR, DONE" to skip the loop entirely if j = 0.
4.31.4 [20] <§4.10> Rearrange/rewrite the LEGv8 code given above to achieve better performance on the two-issue processor. (Do not unroll the loop, however.)
4.31.5 [30] <§4.10> Repeat Exercise 4.31.1, but this time use your optimized code from Exercise 4.31.4.
4.31.6 [10] <§4.10> What is the speedup of going from a one-issue processor to a two-issue processor when running the optimized code from Exercises 4.31.3 and 4.31.4.
4.31.7 [10] <§4.10> Unroll the LEGv8 code from Exercise 4.31.3 so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the one-issue processor. You may assume that j is a multiple of 4.

4.31.8 [20] <§4.10> Unroll the LEGv8 code from Exercise 4.31.4 so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the two-issue processor. You may assume that j is a multiple of 4. (Hint: Re-organize the loop so that some calculations appear both outside the loop and at the end of the loop. You may assume that the values in temporary registers are not needed after the loop.)
4.31.9 [10] <§4.10> What is the speedup of going from a one-issue processor to a two-issue processor when running the unrolled, optimized code from Exercises 4.31.7 and 4.31.8?
4.31.10 [30] <§4.10> Repeat Exercises 4.31.8 and 4.31.9, but this time assume the two-issue processor can run two arithmetic/logic instructions together. (In other words, the first instruction in a packet can be any type of instruction, but the second must be an arithmetic or logic instruction. Two memory operations cannot be scheduled at the same time.)