**ZACHARY HALPERN**

**CS 320 – Fall 2017**

**Homework Assignment 3**

**Due: Monday, September 25th, in class**

1. Assume that the new instruction called “store sum” (SS) needs to be added to LEGv8 instruction set. In this question, you have to describe changes needed for single-cycle datapath and control logic to implement this new instruction. The instruction description is as follows:

SS Rd, Rm, Rn: Mem[Reg[Rd]]=Reg[Rn]+Reg[Rm];

Assume that the positions of the register fields (Rd, Rn, Rm) within the instruction are the same as in the R-format instructions shown on slide 23. Answer the following questions:

1. [10%] Describe the operation of this new instruction step-by-step showing what data transfers occur during each cycle and what operations are being performed.

Start with an instruction fetch with the program counter. Then decode through the control block. Execute the add instruction of Rn and Rm using the ALU. Write access called to mem spot location stored in Rd.

1. [10%] Describe the changes needed for the processor datapath (shown on Figure 4.23 and slide 29) to implement this new instruction. Depict these changes on a datapath figure. You can either print out a copy of slide 29 and make changes there, or draw relevant parts from scratch.
2. [10%] Describe changes needed for the control logic, including any new control signals. Use the format similar to Figure 4.18 to present your changes.

Added a new line from the control unit to the registers. This will indicate that we need to get the values of the 2 registers then ship them through the ALU. The mux on the other side will either put the data back (for when added) or will write the data to an address, if it’s the memory register location. The address will be stored in a result register where it will then get passed into the data memory location to be written to.

1. Consider the CPU shown in Figure 4.23 in the book and on slide 29. Assume that the logic blocks used to implement the datapath have the following latencies: Instruction memory – 250 ps, data memory – 250 ps, register file – 150 ps, ALU – 200ps, Adder – 150 ps, single gate – 5ps, sign-extension logic – 50ps, control logic – 50 ps, register read – 30ps, register setup – 20ps. “Register read” is the time needed after the rising edge of the clock for the new register value to appear at 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 PC, register file and memory. You do not have to account for the register file and memory write delays separately. Assume that we only execute five instructions – LDUR, STUR, CBZ, B and ADD.
2. [5%] What is the latency of the ADD instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?

ADD: Register read for PC + Register setup + instruction memory + Register file + ALU + Control logic

30 + 20 + 250 + 150 + 200 + 50 = 700ps

1. [5%] What is the latency of LDUR?

LDUR: Reg read for PC + Reg setup + instruction memory + register file + ALU + data memory + sign extension + control block

30 + 20 + 250 + 150 + 200 + 250 + 50 + 50 = 1000ps

1. [5%] What is the latency of STUR?

STUR: Reg read for PC + Reg setup + instruction memory + register file + ALU + data memory + sign extension + control block

30 + 20 + 250 + 150 + 200 + 250 + 50 + 50 = 1000ps

1. [5%] What is the latency CBZ?

CBZ: Reg read for PC + Reg setup + instruction memory + register file + sign extension + control block + ALU control + ALU + adder

30 + 20 + 250 + 150 + 50 + 50 + 200 + 150 = 900ps

1. [5%] What is the latency of B?

B: Reg read for PC + Reg setup + Instruction memory + Sign extension + ALU Control + Adder

30 + 20 + 250 + 50 + 50 + 150 = 550ps

1. [5%] What is the minimal clock period for this CPU?

MAX(all instructions) = 1000ps

Consider now the addition of a multiplier to this CPU. Assume that this addition will add 300ps to the latency of the ALU, but will reduce the number of instructions by 5%, because there will no longer be a need to emulate the multiply instruction in software.

1. [5%] What is the processor cycle time with the addition of the multiplier?

Still 1000ps

1. [5%] What is the speedup achieved by this addition?

1000 / (1300 \* 0.95) so it’s a slower system

1. [10%] What is the slowest that the new ALU can be and still result in improved performance?

1000 / x \* .95 = 1, so 1052ps

1. [20%] Consider a 5-stage pipelined datapath with data forwarding logic. Assume that 50% of instructions executed are the load instructions (LDUR), and that half of these load instructions have a dependent instruction that immediately follows the load. There are no other dependencies in the program. Estimate the speedup of this processor compared to single-cycle non-pipelined implementation.

Assume 100 instructions run, 50 are LDUR, 25 have dependency on instruction prior to load

100 instructions \* 5 stages = 500 stages / cycles without pipelining

**F D EX MEM WB**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I1 |  |  |  |  |
| I2 | I1 |  |  |  |
| I3 | I2 | I1 |  |  |
| I4 | I3 | I2 | I1 |  |
| I5 | I4 | I3 | I2 | I1 |

First set of 75 instructions that have no dependencies run in (75+5-1 = 79) cycles

Last 25 that have dependencies will require a bubble

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I75 |  |  |  |  |
| I76 | I75 |  |  |  |
| I77 | I76 | I75 |  |  |
|  |  |  | I75 |  |
| I78 | I77 | I76 |  | I75 |
| I78 | I77 |  | I76 |  |
| I79 | I78 | I77 |  | 76 |

This makes it take 6 cycles per instruction, except the first one so

25+(25\*6)-1 = 174 cycles

174+79 = 253 cycles

500/253 = 1.93x speedup