ECE M116C - CS M151B Computer Architecture Systems - UCLA

Homework 2

Fall 2022

Deadline: Friday, Oct. 14, 11:55 PM

(Upload it to Gradescope)

Q1. Consider the following latencies for different units in the (single-cycle) datapath described in the class:

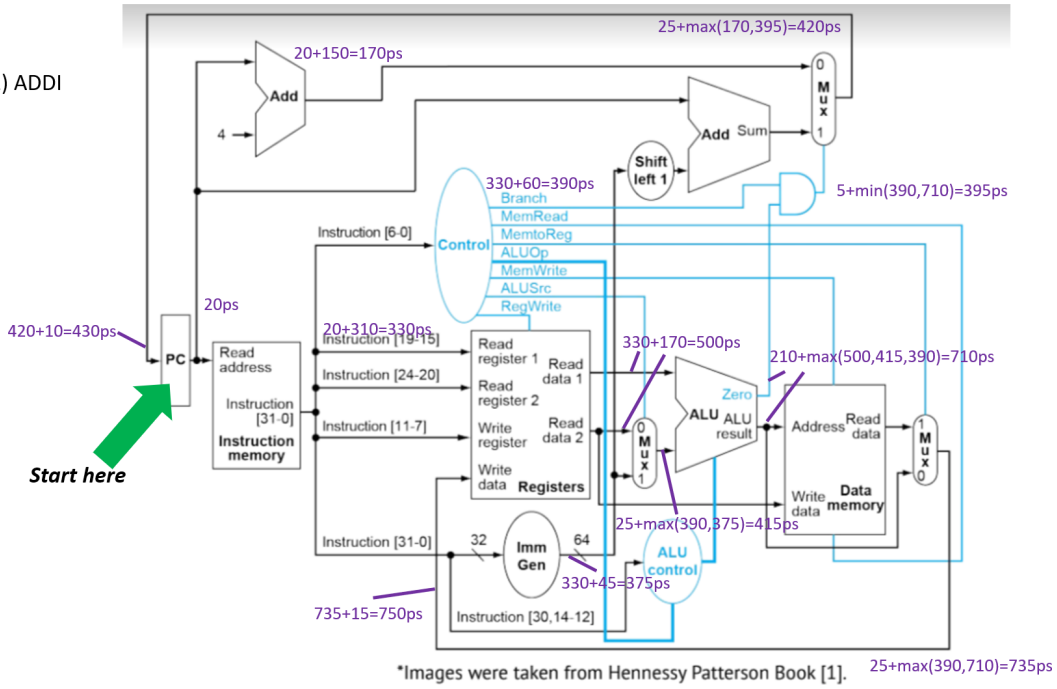
Mem (I and D) (read)	**Reg. File (read)**	**Reg. File (write)**	**Mux (any size)**	**ALU**	**Gate (any)**
310 ps	170 ps	15 ps	25 ps	210 ps	5 ps

Imm Gen	**PC (read)**	**Adder**	**PC (write)**	**Mem (D) (write)**	**Control**
45 ps	20 ps	150 ps	10 ps	15 ps	60 ps

Answer the following questions:

a) What is the latency for ADDI instruction?
(i.e., how long does it take to complete this instruction? Assume that the beginning of the instruction is reading the PC. Note that there might be multiple parallel paths, so you need to report the MAX). Show your work.

1) ADDI



*Images were taken from Hennessy Patterson Book [1].

NOTE: Times after each step represent total time elapsed (from accessing PC)

Datapath:
Accessing PC: 20ps
Accessing instruction: 330ps
Both inputs of ALU ready: 500ps
Output of ALU:710ps
Data through "MemToReg" mux: 735ps
Data written bag to regfile: 750ps

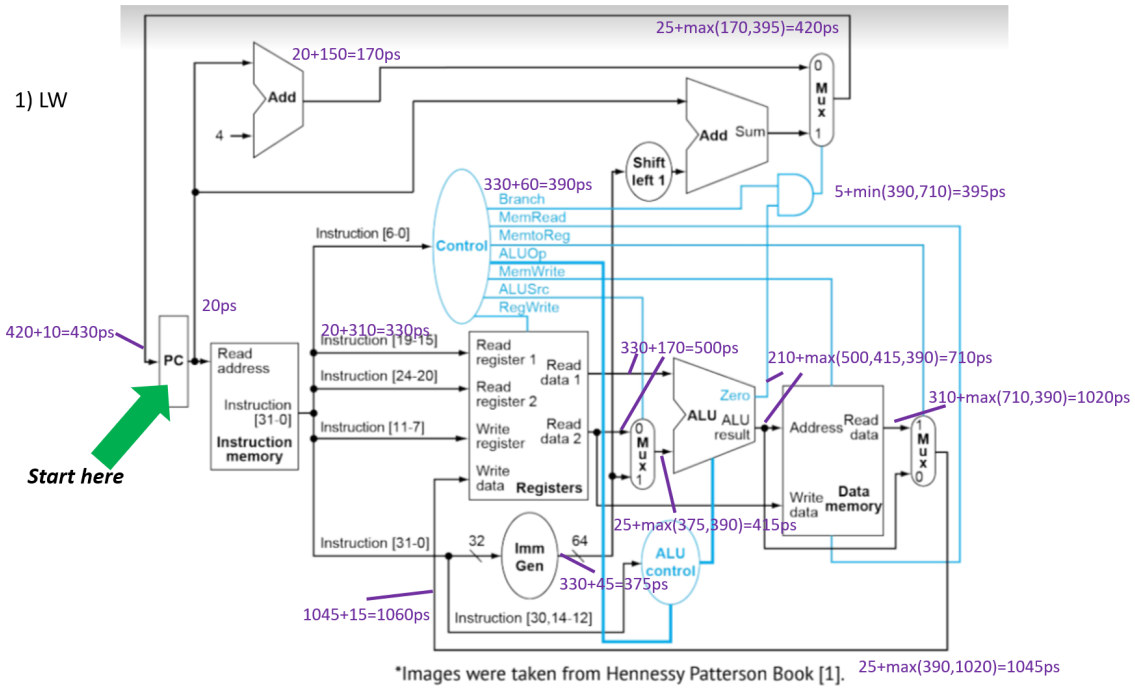
PC Updating:
Accessing PC: 20ps
Adding 4 to PC: 170ps
Selecting correct PC: 420ps
Updating PC: 430ps

max(750ps, 430ps) = 750ps

Note: for the "and" gate that provides input to the uppermost mux in the PC update portion, we take min(390, 710) because we know the instruction

is not a branch whenever either the branch control bit, or the zero flag, is first zero. This is the same reasoning for the LW and SW instructions.

b) What is the latency for LW instruction?



*Images were taken from Hennessy Patterson Book [1].

NOTE: Times after each step represent total time elapsed (from accessing PC)

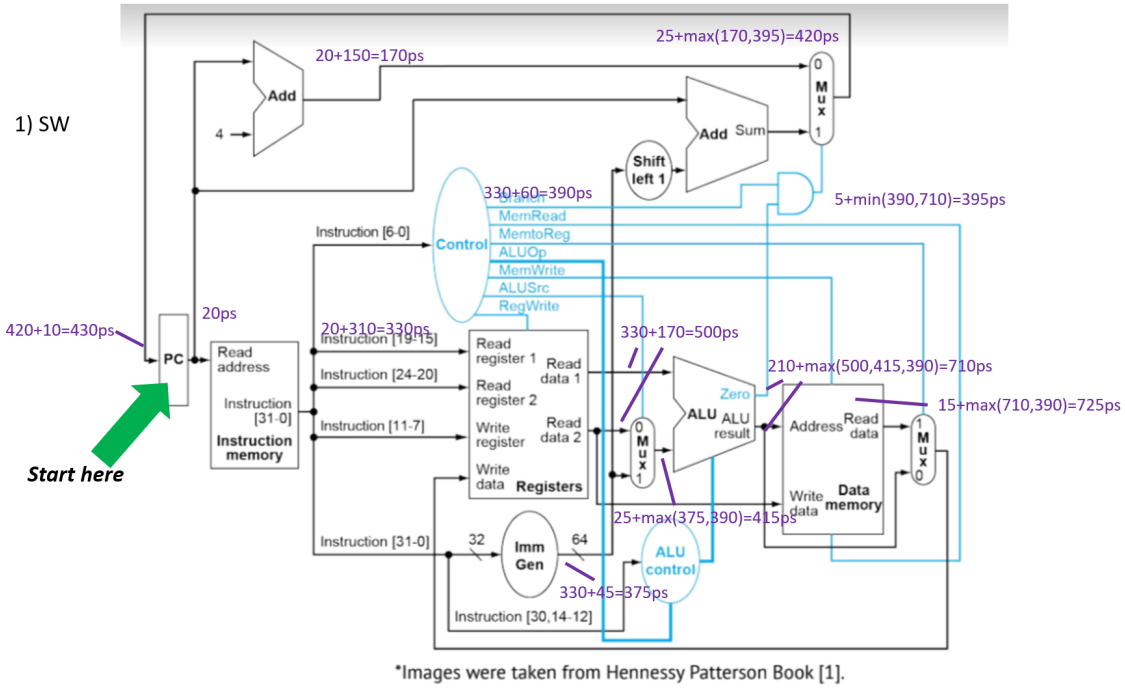
Datapath:
Accessing PC: 20ps
Accessing instruction: 330ps
Both inputs of ALU ready: 500ps
Output of ALU:710ps
Data read from memory: 1020ps
Data through "MemToReg" mux: 1045ps
Data written bag to regfile: 1060ps

PC Updating:
Accessing PC: 20ps
Adding 4 to PC: 170ps
Selecting correct PC: 420ps

Updating PC: 430ps

max(1060ps, 430ps) = 1060ps

c) What is the latency for SW instruction?



*Images were taken from Hennessy Patterson Book [1].

NOTE: Times after each step represent total time elapsed (from accessing PC)

Datapath:
Accessing PC: 20ps
Accessing instruction: 330ps
Both inputs of ALU ready: 500ps
Output of ALU:710ps
Data stored into memory: 725ps

PC Updating:
Accessing PC: 20ps
Adding 4 to PC: 170ps
Selecting correct PC: 420ps
Updating PC: 430ps

max(725ps, 430ps) = 725ps

NOTE: you may be wondering why we don't include the latency of the "MemToReg" mux and writing back to memory. This is because store instructions don't write back to the register file, thus we don't need to include these latencies in our slowest path.

d) What is the minimum clock cycle for this processor?

max(750, 1060, 725) = 1060ps

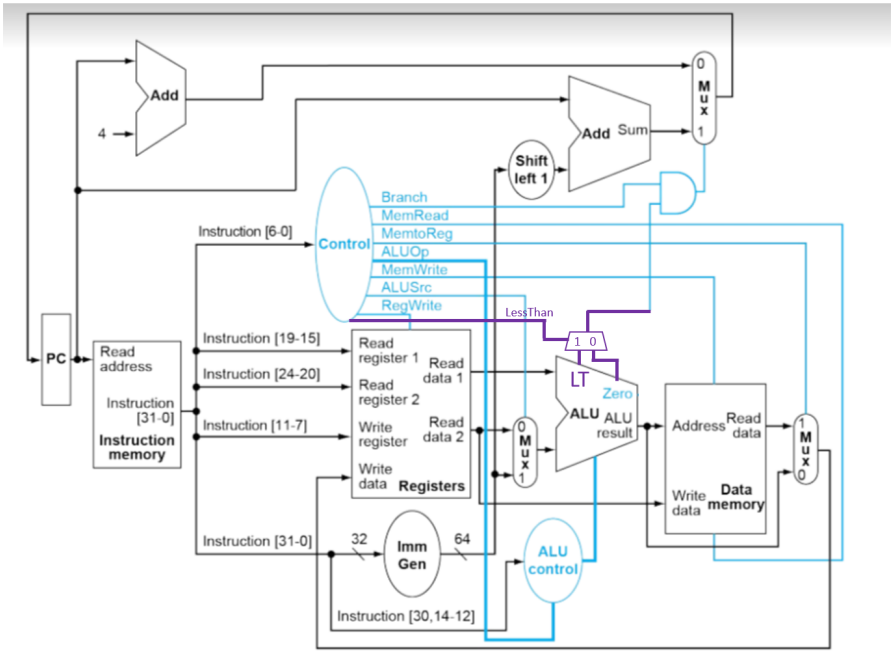
The minimum clock cycle is the clock cycle period that is long enough such that the slowest instruction can finish processing. For these three instructions, the load instruction has the largest latency (1060 ps), thus, the clock cycle period needs to be >=1060ps.

Q2. We want to add two new instructions to our datapath+controller design discussed in class. Redraw the datapath+controller with the changes needed for supporting these two instructions (show the control signals with a different color). You can either draw two different plots or show the changes needed for both instructions in the same plot.

- The first instruction is BLT (branch less than).
- The second instruction is a new instruction called "shift and add" where the goal is shifting left rs1 with the value in rs2 and then adding the shifted value with its original value and finally storing the result in rd. Here is the mathematical definition (assume that this is an R-type instruction):

```
ShiftAdd rd, rs1, rs2 ----->  x[rd] = (x[rs1]<<x[rs2]) + x[rs1]
```

2) BLT

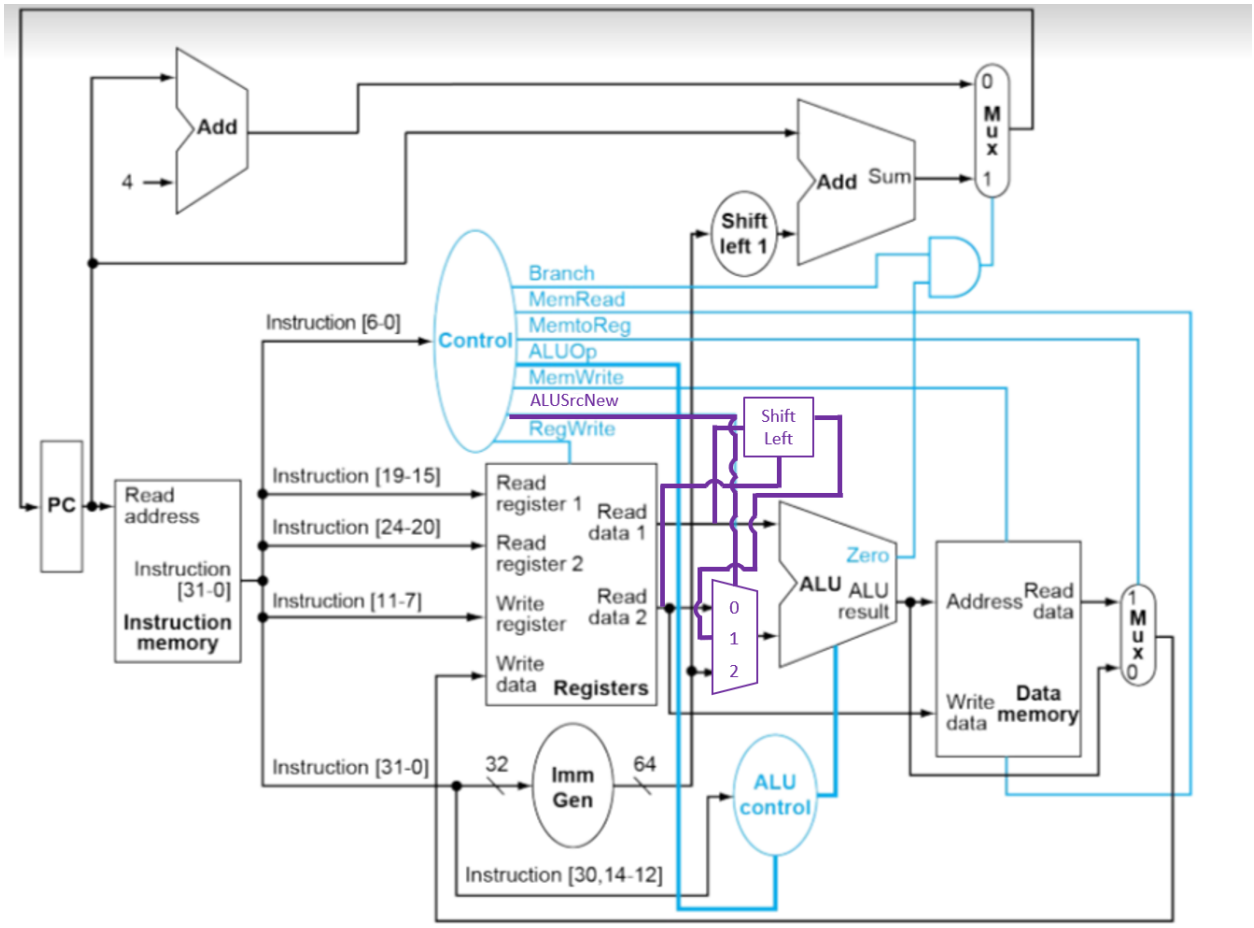


*Images were taken from Hennessy Patterson Book [1].

To support the instruction "blt", we need another "flag" from the ALU. Instead of the "zero" flag that states whether two register values are equal, we need a flag that states if the value in rs1 is less than the value in rs2. The operation within the ALU to do this would be a subtraction where we grab the sign bit of the result for the flag. If the subtraction result is negative, the sign bit is 1, in which the "LT" flag would be 1.

One can then use a mux to select between the "LT" and "Zero" flags, with the selector bit being "LessThan", a new control bit coming from the controller. This way, we can use the correct flag for the "and" gate used in determining if a branch was taken.

Note: one can also simplify the datapath by resolving the zero flag and blt flag within the ALU. Instead of adding an extra port to the ALU and having a mux to select between the two flags to pick the input into the "and" gate, we can use an "or" gate within the ALU. When one of the zero flag or the blt flag is true, this means we want to branch and we can use the "or" gate output directly into the "and" gate (no extra mux needed). This is the most cost effective solution in fact.



*Images were taken from Hennessy Patterson Book [1].

To support the shift and add instruction, we need to add/edit a few pieces of hardware and control logic. First, we need to implement the left shifter. We can use a "shift left" block that takes in the value from RS2 as the "shift" amount input and the value in RS1 as the data input. Once we have this shifted value, we need to select between this shifted value, the value in RS2, and the imm, as in the "shift and add" instruction we need to add the value in RS1 to this shifted value. We can modify the "ALUSrc" mux to support 3 inputs (shifted value, value in RS2, imm). The original control signal "ALUSrc" is modified to "ALUSrcNew" and uses 2 bits instead of 1 bit to select between 3 options within the mux. The upper input of the ALU stays the same (feed the value of RS1 through).

NOTE: another option to do the shift and add is to do the shift on the upper side of the ALU (have the result of the shift go into the first ALU input instead). This would require one to add an additional mux to select between the value in RS1 (default) and the shifted value (to maintain normal operation), which adds more

complexity to the hardware. The "ALUSrcNew" mux would still need 3 inputs (value in RS1, value in RS2, imm).

NOTE: another option is to have the shift occur inside the ALU, as the ALU takes in the value in RS1 as input and the value in RS2 (assuming select it in ALUSrc mux). This requires one to select the correct output of the ALU, meaning we need to have a ALUControl output that can select the shift and add result.