COSE222: Computer Architecture
Assignment #3

Due: Nov 29, 2020 (Sunday) 11:59pm on Blackboard

Solutions (Total score: 219)

Please answer for the questions. Write your student ID and name on the top of the document. Submit your homework with "**PDF**" format only. (You can easily generate the pdf files from Microsoft Word or HWP. You can also handwrite your answers to scan the handwritten documents with "**PDF**" format. You may use the document capture applications such as "Office Lens" for scanning your documents with your smartphones.)

The answer rules:
(1) You can write answers in both Korean and English.
(2) Please make your final answer numbers have two decimal places.
(3) Performance of A is improved by *NN* % compared to performance B if PerfA / PerfB = 1.*NN*.

The following figure shows the RISC-V single-cycle processor covered during the classes.

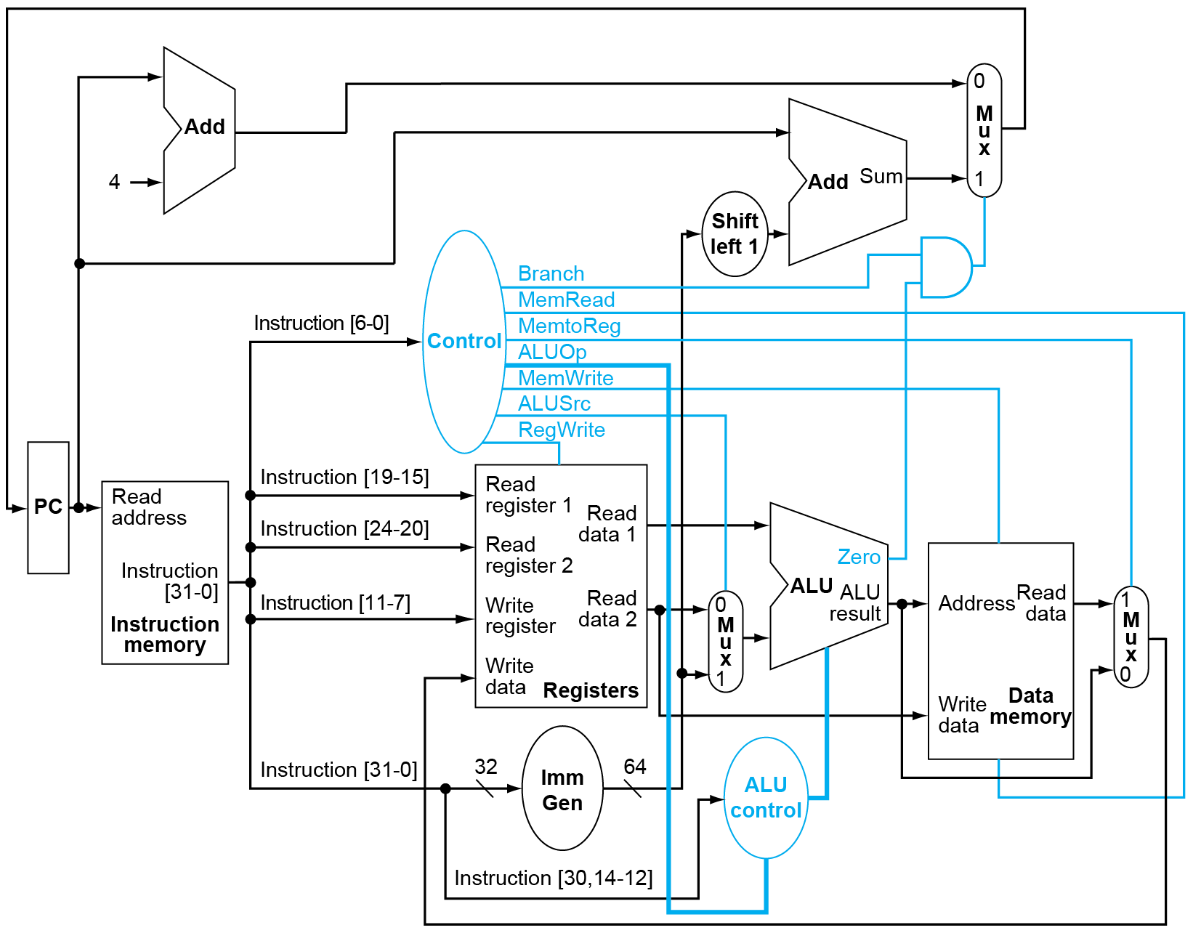


Figure 1. RISC-V single-cycle processor

1. Consider the following instruction mix: (I-type means instructions that use *immediate* data)

R-type	I-type (non-ld)	Load	Store	Branch	Jump
27%	23%	20%	15%	11%	4%

(a) What fraction of all instructions use data memory? [5]

(b) What fraction of all instructions use instructions memory? [5]

(c) What fraction of all instructions use the sign extend? [5]

(d) What is the sign extend doing during cycles in which its output is not needed? [5]

2. For the single-cycle processor design, we do *NOT* consider I-type instructions such as `addi` and `andi`. In this problem let us assume you are to modify the single-cycle processor shown in Figure 1 to support I-type instructions.

(a) What additional logic blocks, if any, are needed to add I-type instructions to the single-cycle processor shown in Figure 1? List any required logic blocks and explain their purpose. [5]

(b) List the values of the signals generated by the control unit for `addi`. Explain the reasoning for any "don't care" control signals. [10]

3. Let us assume that the clock period of the single-cycle processor shown in Figure 1 is 1000 ps. Consider the addition of a multiplier to the CPU. This addition will add 250 ps to the latency of the ALU, but will reduce the number of instructions by 4% (because there will no longer be a need to emulate the multiply instructions).

(a) What is the clock cycle time with this improvement? [5]

(b) What is the **speed** up achieved by adding this improvement? [5]

(c) What is the longest delay by the newly added multiplier to result in performance improvement at least? [5]

4. Let us assume that you are requested to add a new instructions, `lwi.d  rd, rs1, rs2` ("load with increment") to RISC-V. Explain how to figure out control signals of the single-cycle processor shown in Figure 1. [10]

5. Let us assume that you are requested to add additional branch instructions such as BNE (branch if not equal), BLT (branch if less than), and BGE (branch if greater or equal) in the single-cycle processor design in Lab #3. The following table shows the information of the branch instructions supported by the single-cycle processor.

Inst.	Description	opcode	funct3
BEQ	`if (R[rs1] == R[rs2]) PC = PC + {imm, 1'b0}`	1100011	000
BNE	`if (R[rs1] != R[rs2]) PC = PC + {imm, 1'b0}`		001
BLT	`if (R[rs1] < R[rs2]) PC = PC + {imm, 1'b0}`		100
BGE	`if (R[rs1] >= R[rs2]) PC = PC + {imm, 1'b0}`		101

Assume that the values in `rs1` and `rs2` are all *signed* values and overflows/underflows do not happen in ALU. We add the signal (`branch_true`) that indicates the condition of a branch instruction is true. Complete the following SystemVerilog snippet to support the branch instructions in the above table. (*Hint: you should use `alu_result` to check branch results.*) [15]

```
logic        branch, branch_true;
logic        alu_zero;      // zero detection in ALU result
logic [63:0] alu_result;    // ALU result

assign branch = (opcode==7'b1100011) ? 1'b1: 1'b0;

always_comb begin
    case (funct3[2:0])
        3'b000: branch_true = alu_zero;
        3'b001: branch_true =                    // ← FILL THIS
        3'b100: branch_true =                    // ← FILL THIS
        3'b101: branch_true =                    // ← FILL THIS
        default: branch_true = 1'b0;
    endcase
end

assign pc_next = (branch & branch_true) ? pc_next_branch: pc_next_plus4;
```

6. In this problem we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies.

IF	ID	EX	MEM	WB
250 ps	400 ps	250 ps	300 ps	200 ps

Also, assume that instructions executed by this processor are broken down as follows.

ALU/logic	Jump/Branch	Load	Store
45%	20%	20%	15%

(a) What is the clock cycle time in a pipelined and non-pipelined processor? [5]

(b) What is the total latency of an `ld` instruction in a pipelined and non-pipelined processor? [5]

(c) If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would split and what is the new clock cycle time of the processor? [5]

(d) Assuming there are no stalls or hazards, what is the utilization of the data memory? [5]

(e) Assuming there are no stalls or hazards, what is the utilization of the write-register port of the register file? [5]

7. What is the minimum number of cycles needed to completely execute *N* instructions on a CPU with *K* stage pipeline? Justify your formula. [5]

8. Add “`NOP`” (No operation) instructions to the code below so that it will run correctly on the 5-stage pipelined processor which we learned during the classes. Answer for the following conditions of the hardware supports.

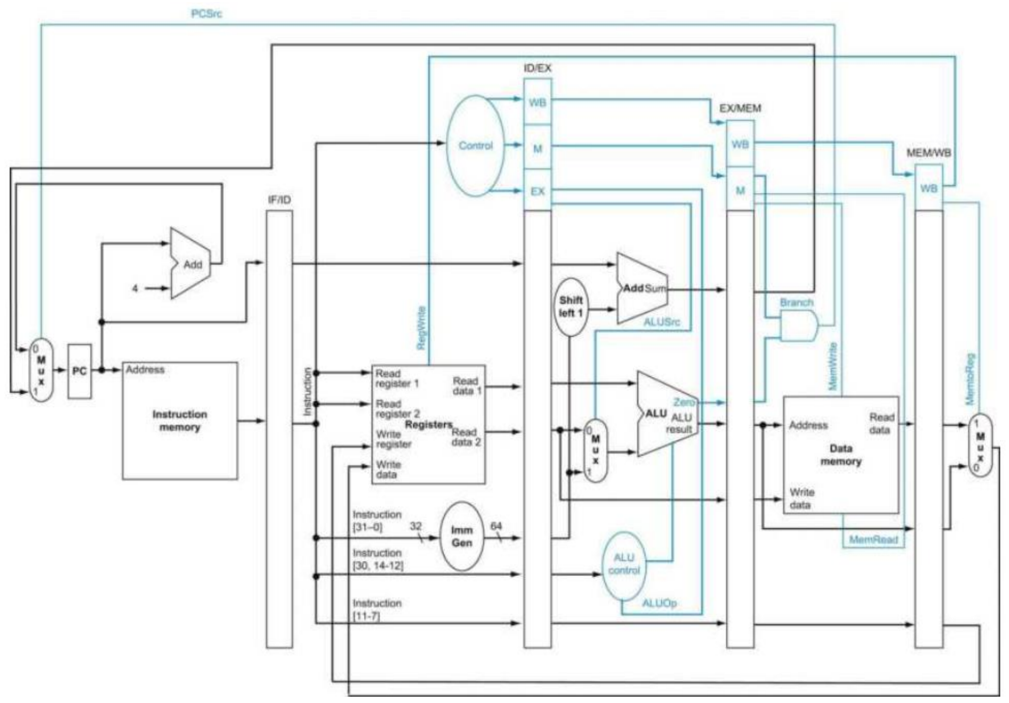
```
ld x4, 0(x5)
add x11, x4, x6
add x11, x11, x11
add x12, x11, x4
```

(a) The processor does not handle data hazards. Only *internal forwarding* inside of the register file is supported. [8]

(b) The processor does not handle data hazards. The processor supports the data forwarding from *MEM/WB pipeline register* and *internal forwarding* inside the register file. [8]

(c) The processor does not handle data hazards. The processor supports the data forwarding from *EX/MEM and MEM/WB pipeline registers* and *internal forwarding* inside the register file. [8]

9. Consider the following 5-stage pipelined processor architecture. (See Figure 4.49 in the textbook) We will use this processor architecture as a baseline. Namely, branch instructions are resolved in EX stage and the result of a branch instruction is transferred in MEM stage. Assume that the hazard detection unit is implemented to detect hazard conditions and control pipelining of the processor.



Let us assume that the following assembly code is executed in the 5-stage pipelined processor. Assume that the branch instruction (beq) is taken.

```
        ld   x29, 12(x16)
        ld   x30, 8(x16)
        sub  x30, x29, x30
        beq  x30, x5, label
        add  x15, x11, x14
        and  x14, x4, x5
label:  sub  x15, x30, x14
```

(a) Assume that only internal forwarding inside of the register file is supported and no data forwarding s supported from the pipeline registers. Draw a pipeline diagram to show where the code above will stall. Indicate flushes as "---". [10]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`ld x29`	IF	ID	EX	ME	WB										
`ld x30`															
`sub x30`															
`beq x30`															
`add x15`															
`and x14`															
`sub x15`															

(b) Assume that data forwarding is supported from MEM/WB pipeline register and interna forwarding is supported inside of the register file. Draw a pipeline diagram to show where the code above will stall. Indicate flushes as "---". [10]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`ld x29`	IF	ID	EX	ME	WB										
`ld x30`															
`sub x30`															
`beq x30`															
`add x15`															
`and x14`															
`sub x15`															

(c) Assume that data forwarding is supported from EX/MEM and MEM/WB pipeline registers and internal forwarding inside of the register file is supported. Draw a pipeline diagram to show where the code above will stall. Indicate flushes as "---". [10]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`ld x29`	IF	ID	EX	ME	WB										
`ld x30`															
`sub x30`															
`beq x30`															
`add x15`															
`and x14`															
`sub x15`															

(d) Assume that zero detection is performed using the data outputs from the register file in ID stage. Data forwarding is supported from EX/MEM and MEM/WB pipeline registers and internal forwarding inside of the register file is supported. Draw a pipeline diagram to show where the code above will stall. Indicate flushes as "---". (Hint: you should consider data hazards caused by branch instructions.) [10]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`ld x29`	IF	ID	EX	ME	WB										
`ld x30`															
`sub x30`															
`beq x30`															
`add x15`															
`and x14`															
`sub x15`															

10. If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. As a result the MEM and EX stages can be overlapped and the pipeline has only 4 stages.

(a) How will the reduction in pipeline depth affect the clock cycle time? [5]

(b) How might this change improve the performance of the pipeline? [5]

(c) How might this change degrade the performance of the pipeline? [5]

10. The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this problem, let us assume that the breakdown of dynamic instructions into various instruction categories is as follows.

R-type	beq/bne	jal	ld	sd
40%	20%	5%	25%	10%

Also assume the following branch predictor accuracies.

Always-taken	Always-Not-Take	2-bit predictor
40%	60%	90%

(a) Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the ID stage and applied in the EX stage that there are no data hazards, and that no delay slots are used. [5]

(b) Repeat the above problem for the “always-not-taken” predictor. [5]

(c) Repeat the above problem for the 2-bit predictor. [5]

11. Consider the following C code:

```
for (i = 0; i < 100; i++)          // 1st loop
  for (j = 0; j < 100; j++)        // 2nd loop
    ……
```

Let us assume that there are no conditional branches inside of the loops. When this C code is compiled, the conditional branch instructions are at the end of the loops. If the branch is taken the program jumps to the beginning of the loop.

(a) Let us assume that **one-bit** branch predictors are applied for the both loops. Calculate the hit ratio (%) of each branch predictor. The branch predictors are initialized as "0" (NOT TAKEN), and each branch predictor can be distinguished by program counter. [10]

For the first loop:

For the second loop:

(2) Let us assume that two-bit branch predictors are applied for the both loops. Calculate the hit ratio (%) of each branch predictor. The branch predictors are initialized as "00" (not taken), and each branch predictor can be distinguished by program counter. For the two-bit predictor, states "00" and "01" are predicted as "NOT TAKEN" and states "10" and "11" are predicted as "TAKEN". [10]

For the first loop:

For the second loop: