CS 61C RISC-V Single Cycle Datapath, Pipelining

Fall 2023 Discussion 7

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 The single cycle datapath makes use of all hardware units for each instruction.

False. All units are active in each cycle, but their output may be ignored (gated) by control signals.

1.2 It is possible to execute the stages of the single cycle datapath in parallel to speed up execution of a single instruction.

False. Each stage depends on the value produced by the stage before it (e.g., instruction decode depends on the instruction fetched).

1.3 If the delay of reading from IMEM is reduced, then any (non-empty) program using the single cycle datapath will speed up.

True. Since every instruction must read from IMEM during the instruction fetch stage, making the IMEM faster will speed up every single instruction.

1.4 The control signals used throughout all datapath stages to guide a correct 'execution line' all come from decoding an instruction's unique binary encoding only.

False. PCSel is used to determine what instruction will be executed next, which is not immediately known for a branch instruction. This is discovered in the Branch Comparator after the register values are retrieved from the RegFile. The comparator then produces the BrEq (branch equal) and BrLt (branch less than) flags, which are used in the control logic with the instruction encoding to produce the PCSel signal.

1.5 Storing instructions and loading instructions are the only instructions that require input/output from DMEM.

True. For all other instructions, we don't need to read the data that is read out from DMEM, and thus don't need to wait for the output of the MEM stage.

1.6 It is possible to use both the output of the immediate generator and the value in register rs2.

False. You may only use *either* the immediate generator or the value in register rs2. Notice in our datapath, there is a mux with a signal (BSel) that decides whether we use the output of the immediate generator or the value in rs2.

2 Single-Cycle CPU

2.1 For this worksheet, we will be working with the single-cycle CPU datapath provided the last page.

(a) Explain what happens in each datapath stage, and which hardware units in the datapath are used.

IF Instruction Fetch

Send address to the instruction memory (IMEM), and read IMEM at that address.
Hardware units: PC register, +4 adder, PCSel mux, IMEM

ID Instruction Decode

Generate control signals from the instruction bits, generate the immediate, and read registers from the RegFile.
Hardware units: RegFile, ImmGen

EX Execute

Perform ALU operations, and do branch comparison.
Hardware units: ASel mux, BSel mux, branch comparator, ALU

MEM Memory

Read from or write to the data memory (DMEM).
Hardware units: DMEM

WB Writeback

Write back either PC + 4, the result of the ALU operation, or data from memory to the RegFile.
Hardware units: WBSel mux, RegFile

(b) On the datapath, fill in each **round** box with the name of the datapath component, and each **square** box with the name of the control signal.

See last page.

(c) List all possible values that each control signal may take on for the single cycle datapath, then briefly describe what each value means for each signal.

Signal Name	Values	Signal Name	Values
PCSel		RegWEn	
ImmSel		BrEq	
BrLt		ALUSel	
MemRW		WBSel	

PCSel: 0: OldPC + 4 is next PC; 1: ALU value (for branches, jumps, etc...)
RegWEn: 0: WB value is not written to register; 1: WB is written
ImmSel: 0-4 used for I, B, S, J, U type immediates; 5-7 unused
BrEq: 0 when inputs not equal; 1 when inputs equal
BrLt: 0 when `rs1` is not less than `rs2`; 1 when it is less than
ALUSel: note, this is using the same design reference 61C does; may differ based on CPU design; you do **not** have to memorise all of these!

- 0: add (`rd = rs1 + rs2`)
- 1: sll (`rd = rs1 << rs2`)
- 2: slt (`rd = (rs1 < rs2 (signed)) ? 1 : 0`)
- 3: unused
- 4: xor (`rd = rs1 ^ rs2`)
- 5: srl (`rd = (unsigned) A >> B`)
- 6: or (`rd = rs1 | rs2`)
- 7: and (`rd = rs1 & rs2`)
- 8: mul (`rd = (signed) (rs1 * rs2)[31:0]`)
- 9: mulh (`rd = (signed) (rs1 * rs2) [63:32]`)
- 10: unused
- 11: mulhu (`rd = (unsigned) (rs1 * rs2) [63:32]`)
- 12: sub (`rd = rs1 - rs2`)
- 13: sra (`rd = (signed) rs1 >> rs2`)
- 14: unused
- 15: bsel (`rd = rs2`)

MemRW: 0 for all non-write-to-memory operations; 1 to enable writing to main memory
WBSel: 0 for DMEM read output; 1 for ALU output; 2 for PC + 4; 3 unused

2.2 Fill out the following table with the control signals for each instruction based on the datapath on the last page. If the value of the signal does not affect the execution of an instruction, use the * (don't care) symbol to indicate this. If the value of the signal **does** affect the execution, but can be different depending on the program, list all possible values (for example, for a signal with possible values of 0 and 1, write 0/1).

	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
add	*	*	0 (PC + 4)	*	*	0 (Reg)	0 (Reg)	add	0	1	1 (ALU)
ori	*	*	0	I	*	0 (Reg)	1 (Imm)	or	0	1	1 (ALU)
lw	*	*	0	I	*	0 (Reg)	1 (Imm)	add	0	1	0 (MEM)
sw	*	*	0	S	*	0 (Reg)	1 (Imm)	add	1	0	*
beq	0/1	*	0/1	B	*	1 (PC)	1 (Imm)	add	0	0	*
jal	*	*	1 (ALU)	J	*	1 (PC)	1 (Imm)	add	0	1	2 (PC + 4)
blt	*	0/1	0/1	B	0	1 (PC)	1 (Imm)	add	0	0	*

3 Timing the Datapath

3.1 Clocking Methodology

- A **state element** is an element connected to the clock (denoted by a triangle at the bottom). The **input signal** to each state element must stabilize before each **rising edge**.
- The **critical path** is the longest delay path between state elements in the circuit. The circuit cannot be clocked faster than this, since anything faster would mean that the correct value is not guaranteed to reach the state element in the alloted time. If we place registers in the critical path, we can shorten the period by **reducing the amount of logic between registers**.

For this exercise, the times for each circuit element is given as follows:

Clk-to-Q	RegFile Read	PC/RegFile Setup	Mux	Adder
5ns	35ns	20ns	15ns	20ns

ALU	Branch Comp	Imm Gen	MEM Read	DMEM Setup
100ns	50ns	45ns	300ns	200ns

(a) Mark the stages of the datapath that the following instructions use:

	IF	ID	EX	MEM	WB
add	X	X	X		X
ori	X	X	X		X
lw	X	X	X	X	X
sw	X	X	X	X	
beq	X	X	X		
jal	X	X	X		X

(b) How long does it take to execute each instruction? Ignore the length of a clock cycle based off of the critical path, and assume that the setup times to the RegFile and the PC are the same.

1. jal

jal = clk-to-Q + Mem-Read + Imm-Gen + Mux(BSel) + ALU + Mux(PCSel) + PCSetup
= 5ns + 300ns + 45ns + 15ns + 100ns + 15ns + 20ns = 500ns

2. lw

lw = clk-to-Q + Mem-Read + max(RegFileRead + Mux(ASel), Imm-Gen + Mux(BSel)) + ALU + Mem-Read + Mux(WBSel) + RegFileSetup
= 5ns + 300ns + 60ns + 100ns + 300ns + 15ns + 20ns = 800ns

3. sw

sw = clk-to-Q + Mem-Read + max(RegFileRead + Mux(ASel), Imm-Gen + Mux(BSel)) + ALU + DMEM Setup
= 5ns + 300 ns + 60ns + 100ns + 200ns = 665ns

(c) Which instruction(s) exercise the critical path?

Load word (lw), which uses all 5 stages and takes 800ns.

(d) What is the fastest you could clock this single cycle datapath?

$$\frac{1}{800}\text{ nanoseconds} = \frac{1}{800 * 10^{-9}}\text{ seconds} = 1,250,000\text{s}^{-1} = 1.25\text{MHz}$$

(e) Why is the single cycle datapath inefficient?

At any given time, most of the parts of the single cycle datapath are sitting unused. Also, even though not every instruction exercises the critical path, the datapath can only be clocked as fast as the slowest instruction.

(f) How can you improve its performance? What is the purpose of pipelining?

Performance can be improved with pipelining, or putting registers between stages so that the amount of combinational logic between registers is reduced, allowing for a faster clock time.

4 Pipelining Performance

In order to pipeline, we separate the datapath into 5 discrete stages, each completing a different function and accessing different resources on the way to executing an entire instruction.

In the **IF** stage, we use the Program Counter to access our instruction as it is stored in IMEM. Then, we separate the distinct parts we need from the instruction bits in the **ID** stage and generate our immediate, the register values from the RegFile, and other control signals. Afterwards, using these values and signals, we complete the necessary ALU operations in the **EX** stage. Next, anything we do in regards with DMEM (not to be confused with RegFile or IMEM) is done in the **MEM** stage, before we hit the **WB** stage, where we write the computed value that we want back into the return register in the RegFile.

These 5 stages, divided by registers as shown in the figure, allow the datapath to provide a pipeline for multiple instructions to operate at the same time, each accessing different resources. A pipelined datapath is provided for you on the next page.

The **latency** of a CPU is the time it takes to execute one instruction, and the **throughput** of a CPU for a workload is the number of instructions executed per unit time. The goal of pipelining is to improve throughput (but does not improve latency).

Assume we are working with a CPU with the following execution times for the 5 stages:

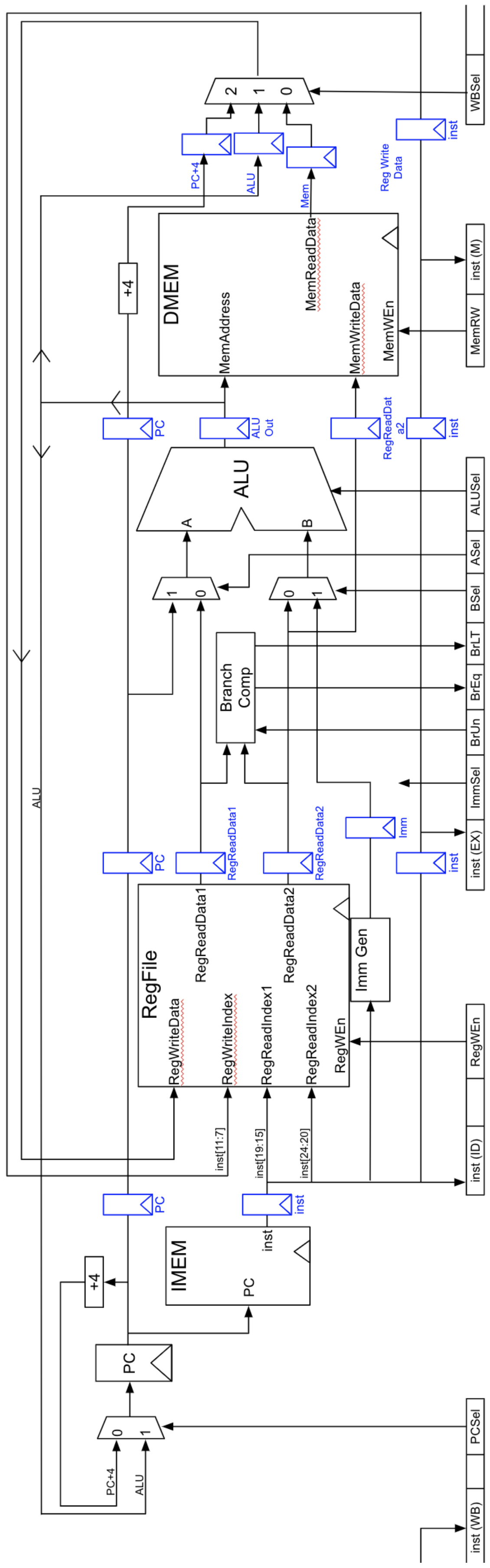
IF	ID	EX	MEM	WB
200 ps	150 ps	250 ps	300 ps	100 ps

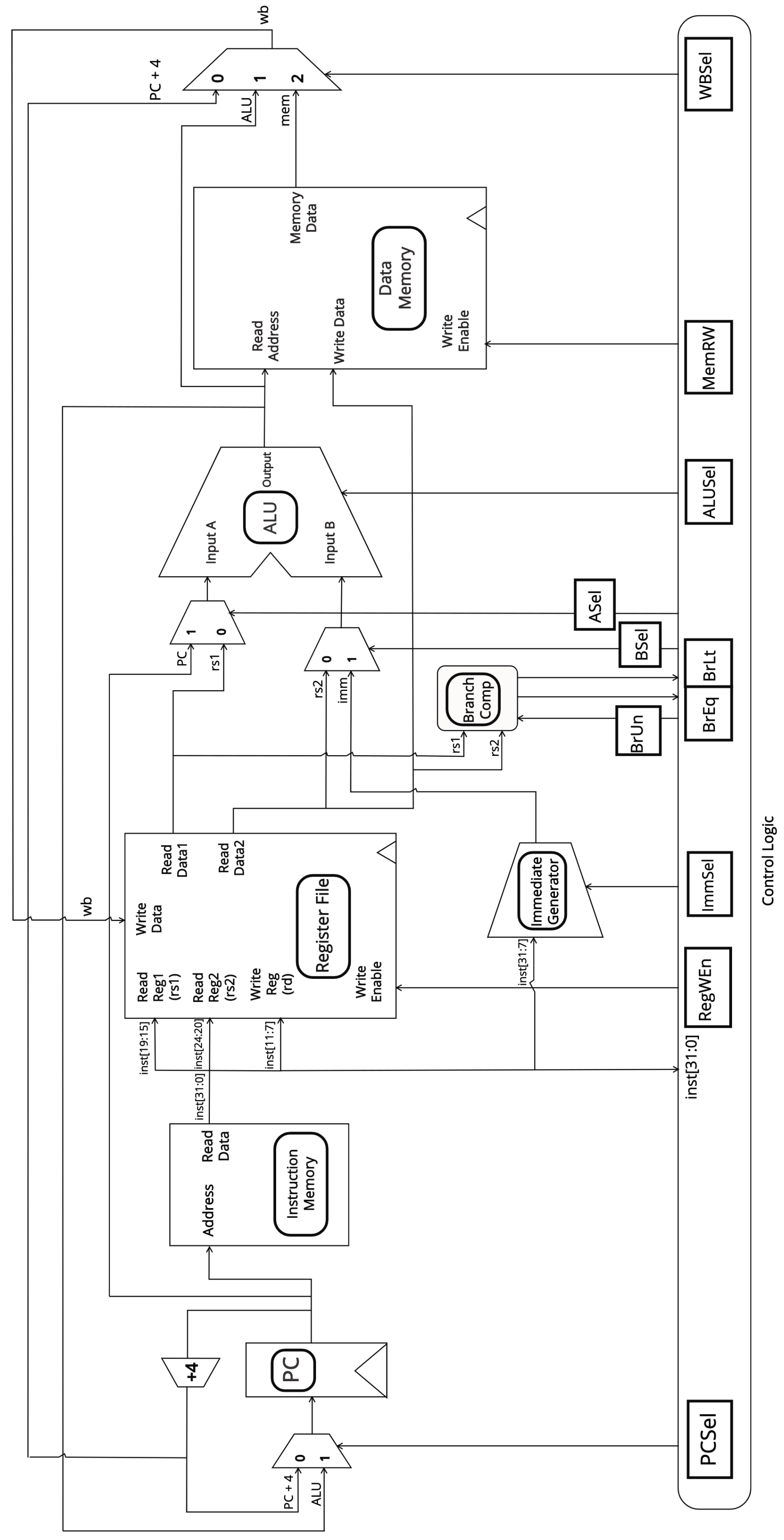
4.1 Given the execution times, what would the latency (in picoseconds) and throughput (in instructions/second) be for a single-cycle CPU, assuming it is clocked as fast as possible?

The cycle time is set by the critical path of the CPU. The critical path in the CPU utilizes all 5 stages, which means the cycle time is the time it takes to go through all 5 stages, or $200 + 150 + 250 + 300 + 100 = 1000$ ps. Since one instruction is executed in one cycle, the latency is 1000 ps. One instruction is completed every 1000 ps, or equivalently 10^{-9} s, so the throughput would be 10^{9} instructions per second.

4.2 Given the execution times, what would the latency (in picoseconds) and throughput in the long run (in instructions/second) be for a pipelined CPU using the 5-stage pipeline, assuming there no hazards?

The cycle time for a pipelined CPU is limited by the slowest stage. In this case, **MEM** is the slowest stage, so the cycle time must be 300 ps. It takes 5 cycles to complete one instruction, so the latency is $300 * 5 = 1500$ ps. In the long run, a pipelined CPU will complete one instruction per cycle, so our CPU will complete one instruction every 300 ps, or equivalently, $3 * 10^{-10}$ s. This means our throughput will be 1 / $(3 * 10^{-10}$ s) or roughly $3.33 * 10^{9}$ instructions per second.


PC
IMEM
PC
inst
+4
PC+4
ALU
0
1
PCSel
inst (WB)
PC
inst
inst[11:7]
inst[19:15]
inst[24:20]
RegFile
RegWriteData
RegWriteIndex
RegReadIndex1
RegReadIndex2
RegWEn
RegReadData1
RegReadData2
Imm Gen
inst (ID)
RegWEn
PC
RegReadData1
RegReadData2
Imm
inst
inst (EX)
ImmSel
Branch Comp
BrUn
BrEq
BrLT
BSel
ASel
ALUSel
ALU
A
B
PC
ALU Out
RegReadData2
inst
DMEM
MemAddress
MemReadData
MemWriteData
MemWEn
MemRW
inst (M)
+4
PC+4
ALU
Mem
Reg Write Data
inst
WBSel
2
1
0


wb
PC + 4
0
1
2
ALU
mem
WBSel
Memory Data
Data Memory
Read Address
Write Data
Write Enable
MemRW
Output
ALU
Input A
Input B
ALUSel
ASel
BSel
BrLt
BrEq
BrUn
Branch Comp
rs1
rs2
PC
imm
Read Data1
Read Data2
Write Data
Register File
Read Reg1 (rs1)
Read Reg2 (rs2)
Write Reg (rd)
Write Enable
Immediate Generator
ImmSel
RegWEn
inst[31:7]
inst[19:15]
inst[24:20]
inst[11:7]
inst[31:0]
Control Logic
Address
Read Data
Instruction Memory
+4
PC
PCSel
PC + 4
ALU