

# ECE 2300 Digital Logic and Computer Organization

## Topic 12: Pipelined Processors

<http://www.csl.cornell.edu/courses/ece2300>  
School of Electrical and Computer Engineering  
Cornell University

revision: 2025-12-10-17-25

### List of Problems

<b>1</b>	<b>Short Answers</b>	<b>3</b>
1.A	Pipelined Processor . . . . .	3
1.B	RAW Dependency . . . . .	3
1.C	Stalling in Fully Bypassed five-stage Pipelined Processor . . . . .	3
1.D	Sqash Operation . . . . .	4
1.E	Five-stage Pipelined Processor . . . . .	4
<b>2</b>	<b>Two-Stage Pipelined Processor</b>	<b>5</b>
2.A	Program 1: Pythagorean Theorem . . . . .	5
2.A.1	Control Dependencies . . . . .	6
2.A.2	Data Dependencies . . . . .	6
2.A.3	Fix 1: Software Scheduling . . . . .	6
2.A.4	Fix 2: Hardware Stalling . . . . .	8
2.A.5	Fix 3: Hardware Bypassing . . . . .	9
2.B	Program 2: Factorial Function . . . . .	10
2.B.1	Control Dependencies . . . . .	10
2.B.2	Data Dependencies . . . . .	10
2.B.3	Fix 1: Software Scheduling . . . . .	11
2.B.4	Fix 2: Hardware Stalling . . . . .	13
2.B.5	Fix 3: Hardware Bypassing . . . . .	14
<b>3</b>	<b>Five-Stage Pipelined Processor</b>	<b>15</b>
3.A	Program 1: Pythagorean Theorem . . . . .	16
3.A.1	Hardware Stalling . . . . .	16

3.A.2	Hardware Bypassing . . . . .	16
3.B	Program 2: Factorial Function . . . . .	18
3.B.1	Hardware Stalling . . . . .	18
3.B.2	Hardware Bypassing . . . . .	19
3.C	Program 3: Array Reversal . . . . .	21
3.C.1	Hardware Stalling . . . . .	22
3.C.2	Hardware Bypassing . . . . .	23

**Problem 1. Short Answers****Part 1.A Pipelined Processor**

Explain the general idea of pipelined processors? What are their advantages in comparison to single cycle or multi-cycle processors. What disadvantages do pipelined processors have?

---

---

---

---

---

---

---

---

**Part 1.B RAW Dependency**

What is a RAW dependency? Provide a code example and describe potential issues when being executed in a pipelined processor. What are potential fixes?

---

---

---

---

---

**Part 1.C Stalling in Fully Bypassed five-stage Pipelined Processor**

Can stalling occur in our canonical fully bypassed five-stage processor? What instruction sequence would trigger stalling.

---

---

---

---

---

**Part 1.D Squash Operation**

**Explain the squash operation: When and why is it performed? What does it do?**

---

---

---

---

---

**Part 1.E Five-stage Pipelined Processor**

**Name and describe the five pipeline stages in our canonical five-stage pipelined processor.**

---

---

---

---

---

---

---

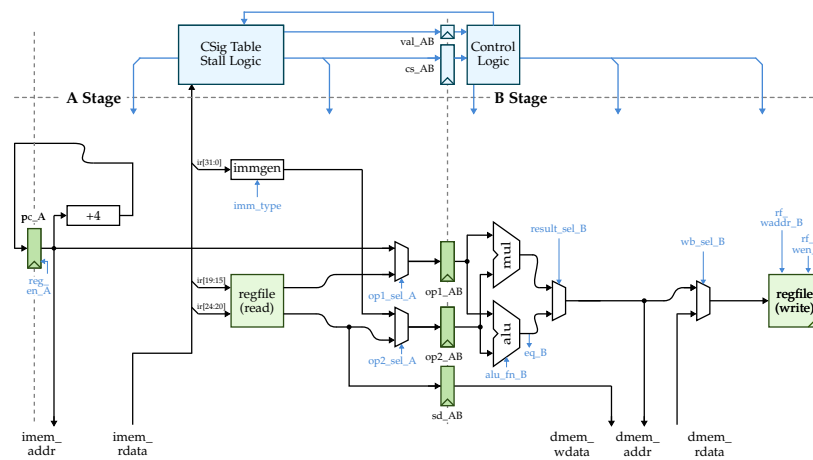
---

---

## Problem 2. Two-Stage Pipelined Processor

In this topic, we are exploring pipelined processors, beginning with the two-stage pipelined processor design from lecture. We will run two programs from the previous practice problems: the Pythagorean theorem and factorial function programs. You will need to identify the critical control and data dependencies within these programs. Afterwards, we will resolve these dependencies using software scheduling, hardware stalling, and hardware bypassing.

Below is the two-stage pipelined processor with hardware-stalling support.



### Part 2.A Program 1: Pythagorean Theorem

Program 1 computes the length of the hypotenuse of a right triangle (using integers, not floating-point numbers) via the Pythagorean theorem (see equation below). An IO-mapped accelerator for computing the square root function is connected to the multi-cycle processor. The accelerator reads from out0 (address 528), requires one cycle to compute (during which the processor must busy-wait), and then writes the integer square root (rounded down) to in0 (address 512).

$$C = \sqrt{A^2 + B^2}$$

```

1 sw    x0, 528(x0)
2 lw     x5, 256(x0)
3 lw     x6, 260(x0)
4 mul    x5, x5, x5
5 mul    x6, x6, x6
6 add    x5, x5, x6
7 sw     x5, 528(x0)
8 addi   x0, x0, 0    # wait sqrt comp
9 lw     x5, 512(x0)
10 sw    x5, 256(x0)

```

### 2.A.1 Control Dependencies

**Does this program contain any control dependencies?**

### 2.A.2 Data Dependencies

Mark all RAW data dependencies in the assembly code. Which dependencies would be hazards if not fixed for our two-stage pipelined-processor?

---

---

---

---

---

### 2.A.3 Fix 1: Software Scheduling

**Fix the RAW data dependencies with software scheduling. Write the resulting TinyRV1 assembly code below. Hint: Insert *nop* instructions to prevent data hazards.**

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.









**Part 2.B Program 2: Factorial Function**

Next, we will compute the factorials (see equation below) for the input stored in in0 at address 512. When complete, this program will store the result in out0 at address 528. **Inspect and understand the TinyRV1 code below.**

$$n! = \prod_{k=1}^n k = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$$

```

1  addi x5, x0, 1
2  lw   x6, 512(x0)    # read in0
3  addi x6, x6, 1      # stop = in0+1
4  addi x7, x0, 1      # fact = 1
5  bne  x6, x5, loop
6  jal  x0, end        # catch 0!
7  loop:
8  mul  x7, x5, x7      # fact = fact*i
9  addi x5, x5, 1
10 bne  x5, x6, loop
11 end:
12 sw   x7, 528(x0)    # out0 = fact

```

**2.B.1 Control Dependencies**

What control dependencies exist in the TinyRV1 assembly code above? Why is the jal instruction no issue, but the bne instruction?

---



---



---



---



---

**2.B.2 Data Dependencies**

Mark all RAW data dependencies in the assembly code. Which dependencies would be hazards if not fixed for our two-stage pipelined-processor?

---



---



---



---



---





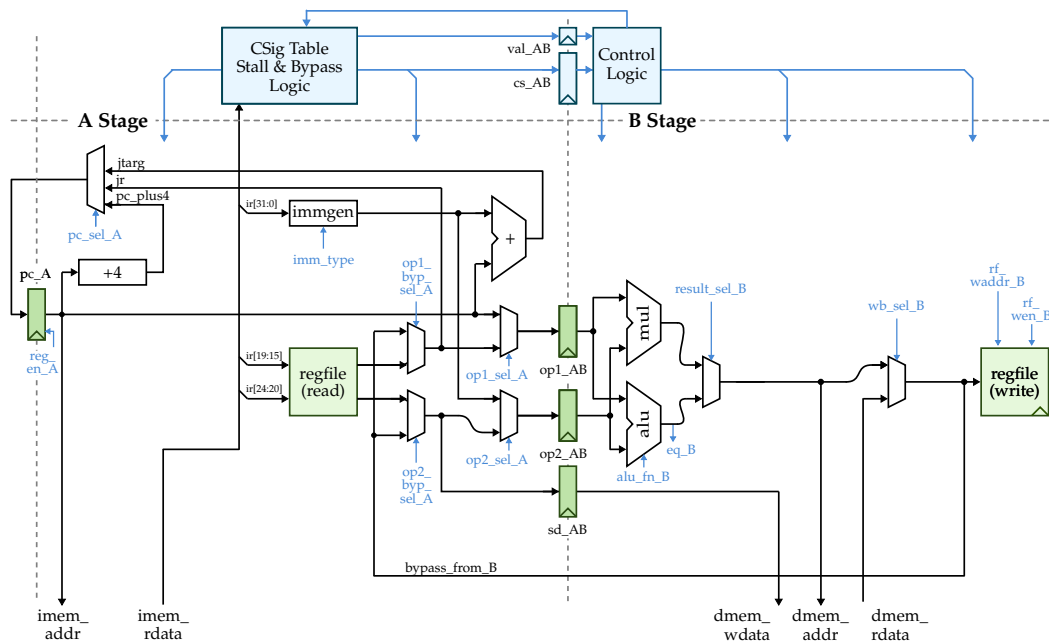




### Problem 3. Five-Stage Pipelined Processor

Next, we are continuing with the five-stage pipelined processor design from lecture. We will explore both using hardware stalling and hardware bypassing to dissolve RAW data and control dependency issues.

Below the canonical fully-bypassed five-stage pipelined processor.



**Part 3.A Program 1: Pythagorean Theorem****3.A.1 Hardware Stalling**

Complete the pipeline diagram of Program 1 (Section 2.A) when executing on a five-stage pipelined processor with hardware stalling support. Include microarchitectural dependency arrows.

sw	x0, 528(x0)																	
lw	x5, 256(x0)																	
lw	x6, 260(x0)																	
mul	x5, x5, x5																	
mul	x6, x6, x6																	
add	x5, x5, x6																	
sw	x5, 528(x0)																	
addi	x0, x0, 0																	

lw	x5, 512(x0)																	
sw	x5, 256(x0)																	

Determine the number of needed cycles.

---

**3.A.2 Hardware Bypassing**

Next, complete the pipeline diagram of Program 1 (Section 2.A) when executing on the canonical five-stage fully-bypassed pipelined processor. Include microarchitectural dependency arrows.

sw	x0, 528(x0)																	
lw	x5, 256(x0)																	
lw	x6, 260(x0)																	
mul	x5, x5, x5																	
mul	x6, x6, x6																	
add	x5, x5, x6																	
sw	x5, 528(x0)																	
addi	x0, x0, 0																	
lw	x5, 512(x0)																	
sw	x5, 256(x0)																	



Determine the number of needed cycles.

---

Complete the pipeline diagram of Program 1 (Section 2.A) when executing on a five-stage pipelined processor. However, this processor is not fully-bypassed due to design decisions. Instead it only has bypasses from the memory and write-back stage to decode (no bypass from execute to decode). Include microarchitectural dependency arrows.

sw	x0, 528(x0)																		
lw	x5, 256(x0)																		
lw	x6, 260(x0)																		
mul	x5, x5, x5																		
mul	x6, x6, x6																		
add	x5, x5, x6																		
sw	x5, 528(x0)																		
addi	x0, x0, 0																		
lw	x5, 512(x0)																		
sw	x5, 256(x0)																		

Determine the number of needed cycles.

---

**Part 3.B Program 2: Factorial Function**

In this section, we investigate the factorial function (Program 2 from Section 2.B). Since computing the factorial of large numbers requires many loop iterations, we focus exclusively on the main loop and omit the pre- and post-processing steps from our analysis.

```

1 loop:
2 mul  x7, x5, x7    # fact = fact*i
3 addi x5, x5, 1
4 bne  x5, x6, loop

```

**3.B.1 Hardware Stalling**

We will start our analysis with a five-stage pipeline processor with hardware stalling and speculative execution of the next instruction.

**Complete the pipeline diagram. Include RAW data and control microarchitectural dependency arrows.** *Hint: Data dependencies can reach from one loop to another.*

mul x7, x5, x7																			
addi x5, x5, 1																			
bne x5, x6, loop																			
opA																			
opB																			
mul x7, x5, x7																			

**How many cycles are required per loop? What is the average number of cycles per instruction?**

---



---



---

**Determine the run time for computing factorial 22.** *Note: Assume the clock period to be  $147\tau$  for this processor. Do not consider pre- and post-processing.*

---



---



---



---

Next, we will execute the same code again with hardware stalling. However, this time the processor has an advanced branch predictor and always correctly predicts the branch operation.

**Complete the pipeline diagram. Include both RAW data and control microarchitectural dependency arrows.**

*Note: The first `mul` instruction does not stall. However, in subsequent iterations, the `mul` instruction stalls due to the preceding `bne` stall. Since capturing this stalling behavior is critical for our analysis, we recommend measuring the loop duration from one `addi` instruction to the next `addi` instruction (i.e., across one complete iteration).*


**How many cycles are required per loop? What is the average number of cycles per instruction?**

---



---



---

### 3.B.2 Hardware Bypassing

In this pipeline diagram we execute the loop on our canonical fully-bypassed five-stage pipelined processor. Unfortunately, it includes again the "dumb" branch predictor, which always predicts the following instruction in the binary.

**Complete the pipeline diagram. Include both RAW data and control microarchitectural dependency arrows.**


**How many cycles are required per loop? What is the average number of cycles per instruction?**

---



---



---

Lastly, we enabled loop unrolling in our compiler. Thus, the compiler combined two loop iterations into a single *extended-loop*. See code below. Assume the input factorial to be even.

```

1 extended-loop:
2   mul  x7, x5, x7
3   addi x5, x5, 1
4   mul  x7, x5, x7
5   addi x5, x5, 1
6   bne  x5, x6, extended-loop

```

**Complete the pipeline diagram. Include both RAW data and control microarchitectural dependency arrows.**


**How many cycles are required per extended-loop? What would be the corresponding value per loop without unrolling? What is the average number of cycles per instruction?**

---



---



---

**Determine the run time for computing factorial 22.** Note: Assume the clock period to be  $159\tau$  for this processor. Do not consider pre- and post-processing.

---



---



---



---

**Part 3.C Program 3: Array Reversal**

In this assignment, we will investigate executing the array reversal program (already familiar from the practice problems of Topics 10 and 11) on five-stage pipelined processors with hardware stalling as well as hardware bypassing support. The pointers to the first and last elements are initially stored in the x10 and x11 registers, respectively. However, we move them into registers x5 and x6 before starting the main loop.

**Inspect and understand the TinyRV1 code below.**

```
1  addi x5, x10, 0      # ptr front
2  addi x6, x11, 0      # ptr back
3  loop:
4  lw   x10, 0(x5)      # swap values
5  lw   x11, 0(x6)
6  sw   x11, 0(x5)
7  sw   x10, 0(x6)
8  addi x5, x5, 4       # incr ptr front
9  bne  x5, x6, check2  # check if ptrs equal
10 jal  x0, done
11 check2:
12 addi x6, x6, -4      # decr ptr back
13 bne  x5, x6, loop
14 done:
15 addi x10, x0, 0      # return success
16 addi x0, x0, 0
```

We will assess the performance when reversing an array of 2049 32-bit values. Executing the entire program in our pipeline diagram is infeasible. **What code segment should we execute to still enable us to assess the overall performance?**

---

---

---

---

---

**3.C.1 Hardware Stalling**

We will start our analysis with a five-stage pipeline processor with hardware stalling and speculative execution of the next instruction (pc+4).

**Complete the pipeline diagram. Include RAW data and control microarchitectural dependency arrows.**



**How many cycles are required per loop? What is the average number of cycles per instruction?**

---



---



---



**How many cycles are required per loop? What is the average number of cycles per instruction?**

---

---

---

**Determine the run time for reversing the array with 2049 elements.** *Note: Assume the clock period to be  $159\tau$  for this processor.*

---

---

---

---

*The ECE 2300 TAs wish you the best of luck on the final. You've got this!*