Question: Single-cycle CPU implementation (40 points)

On the next page is a single-cycle datapath for a machine **very different** than the one we saw in lecture. It supports the following (complex) instructions:

```
lw_add   rd, (rs), rt              # rd = Memory[R[rs]] + R[rt];
addi_st  (rs), rs, imm             # Memory[R[rs]] = R[rs] + imm;
sll_add  rd, rs, rt, imm           # rd = (R[rs] << imm) + R[rt];
```

All instructions use the same format (shown below), but not all instructions use all of the fields.

Field	op	rs	rt	rd	imm
Bits	31-26	25-21	20-16	15-11	10-0

Part (a)
For each of the above instructions, specify how the control signals should be set for correct operation. Use **X** for **don't care**. ALUOp can be **ADD**, **SUB**, **SLL**, **PASS_A**, or **PASS_B** (*e.g.,* PASS_A means pass through the top operand without change). Full points will only be awarded for the fastest implementation. (20 points)

inst	ALUsrc1	ALUsrc2	ALUsrc3	ALUop1	ALUop2	MemRead	MemWrite	RegWrite
lw_add								
addi_st								
sll_add								

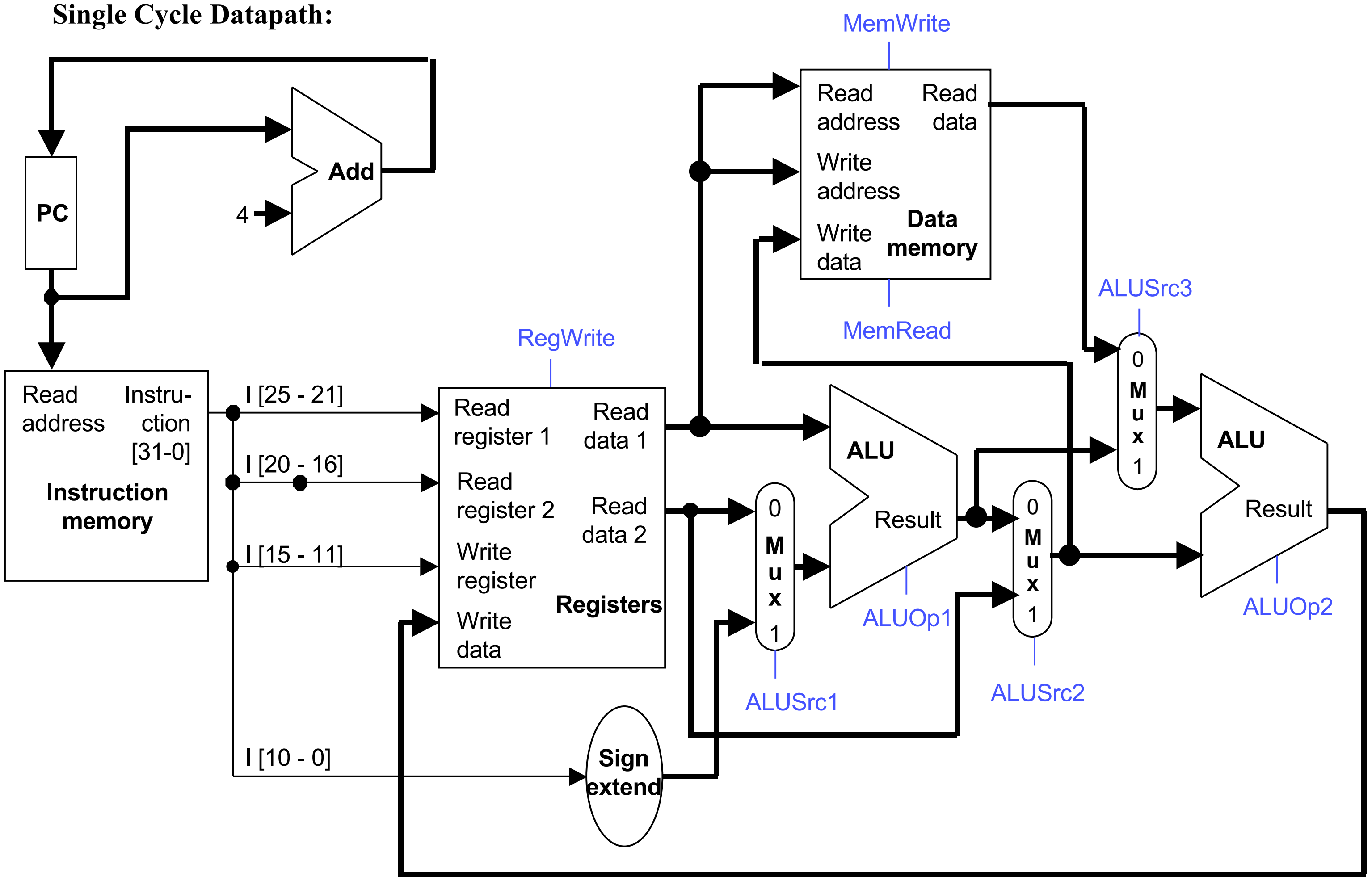
Part (b)
Given the functional unit latencies as shown to the right, compute the minimum time to perform each type of instruction. Explain. (15 points)

Func. Unit	**Latency**
Memory	3 ns
ALU	4 ns
Register File	2 ns

inst	Minimum time	Explain
lw_add		
addi_st		
sll_add		

Part (c)
What is the CPI and cycle time for this processor? (5 points)

Single Cycle Datapath:



Performance

1. Formula for computing the CPU time of a program P running on a machine X:

$$CPU\ time_{X,P} = Number\ of\ instructions\ executed_{P}\ x\ CPI_{X,P}\ x\ Clock\ cycle\ time_{X}$$

2. CPI is the average number of clock cycles per instruction:

$$CPI = Number\ of\ cycles\ needed\ /\ Number\ of\ instructions\ executed$$

3. Speedup is a metric for relative performance of 2 executions:

$$\begin{aligned} Speedup &= Performance\ after\ improvement\ /\ Performance\ before\ improvement \\ &= Execution\ time\ before\ improvement\ /\ Execution\ time\ after\ improvement \end{aligned}$$

Question: Pipelining (45 points)

Part (a)
Give a non-computing example of pipelining not involving laundry. (5 points)

Part (b)
Assume you have 10 items to process. If you can pipeline the processing into 5 steps (perfectly balancing the pipeline stages), how much faster does pipelining enable you to complete the process? You can leave your answer as an expression. (10 points)

Question, continued

Consider the following MIPS code:

```
loop: sub     $t3,    $t3,     $t0
      add     $t0,    $t0,     $t0
      addi    $t1,    $t0,     5
      lw      $t2,    0($t0)
      add     $t2,    $t2,     1
      sw      $t2,    0($t1)
      bgt     $t3,    $zero,   loop
```

Part (c) Label all dependences within one iteration of this code (not just the ones that will require forwarding). One iteration is defined as the instructions between the `sub` and `bgt` *inclusive.* (10 points)

Part (d) The above code produces the pipeline diagram shown below when run on a 5-stage MIPS pipeline with stages IF (fetch), ID (decode), EX (execute), MEM (memory) and WB (write-back).

Note: stalls are indicated by –, and registers can be read in the same cycle in which they are written.

Inst	iter	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
sub	N	IF	ID	EX	MEM	WB												
add	N		IF	ID	EX	MEM	WB											
addi	N			IF	ID	-	EX	MEM	WB									
lw	N				IF	-	ID	EX	MEM	WB								
add	N						IF	ID	-	EX	MEM	WB						
sw	N							IF	-	ID	EX	MEM	WB					
bgt	N								-	IF	ID	EX	MEM	WB				
sub	N+1										-	-	IF	ID	EX	MEM	WB	

For each of the following questions, justify your answer or explain why it cannot be answered using the given information. *Note: full credit requires explanations*. (20 points)

Is forwarding from EX/MEM to the EX stage implemented? Explain.

Is forwarding from MEM/WB to the EX stage implemented? Explain.

Note: The branch target is computed in the ID stage. In which stage are branches resolved? Explain.

The branch target is computed in the ID stage. What is the branch prediction scheme? Explain.

Question: Interrupt Handler Routine (25 points)

For the following questions, refer to the interrupt/exception handler code on the last page of this exam. Your answers should be no more that two sentences.

Part (a)

Explain the role of the statement labeled by **A**. Why is this necessary given that $at is not referenced elsewhere in the code? (5 points)

Part (b)

MIPS functions preserve callee-saved registers on the stack. In contrast, the interrupt handler saves registers in the .data segment (*e.g.,* in save0 and save1 in part B of the given code). Why is the stack **not** used in the interrupt handler? Also, why is the stack used to preserve registers in MIPS functions? (5 points)

Part (c)

Why does the line labeled with **C** jump to "interrupt dispatch" and not "done" ? (5 points)

Part (d)

Explain the line labeled by **D**. (5 points)

Part (e)

What code must be executed in order to reach the interrupt/exception handler? An explanation is sufficient; we don't need to see the exact code. (5 points)

Question 2, Interrupt Handler Routine

```
interrupt_handler:
    .set noat
    move     $k1, $at                  ← A
    .set at
    sw       $a0, save0
    sw       $a1, save1                ← B

    mfc0     $k0, $13                         # Get Cause register
    srl      $a0, $k0, 2
    and      $a0, $a0, 0xf                    # ExcCode field
    bne      $a0, 0, non_intrpt

interrupt_dispatch:
    mfc0     $k0, $13
    beq      $k0, $zero, done

    and      $a0, $k0, 0x1000
    bne      $a0, 0, bonk_interrupt

    li       $v0, 4
    la       $a0, unhandled_str
    syscall
    j        done

bonk_interrupt:
    …                                         # turn and set speed.
    sw       $a1, 0xffff0060($zero)           # acknowledge interrupt
    j        interrupt_dispatch  ← C

non_intrpt:
    li       $v0, 4
    la       $a0, non_intrpt_str
    syscall
    j        done

done:
    lw       $a0, save0
    lw       $a1, save1
    mfc0     $k0, $14          ← D
    .set noat
    move     $at $k1
    .set at
    rfe
    jr       $k0
    nop
```

Question 3: Concepts (25 points)

Write a short answer to the following questions. For full credit, answers should not be longer than **two sentences**.

Part a) A program is known to have a serial component that takes S seconds to execute, but the rest of the computation is arbitrarily parallelizable. If the program runs in T seconds using P processors, how long would it take to run using X processors? Assume T > S and P > 1. (10 points)

Part b) Consider the following program:

```
main(){
  func(5);
}
```

If we forgot to implement the function "func" would an error be raised by the compiler, assembler, linker, or loader? Explain. (5 points)

Part c)
In what circumstances is throughput the desired performance metric and in what circumstances is latency the desired metric? (5 points)

Part c)
Which of the three factors of CPU time can a compiler influence? Provide examples. (5 points)