CSE490/590, Summer 2025 Homework 4 (not graded) Solution

1.

(a) Schedule the following instruction sequence for dual-issue MIPS. Consider one ALU/branch instruction and one load/store instruction can be executed in parallel when there are no data dependencies:

```
Loop: lw $t0, 0($s1)       // $t0=array element
      add $t0, $t0, $s2    // add scalar in $s2
      sw $t0, 0($s1)       // store result
      addi $s1, $s1,-4     // decrement pointer
      add $s4, $s5, $s4    // update $s4
      bne $s1, $zero, Loop // branch when $s1!=0
```

	ALU/Branch	Load/Store	Cycle
Loop:	add $s4, $s5, $s4	lw $t0, 0($s1)	1
	addi $s1, $s1, -4	nop	2
	add $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4
			5
			6

a. Compute the IPC in part (a)

IPC = 6/4 = 1.5

2.

Assume a MIPS 5-stage pipelined processor **without** forwarding, show loop unrolling so that there are four copies of the loop body for the following instruction sequence. Schedule the instructions to avoid stalls.

```
Loop: lw $s0, 0($t0)
      addi $t2, $t2, $s0 // $t2 contains the sum of the array
      addi $t0, $t0, 4
      bne $t0, $t1, Loop
```

The above code iterates over an array of integers and computes its sum.

Assume that `$t0` contains the address of the first word, (`$t1 - 4`) is the address of last word, and that the difference between the addresses in `$t0` and `$t1` is a multiple of 16.

Eliminate any obviously redundant computations. Assume registers `$s0` to `$s7` and `$t3` to `$t9` are free to use for any purpose.

Fewer instructions solution (causes unnecessary stalls, still acceptable answer):

```
Loop:  lw $s0, 0($t0)
       addi $t2, $t2, $s0
```

```
lw $s0, 4($t0)
addi $t2, $t2, $s0
lw $s0, 8($t0)
addi $t2, $t2, $s0
lw $s0, 12($t0)
addi $t2, $t2, $s0
addi $t0, $t0, 16
bne $t0, $t1, Loop
```

Time-efficient solution (avoids stalls in loop body when possible):

```
      // Load all the values needed from memory
Loop: lw $s0, 0($t0) // lw instructions are bunched together so that
      lw $s1, 4($t0) // their corresponding addi instructions are
      lw $s2, 8($t0) // separated to avoid stalls
      lw $s3, 12($t0)
      addi $t0, $t0, 16 // addi needs to be at least 3 instructions
                        // before the bne to avoid a stall
      // Compute all the partial sums for this iteration
      addi $s4, $s4, $s0
      addi $s5, $s5, $s1 // Partial sum registers are needed to help
      addi $s6, $s6, $s2 // avoid stalls (each addi writes to a
      addi $s7, $s7, $s3 // different register)
      bne $t0, $t1, Loop
      // Compute the final sum from all the partial sums
      add $t3, $s4, $s5
      add $t4, $s6, $s7 // Last instruction stalls but that's ok
      add $t2, $t3, $t4 // since it is not part of the loop
```

3. For the code sequence shown below

```
loop:
    l.d $f12, 0($f5)
    add.d $f6, $f6, $f12
    daddui $f5, $f5, -8
    bne $f5, $f9, loop // $f9 holds the address of the last value to be operated on.
```

a) Show loop unrolling so that there are four copies of the loop body Assume $f5, $f9 (that is, the size of the array) are initially a multiple of 32, which means that the number of loop iterations is a multiple of 4. Eliminate any obviously redundant computations and do not reuse any of the registers.

```
l.d $f12, 0($f5)
add.d $f7, $f7, $f12
l.d $f13, -8($f5)
add.d $f8, $f8, $f13
l.d $f14, -16($f5)
add.d $f10, $f10, $f14
l.d $f15, -24($f5)
```

```
add.d $f11, $f11, $f15
daddui $f5, $f5, -32   ; -32 to account for 4 copies in the loop
bne $f5, $f9, loop
add.d $f16, $f7, $f8
add.d $f17, $f10, $f11
add.d $f18, $f16, $f17
```

b) Compute the number of cycles needed for 4 iterations

Instruction producing result	Instruction using result	Latency in cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

```
1. l.d $f12, 0($f5)
2. stall
3. add.d $f7, $f7, $f12
4. l.d $f13, -8($f5)
5. stall
6. add.d $f8, $f8, $f13
7. l.d $f14, -16($f5)
8. stall
9. add.d $f10, $f10, $f14
10. l.d $f15, -24($f5)
11. stall
12. add.d $f11, $f11, $f15
13. daddui $f5, $f5, -32  ;double add ALU with bne not given; assumed a 1 cycle latency
14. stall
15. bne $f5, $f9, loop
16. add.d $f16, $f7, $f8
17. add.d $f17, $f10, $f11
18. stall
19. stall
20. stall
21. add.d $f18, $f16, $f17
```

CSE490/590, Summer 2025 Homework 4 (not graded) Solution

4. Consider the following code sequence.

I1: lw $s4, 0($s1)
I2: or $s2, $s4, $s1
I3: and $s6, $s5, $s3

Highlight the Hazard and discuss how out of order processor will help when lw $s4, 0($s1) encounters a cache miss?

I1: lw $s4, 0($s1)
I2: or $s2, $s4, $s1
I3: and $s6, $s5, $s3
I3 can execute and wait for write back stage until the data is loaded in $s4 in I1 and eventually forwarded to I2

5. In the following instruction sequence, find the hazards. Rename the registers to eliminate the anti and output dependences

div.s r1,r2,r3
mult.s r4,r1, r5
add.s r1 ,r3, r6
sub.s r3,r1, r4

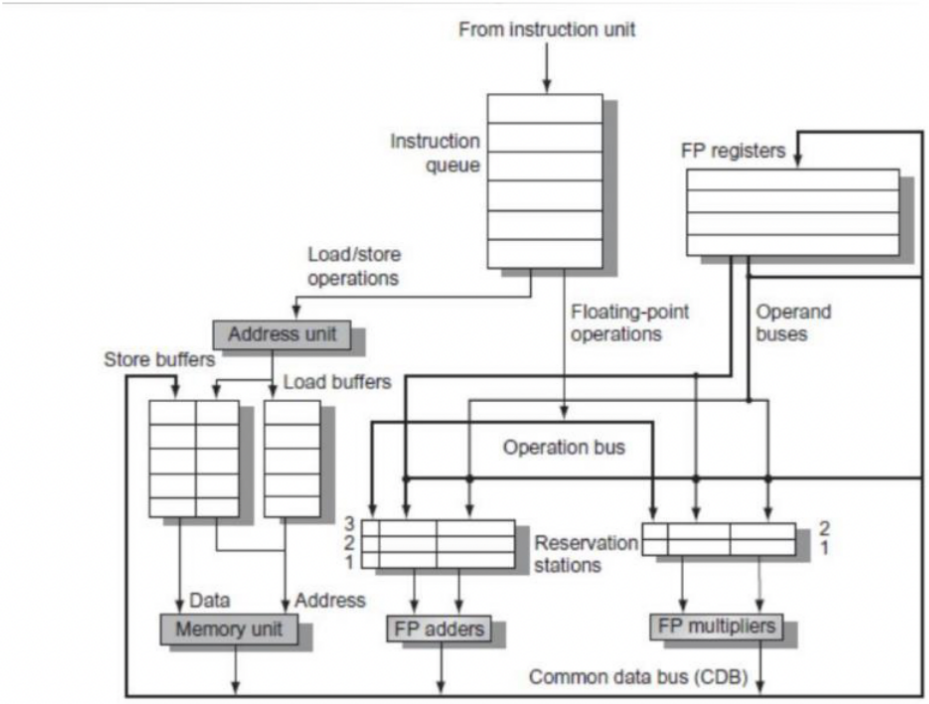
div.s r1,r2,r3
mult.s r4,r1, r5 // instr1 instr2 r1→ RAW (Data Dependence)
add.s r1 ,r3, r6 // instr1 instr3 r1 → WAW (Output Dependence)
sub.s r3,r1, r4 // instr2 instr4 r4 → RAW (Data Dependence), instr3 instr4 r3 → WAR
// instr1 instr4 r3 → Output Dependence (Can be WAR)

After Register Renaming:
div.s r1,r2,r3
mult.s r4,r1, r5
add.s r8 ,r3, r6
sub.s r9,r8, r4

CSE490/590, Summer 2025 Homework 4 (not graded) Solution

6. Consider the following instruction sequence (floating point) on a processor (shown below) which uses Tomasulo's algorithm to dynamically schedule instructions (dual issue per cycle - no speculation)

```
w:  ADD R4, R0, R8
x:  MUL R2, R0, R4
y:  ADD R4, R4, R8
z:  MUL R8, R4, R2
```



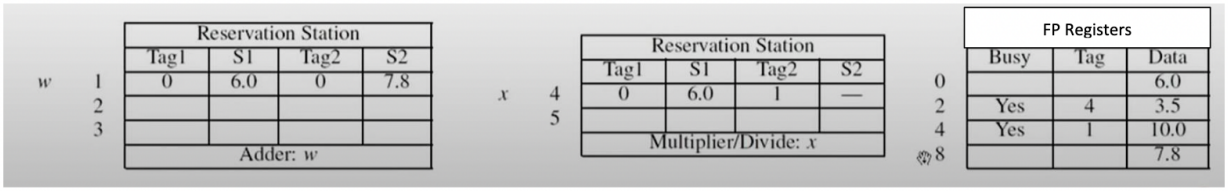
The processor has the following non-pipelined execution units:
A 2-cycle, FP add unit
A 3-cycle, FP multiply unit
Assume instructions can begin to execute in the same cycle as soon as its dispatched and resides in Reservation Stations
Trace the execution by showing Reservation Stations and FP Registers at the end of cycles# 1,2,3,5 and 6

Reservation Station of Multiplier/Divider is numbered as 4 and 5 as opposed to 1 and 2 as shown in the processor diagram above FP Registers
Busy -> Denotes if the operand is used in other operations or if it's ready
Tag -> Denotes the reservation station number that uses the operand
Reservation Stations
S1, S2 -> Denotes the source operands used in the instruction sequence
Tag1 -> 0 indicates the values is available and can be dispatched to the ALU unit if its free

CSE490/590, Summer 2025 Homework 4 (not graded) Solution

-> Any other number indicates if the value is dependent on the completion of the specific instruction in the reservation station
[Tag, tag1 and Busy bits are just added for explanation purposes. You can ignore those if you find them confusing]

Cycle# 1:

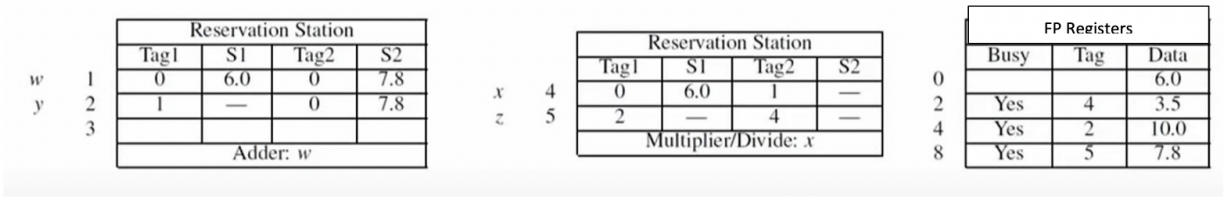


		Reservation Station			
		Tag1	S1	Tag2	S2
w	1	0	6.0	0	7.8
	2				
	3				
		Adder: w			

		Reservation Station			
		Tag1	S1	Tag2	S2
x	4	0	6.0	1	—
	5				
		Multiplier/Divide: x			

	FP Registers		
	Busy	Tag	Data
0			6.0
2	Yes	4	3.5
4	Yes	1	10.0
8			7.8

Cycle# 2:

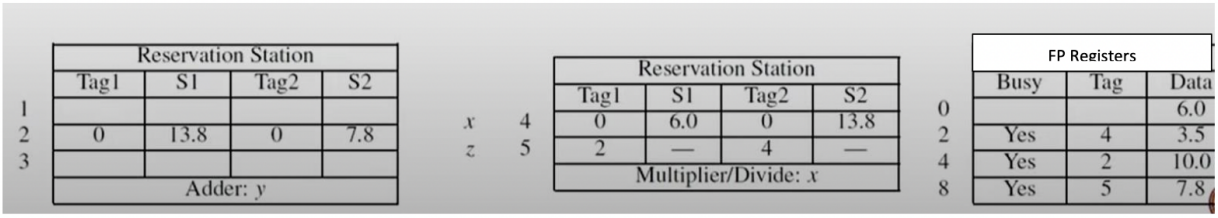


		Reservation Station			
		Tag1	S1	Tag2	S2
w	1	0	6.0	0	7.8
y	2	1	—	0	7.8
	3				
		Adder: w			

		Reservation Station			
		Tag1	S1	Tag2	S2
x	4	0	6.0	1	—
z	5	2	—	4	—
		Multiplier/Divide: x			

	FP Registers		
	Busy	Tag	Data
0			6.0
2	Yes	4	3.5
4	Yes	2	10.0
8	Yes	5	7.8

Cycle#3:

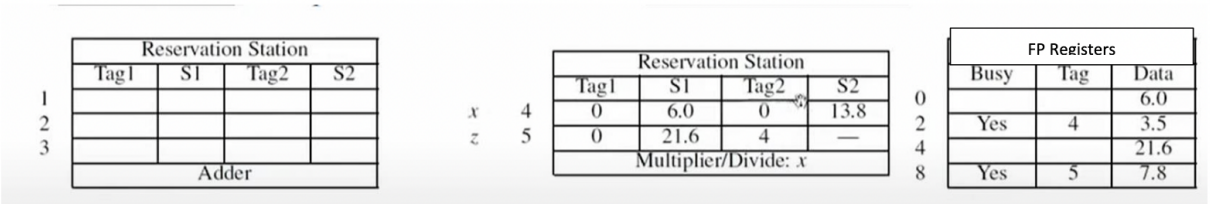


		Reservation Station			
		Tag1	S1	Tag2	S2
1					
2		0	13.8	0	7.8
3					
		Adder: y			

		Reservation Station			
		Tag1	S1	Tag2	S2
x	4	0	6.0	0	13.8
z	5	2	—	4	—
		Multiplier/Divide: x			

	FP Registers		
	Busy	Tag	Data
0			6.0
2	Yes	4	3.5
4	Yes	2	10.0
8	Yes	5	7.8

Cycle#5:



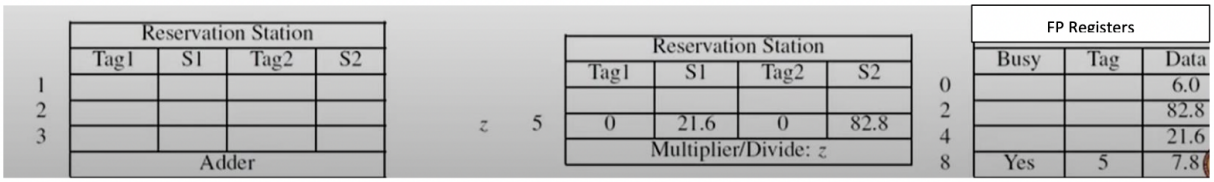
		Reservation Station			
		Tag1	S1	Tag2	S2
1					
2					
3					
		Adder			

		Reservation Station			
		Tag1	S1	Tag2	S2
x	4	0	6.0	0	13.8
z	5	0	21.6	4	—
		Multiplier/Divide: x			

	FP Registers		
	Busy	Tag	Data
0			6.0
2	Yes	4	3.5
4			21.6
8	Yes	5	7.8

Cycle#6:

CSE490/590, Summer 2025 Homework 4 (not graded) Solution



	Reservation Station			
	Tag1	S1	Tag2	S2
1				
2				
3				
	Adder			

z 5

	Reservation Station			
	Tag1	S1	Tag2	S2
	0	21.6	0	82.8
	Multiplier/Divide: z			

	FP Registers		
	Busy	Tag	Data
0			6.0
2			82.8
4			21.6
8	Yes	5	7.8

More details: Solution for 1-9 cycles: