6 Pipelining [65 points]

Consider two pipelined machines implementing the MIPS ISA, Machine A and Machine B. Both machines have *one ALU* and the following *five pipeline stages*, very similar to the basic 5-stage pipelined MIPS processor we discussed in lectures:

1. Fetch (one clock cycle)
2. Decode (one clock cycle)
3. Execute (one clock cycle)
4. Memory (one clock cycle)
5. Write-back (one clock cycle).

Machines A and B have the following specifications:

	Machine A	**Machine B**
Data Forwarding/Interlocking	Does **NOT** implement interlocking in hardware. Relies on the compiler to order instructions or insert `nop` instructions such that dependent instructions are correctly executed.	Implements data dependence detection and data forwarding in hardware. On detection of instruction dependence, it forwards an operand from the memory stage or from the write-back stage to the execute stage. The result of a load instruction (`lw`) can *only* be forwarded from the write-back stage.
Internal register file forwarding	Implemented (i.e., an instruction writes into a register in the first half of a cycle and another instruction can correctly access the same register in the second half of the cycle).	Same as Machine A
Branch Prediction	Predicts all branches as *always-taken*, and the next program counter is available after the decode stage.	Same as Machine A

Consider the following code segment:

```
Loop: lw   $1, 0($4)
      lw   $2, 400($4)
      add  $3, $1, $2
      sw   $3, 0($4)
      sub  $4, $4, #4
      bnez $4, Loop
```

Initially, `$1 = 0, $2 = 0, $3 = 0, and $4 = 400`.

(a) [15 points] Re-write the code segment above *with minimal changes* so that it gets correctly executed in Machine A *with minimal latency*. You can either insert `nop` instructions or reorder instructions as needed.

```
Loop:  lw $1, 0($4)
       lw $2, 400($4)
       nop
       nop
       add $3, $1, $2
       nop
       nop
       sw $3, 0($4)
       sub $4, $4, #4
       nop
       nop
       bnez $4, Loop
```

(b) [15 points] Fill the table below with the timeline of the first loop iteration of the code segment in Machine A.

Instruction	**Clock cycle number**																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
`lw $1, 0($4)`	F	D	E	M	W														
`lw $2, 400($4)`		F	D	E	M	W													
`nop`			F	D	E	M	W												
`nop`				F	D	E	M	W											
`add $3, $1, $2`					F	D	E	M	W										
`nop`						F	D	E	M	W									
`nop`							F	D	E	M	W								
`sw $3, 0($4)`								F	D	E	M	W							
`sub $4, $4, #4`									F	D	E	M	W						
`nop`										F	D	E	M	W					
`nop`											F	D	E	M	W				
`bnez $4, Loop`												F	D	E	M	W			

(c) [10 points] Calculate the number of cycles it takes to execute the code segment on Machine A. Show your work in the box.

Total number of cycles: 1303.

Explanation:
The compiler reorders instructions and places six `nop`-s.
This is the execution timeline of the first iteration:

Each iteration consists of 12 instructions. Since the next program counter is available after the decode stage of bnez, the next iteration starts with an additional delay of 1 cycle.
The last iteration takes 16 cycles, to drain the pipeline.
Thus the entire program runs for 99 * 13 + 16 = 1303 cycles.

(d) [15 points] Fill the table below with the timeline of the first loop iteration of the code segment in Machine B.

Instruction	**Clock cycle number**																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
`lw $1, 0($4)`	F	D	E	M	W														
`lw $2, 400($4)`		F	D	E	M	W													
`add $3, $1, $2`			F	D	*	E	M	W											
`sw $3, 0($4)`				F	*	D	E	M	W										
`sub $4, $4, #4`						F	D	E	M	W									
`bnez $4, Loop`							F	D	E	M	W								
`lw $1, 0($4)`								*	F	D	E	M	W						

(e) [10 points] Calculate the number of cycles it takes to execute the code segment on Machine B. Show your work in the box.

Total number of cycles: 803.

Explanation:
1 - Foward $2 from W to E in cycle 6.
2 - Foward $3 from M to E in cycle 7.
3 - Foward $4 from M to E in cycle 9.

Each iteration takes 8 cycles, including one cycle delay after bnez, because to the next program counter is available only after the decode stage of bnez.
The last iteration takes 11 cycles, to drain the pipeline.
Thus total number of cycles is 99*8 + 11 = 803 cycles.