1.)

A.]

```
int x2;          //Initialize x2 to be loaded to x1
x3 = x2 + 396    //Assume the initial value of x3 is x2 + 396

while(x4 != 0) {      //Loop:
      x1 = x2;        //load x1 from address 0+x2
      x1 += 1;        //x1 = x1 + 1
      x2 = x1;        //store x1 at address 0+x2
      x2 += 4;        //x2 = x2 +4
      x4 = x3 - x2    //x4 = x3 - x2
}                     //Loop will continue while x4 != 0
```

B.]

Instruction	Clock Cycles																				
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Loop: lw x1, 0(x2)		IF	ID	EX	M	WB															
addiw x1, x1, 1			IF	*S*	*S*	ID	EX	M	WB												
sw x1, 0(x2)						IF	*S*	*S*	ID	EX	M	WB									
addiw x2, x2, 4									IF	ID	EX	M	WB								
subw x4, x3, x2										IF	*S*	*S*	ID	EX	M	WB					
bne x4, x0, -24													IF	*S*	*S*	ID	EX	M	WB		
lw x1, 0(x2)																IF	no-op				

C.]

Cell #1 → Cell #16 = 15 cycles

For every recycled loop (where x4 != 0) pc is set at ID, so when EX is run the loop resets until x4 = 0

Assuming that the initial value of x3 is x2 + 396, and the loop starts executing from the first instruction, the first time through the loop x4 will be equal to (x2+396) - x2 = 396. On each subsequent iteration of the loop, the value of x2 is incremented by 4, and therefore the value of x4 is decremented by 4. The loop will continue to execute until x4 becomes equal to 0. Therefore, the loop will run 396/4 = 99 times.

[(15 cycles)*(98 loops) + (18 cycles)*(1 final loop)] = **1488 cycles**

D.]

Instruction	Clock Cycles																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Loop: lw x1, 0(x2)		IF	ID	EX	M	WB															
addiw x1, x1, 1			IF	*S*	ID	EX	M	WB													
sw x1, 0(x2)					IF	ID	EX	M	WB												
addiw x2, x2, 4						IF	ID	EX	M	WB											
subw x4, x3, x2							IF	ID	EX	M	WB										
bne x4, x0, -24								IF	ID	EX	M	WB									
Loop++									IF	ID	EX	M	WB								

E.]

[(98 loops)*(8 cycles)] + (11 cycles on the first iteration) = **795 cycles**

F.]

Instruction	Clock Cycles																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Loop: lw x1, 0(x2)		IF	ID	EX	M	WB															
addiw x1, x1, 1			IF	*S*	ID	EX	M	WB													
sw x1, 0(x2)					IF	ID	EX	M	WB												
addiw x2, x2, 4						IF	ID	EX	M	WB											
subw x4, x3, x2							IF	ID	EX	M	WB										
bne x4, x0, -24								IF	ID	EX	M	WB									
Delay Branch Slot																					
Loop++										IF	ID	EX	M	WB							

G.]

```
lw          x1,0(x2)    ; load X1 from address 0+x2
Loop:
addiw       x1,x1,1     ; x1 = x1 + 1
sw          x1,0(x2)    ; store x1 at address 0+x2
addiw       x2,x2,4     ; x2 = x2 +4
subw        x4,x3,x2    ; x4 = x3 - x2
bne         x4,xo-24    ; branch to loop if x4!= 0
lw          x1,0(x2)    ; load X1 from address 0+x2
```

H.]

Instruction	Clock Cycles																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw x1, 0(x2)		IF	ID	EX	M	WB															
Loop: addiw x1, x1, 1			IF	*S*	ID	EX	M	WB													
sw x1, 0(x2)					IF	ID	EX	M	WB												
addiw x2, x2, 4						IF	ID	EX	M	WB											
subw x4, x3, x2							IF	ID	EX	M	WB										
bne x4, x0, -24								IF	ID	EX	M	WB									
lw x1, 0(x2)									IF	ID	EX	M	WB								

I.]
Yes, the compiler can assume x1 loads before executing addiw so execute sw before add

```
lw    x1, 0(x2)  ; load X1 from address 0+x2
loop:
sw    x1, 0(x2)  ; store x1 at address 0+x2
addiw x1, x1, 1  ; x1 = x1 + 1
addiw x2, x2, 4  ; x2 = x2 +4
subw  x4, x3, x2 ; x4 = x3 - x2
bne   x4, xo-24  ; branch to loop if x4!= 0
lw    x1, 0(x2)  ; load X1 from address 0+x2
```

J.]

Instruction	Clock Cycles																			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw x1, 0(x2)		IF	ID	EX	M	WB															
Loop: sw x1, 0(x2)			IF	ID	EX	M	WB														
addiw x1, x1, 1				IF	ID	EX	M	WB													
addiw x2, x2, 4					IF	ID	EX	M	WB												
subw x4, x3, x2						IF	ID	EX	M	WB											
bne x4, loop							IF	ID	EX	M	WB										
lw x1, 0(x2)								IF	ID	EX	M	WB									

K.]
[(98 loops)*(11 cycles per loop)] + (11 cycles in the last iteration) = **697 cycles**

L.]
1448/697 = **2.07**

2.)

A.]
The 5-stage pipeline does not contain any register delay; the MEM stage requires the greatest amount of time when called.

Clock cycle time will equal the greatest delay pertaining to all of the stages, which is **2 ns** in this case.

B.]
4 instructions → delay in a 5-stage pipeline = 5/4 = **1.25 cycles per instruction**

C.]
Execution time without pipelining = (1 + 1.5 + 1 + 2 + 1.5) = 7 ns
Execution time with pipelining (and stall) = (2 * 1.25) = 2.5 ns
Speedup = 7/2.5 = **2.8**

Therefore, the speedup of C470 over C360, assuming a stall every four instructions, is 2.8. This means that C470 can execute the same program 2.8 times faster than C360.

3.)

Instruction	Clock Cycles																					
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
`fld f1, 0(x1)`		IF	ID	EX	M	WB																	
`fld f2, 0(x2)`			IF	ID	EX	M	WB																
`fmult.d f3, f2, f1`					IF	*S*	ID	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	M	WB							
`fadd.d f3, f3, 1`							IF	*S*	*S*	*S*	*S*	*S*	*S*	*S*	ID	**A1**	**A2**	**A3**	**A4**	M	WB		
`addi x3, x3, 1`															IF	ID	EX	*S*	*S*	*S*	M	WB	

In Figure C.30, Integer uses 1 EX, FP/Integer uses 7 cycles, and FP adder uses 4 cycles.

4.)

C.7 [10/10]<C.3>In this problem, we will explore how deepening the pipeline affects performance in two ways: faster clock cycle and increased stalls due to data and control hazards. Assume that the original machine is a 5-stage pipeline with a 1 ns clock cycle. The second machine is a 12-stage pipeline with a 0.6 ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard every five instructions, whereas the 12-stage pipeline experiences three stalls every eight instructions. In addition, branches constitute 20% of the instructions, and the mis-prediction rate for both machines is 5%.

a. [10]<C.3>What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?

Execution time = N * CPI * Cycle time

Speedup = Execution time of 5 stage pipeline / Execution time of 12 stage
= (N * (6/5) * 1 ns) / (N * (11/8) * 0.6 ns) = **1.45**