Homework #3

By

Andrew Hutzel (915841776)

Q1 Exercise 4.1)

4.1 Consider the following instruction:
Instruction: AND Rd,Rs,Rt
Interpretation: Reg[Rd] = Reg[Rs] AND Reg[Rt]
4.1.1 [5] <§4.1> What are the values of control signals generated by the control in Figure 4.2 for the above instruction?
4.1.2 [5] <§4.1> Which resources (blocks) perform a useful function for this instruction?
4.1.3 [10] <§4.1> Which resources (blocks) produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

4.1.1 Answer)

Opcode	ExtSel	BSrc	OpSel	MemW	RegW	WBSrc	RegDst	PCSrc
ALU	*	Reg	Func	No	Yes	ALU	Rd	Pc+4

4.1.2 Answer)

The program counter, instruction memory, ALU, and register are the resources that perform the most useful functions for the instruction given.

4.1.3 Answer)

Branch is the resource that produces output, but their outputs are not used for this instruction. Data memory is the resource that doesn't produce output for this instruction.

Q2 4.3)

4.3 When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off . In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400 ps, 100 ps, 30 ps, 120 ps, 200 ps, 350 ps, and 100 ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively.
Consider the addition of a multiplier to the ALU. Th is addition will add 300 ps to the latency of the ALU and will add a cost of 600 to the ALU. Th e result will be 5% fewer instructions executed since we will no longer need to emulate the MUL instruction.
4.3.1 [10] <§4.1> What is the clock cycle time with and without this improvement?
4.3.2 [10] <§4.1> What is the speedup achieved by adding this improvement?
4.3.3 [10] <§4.1> Compare the cost/performance ratio with and without this improvement.

4.3.1 Answer)

Ps ➔ Pico seconds ?

	I-mem	Add	Mux	ALU	Regs	D-Mem	CTRL Blocks
Time (ps)	400	100	30	120	200	350	100
Costs	1000	30	10	100	200	2000	500

With this in mind we calculate WITHOUT MULTIPLEXER first:

Clock cycle = I-Mem + Regs + Mux +ALU+ D-Mem+Mux

=400 + 200 +30+120 +350+30 = 1130 ps

WITH MULTIPLEXER (MUL = 300 PS):

Clock cycle = I-mem +Regs + Mux + ALU +MUL + D-MEM + Mux +Regs

=400+200+ 30+ 120+300+350+30+200 = 1430 ps

4.3.2 Answer)

Speedup given by adding MUL :

$$\frac{new\ clock\ cycle\ time}{old\ clock\ cycle\ time} = \frac{1130 * 100}{95 * 1430} = 0.83\ ps \quad (slows\ down!)$$

4.3.3 Answer)

With MUL:

$$\frac{1000 + 200 + 10 + 10 + 100 + 2000 + 500 + 30 + 30 + 10}{1130} = 3.44$$

Without MUL:

$$\frac{1000 + 200 + 10 + 10 + 10 + 100 + 2000 + 500 + 30 + 30 + 600}{1430} = 3.14$$

4.5 For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

- **add addi not beq lw sw**
- 20% 20% 0% 25% 25% 10%

4.5.1 [10] <§4.3> In what fraction of all cycles is the data memory used?

4.5.2 [10] <§4.3> In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?

4.5.1 Answer)

We know that Data Memory is used in store word and load word. Data memory is only utilized in SW and LW, so we sum the two resulting in: 10%+25%=35%

4.5.2 Answer)

In the control table ExtSel, we know the control signal for the sign extend uses: ADDI, BEQ, LW, and SW. **What isn't used is NOT and ADD because all register values are used and there isn't a constant, or immediate in this instruction.** Again we sum these instructions:

20%+25%+25%+10%= 80%

This tells us that 80% of the cycles need sign-extended.

4.7 In this exercise we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word:
(Upper) 1010 1100 0110 0010 0000 0000 0001 0100. (lower)
Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

0 –1 2 –3 –4 10 6 8 2 –16

4.7.1 [5] <§4.4> What are the outputs of the sign-extend and the jump "Shift left 2" unit (near the top of Figure 4.24) for this instruction word?
4.7.2 [10] <§4.4> What are the values of the ALU control unit's inputs for this instruction?
4.7.3 [10] <§4.4> What is the new PC address aft er this instruction is executed? Highlight the path through which this value is determined.
4.7.4 [10] <§4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.
4.7.5 [10] <§4.4> For the ALU and the two add units, what are their data input values?
4.7.6 [10] <§4.4> What are the values of all inputs for the "Registers" unit?

To better understand the problem we must remember that we're slicing up a given instruction presumably 32 bits into a given diagram. All we need to do is check at certain bits in the given instruction to figure out how to parse the command.

4.7.1 Answer)

WE are given 1010 1100 0110 0010 0000 0000 0001 0100 and we extend using the LOWER bits resulting in this output for **sign-extend**:

0000 0000 0000 0000 0000 0000 0001 0100

Shift Left 2, takes least 26 significant bits and expands it to 28 bits by shifting two values. Lower 26 bits:

00 0110 0010 0000 0000 0001 0100

Shift by 2, giving us 28 bits output now…

0001 1000 1000 0000 0000 0101 0000

4.7.2 Answer)

So we know that the vales of ALUOp is 2 bits of the LEAST significant. So we use the given instruction word 1010 1100 0110 0010 0000 0000 0001 0100 so the **ALUOP = 00.**

Instruction is least significant 6 bits 1010 1100 0110 0010 0000 0000 0001 0100 so **instruction=01 0100**

ALUOP	**Instruction**
00	**01 0100**

4.7.3 Answer)

PC=PC+4, for the new PC address.

Path:

Program counter to ADD (PC+4), this will branch to MUX, and back to PC.

4.7.4 Answer)

Write register for the MUX:

2 0r 0, because we know the write registers aren't used. This will be stored to memory.

ALU MUX:

Immediate register (original instruction, but only using the last six bits). Sign extend:

0000 0000 0000 0000 0000 0000 0001 0100

We of course now have an ALU MIX value which is 20.

Branch and Jump MUX: This will be used to increment the PC addresses. So we have PC+4…

ALU/MEM MUX:

RegW MUX	ALU/Mem MUX	Branch MUX	Jump MUX	ALU MUX
2 or 0	X	PC + 4	PC+4	20

4.7.5 Answer)

ALU UNIT (Parsing this 32 bits: 1010 1100 0110 0010 0000 0000 0001 0100)

Input reads the first register [25-21]= 0 0011 = 3 and so it is register 3. Register 3 has a value of -3 and we know from the previous question the extended 16 bit sign has a value of 20.

ADD (Branch):

A single input is PC+4 and the 16 bit extended has a value of 20 before being shifted left by 2, so we can multiply the two values to get 20*4 = 80. (Note to self, its very interesting to see if we just shift the sign extended value, we will naturally get 80. I guess this I just a good way to double check my answer…)

ADD(PC):

Two inputs PC and 4.

ALU UNIT	ADD (Branch)	ADD (PC)
-3 and 20	PC+4 and 80	PC and 4

4.7.6 Answer)

Read Register 1: [25-21 bits, 00011] = 3

Read Register 2: [20-16, 00010]= 2

Write register and write data: Based off of the instruction we know that it is STORING, because of that there isn't a register write operation, values are ZERO.

Read Register 1	Read Register 2	Write Data	Write Register	Register Write
3	2	0	0 (2 or 0)	0

4.9 In this exercise, we examine how data dependences aff ect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

or r1,r2,r3
or r2,r1,r4
or r1,r1,r2

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding, With Full Forwarding, With ALU-ALU Forwarding Only
250ps 300ps 290ps

4.9.1 [10] <§4.5> Indicate dependences and their type.

4.9.2 [10] <§4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.

4.9.3 [10] <§4.5> Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.

4.9.4 [10] <§4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.9.5 [10] <§4.5> Add nop instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

4.9.6 [10] <§4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

4.9.1 Answer)

The first OR writes to r1 and the next two ORs will also being using r1. This means we will have a RAW (read after write) dependency on r1 from the first OR to the following two uses of ORs. Don't forget the WAR (write after reads) and WAW (write after write).

So The dependencies will look like this [Type, Dependencies]:

RAW ➔ r1 from instruction 1 to instruction 2 and instruction 3.

RAW ➔ r2 from instruction 2 to instruction 3.

WAR ➔ r2 from instruction 1 to instruction 2.

WAR ➔ r1 from instruction 2 to instruction 3.

WAW ➔ r1 from instruction 1 to instruction 3.

4.9.2 Answer)

Hazards and op instructions to eliminate problems from the first set of instruction will appear like so:

Or r1,r2,r3

NOP

NOP

Or r2,r1,r4

NOP
NOP

Or r1,r1,r2

4.9.3 Answer)

Knowing that will full forwarding, an ALU instruction can forward a value to the execution stage of the next instruction without a hazard. And so it will appear like this:

Or r1,r2,r3

Or r2,r1,r4 #There is not a RAW hazard from r1 to instruction 1

Or r1,r1,r2 #There is not a RAW hazard from r2 to instruction 2.

4.94 Answer)

Note: Without any interrupts, a three-instruction sequence executes in 7 cycles (5 to complete the first instruction, then one per instruction). Execution without forwarding must add a stall for every NOP we had in #2, and execution forwarding must add a stall cycle for every NOP we had in 3.

Total execution time = Clock cycle time * number of cycles

No forwarding	With forwarding	Speed-up
(7+4) * 250=2750 ps	7*300 = 2100 ps	Speedup=2750/2100=1.31

4.9.5 Answer)

ALU to ALU only forwarding, an ALU instruction can forward to the next instruction, but not to the second next instruction (because that would mean forwarding from MEM to EX). A load cannot forward at all, because it determines the data value in MEM stage, when it is too late for ALU-ALU forwarding

Instruction Sequence	
Lw $1,40($6)	Cannot use ALU-ALU forwarding ($1 is loaded into MEM)
Add $6, $2, $2	
Nop	
Sw $6, 50($1)	

4.9.6 Answer)

No forwarding	With ALU-ALU forwarding only	Speed-up with ALU-ALU forwarding
(7+1)x300=2400 ps	7x300=2100	2400/2100=1.14 (slow down)

4.11 Consider the following loop.

```
loop:lw r1,0(r1)
and r1,r1,r2
lw r1,0(r1)
lw r1,0(r1)
beq r1,r0,loop
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

4.11.1 [10] <§4.6> Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the fi rst instruction of that iteration up to (but not including) the cycle in which we can fetch the fi rst instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration).

4.11.2 [10] <§4.6> How often (as a percentage of all cycles) do we have a cycle in which all fi ve pipeline stages are doing useful work?

4.11.1 and 4.11.2 Answer~~, needs to be reviewed~~)

Execution has five stages IF, ID, EX, MEM, and WB. Each instruction will complete a stage, the previous instruction

4.11.1 Answer								
Lw r1, 0(r1)	WB							
Lw r1,0(r1)	EX	MEM	WB					
Beq r1,r0, loop	ID	Stall	EX	MEM	WB			
Lw r1,0(r1)	IF	STALL	ID	EX	MEM	WB		
And r1,r1,r2			IF	ID	STALL	EX	MEM	WB
Lw r1,0(r1)				IF	STALL	ID	EX	MEM
Lw r1,0(r1)						IF	ID	STALL
Beq r1,r0, loop							IF	STALL

So we know there will be 8 clock cycles per iteration I the loop and there isn't any clock cycles for all 5 stages of the pipeline are in use. So the percentage of clock cycles where all 5 pipeline stages are doing useful work is, 0%.

4.13 Th is exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

```
add r5,r2,r1
lw r3,4(r5)
lw r2,0(r2)
or r3,r5,r3
sw r3,0(r5)
```

4.13.1 [5] <§4.7> If there is no forwarding or hazard detection, insert nops to ensure correct execution.

4.13.2 [10] <§4.7> Repeat 4.13.1 but now use nops only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modifi ed code.

4.13.3 [10] <§4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?

4.13.4 [20] <§4.7> If there is forwarding, for the fi rst fi ve cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.60.

4.13.5 [10] <§4.7> If there is no forwarding, what new inputs and output signals do we need for the hazard detection unit in Figure 4.60? Using this instruction sequence as an example, explain why each signal is needed.

4.13.6 [20] <§4.7> For the new hazard detection unit from 4.13.5, specify which output signals it asserts in each of the fi rst fi ve cycles during the execution of this code.

4.13.1 Answer, each instruction must FDEMW = Fetch, decode, execute, mem, write)

Things to think about:

1. ~~1.~~ FDEMW
2. We are NOT using forwarding here.

Instructions….
Add r5,r2,r1
Nop
Nop
Lw r3,4(r5)
Lw r2,0(r2)
Nop
Or r3,r5,r3
Nop
Nop
Sw r3,0(r5)

4.13.2 Answer)

1. We ARE using forwarding here.
2. We no longer yield for execution and writing
3. **WE cannot rearrange the given instructions**

Instructions….
Add r5,r2,r1
Lw r3,4(r5)
Lw r2,0(r2)
Or r3,r5,r3
Sw r3,0(r5)

4.13.3 Answer~~, needs to be reviewed~~)

~~Instructions…. (REMOVE ME)~~
~~Add r5,r2,r1~~
~~Lw r3,4(r5)~~
~~nop~~

~~Lw r2,0(r2)~~
~~Nop~~
~~Nop~~
~~Or r3,r5,r3~~
~~Nop~~
~~nop~~
~~Sw r3,0(r5)~~

Without hazard detection unit, then the execution of the instructions will return 0. Because hazard detection unit isn't there and the processor doesn't know when to forward that data. Execution may also be interrupted because of the control flow mismatch.

4.13.4 Answer)

Hazard detection unit is required even with forwarding because its responsible for inserting a one-cycle stall whenever the load provides a value to the instruction that is immediately followed by load. Without this, instructions which depend on the previous instruction will become stagnant data or retrieve previous values from before the load instruction.

Instructions…	Five cycles					Signals		
Add r5,r2,r1	F0	D0	E0	M0	W0	1)PCW:1	ALU1:X	ALU2:X
lw r3, 4(r5)		F0	D0	E0	M0	2)PCW:1	ALU1:X	ALU2:X
Lw r2, 0(r2)			F0	D0	E0	3)PCW:1	ALU1=0	ALU2:0
Or r3, r5,r3				F0	D0	4)PCW:1	ALU1=1	ALU2=0
Sw r3,0(r5)					F0	5)PCW:1	ALU1=0	ALU2=0

4.13.5 Answer)

Reason:

Instruction in ID has to be stalled if it depends on the value which is given by EXECUTION or MEMORY stage. And to do achieve this, we must constantly check the registers. We know that in the instruction in execution, we have to check Rd for loads and R-type. We also know that for

the instructions in memory, the destination register is already given. So we only need to look at that register number.

Inputs:

Additional inputs required to the detection unit are register RD from ID/EX and output of the number from the register from EX/MEM.

Outputs:
No additional outputs are needed. The ability to stop the pipeline using the existing three output signals.

4.13.6 Answer~~, needs to be reviewed~~)

Instructions…	5 cycles
Add r5,r2,r1	IF ID EX MEM WB, PCWRITE=1
nop	
nop	
lw r3, 4(r5)	IF ID, PCWRITE=1
Lw r2, 0(r2)	IF, PCWRITE =1
nop	
Or r3, r5,r3	PCWRITE=0
Nop	
nop	
Sw r3,0(r5)	PCWRITE=0

Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto
Formatted: Font color: Auto

4.18 In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
for(i=0;i!=j;i+=2)
```

b[i]=a[i]–a[i+1];

When writing MIPS code, assume that variables are kept in registers as follows, and that all registers except those indicated as Free are used to keep various variables, so they cannot be used for anything else.

R5 R6 R1 R2 R3 R10, R11, R12

4.18.1 [10] <§4.10> Translate this C code into MIPS instructions. Your translation should be direct, without rearranging instructions to achieve better performance.

4.18.2 [10] <§4.10> If the loop exits aft er executing only two iterations, draw a pipeline diagram for your MIPS code from 4.18.1 executed on a 2-issue processor shown in Figure 4.69. Assume the processor has perfect branch prediction and can fetch any two instructions (not just consecutive instructions) in the same cycle.

4.18.3 [10] <§4.10> Rearrange your code from 4.18.1 to achieve better performance on a 2-issue statically scheduled processor from Figure 4.69.

4.18.4 [10] <§4.10> Repeat 4.18.2, but this time use your MIPS code from 4.18.3.

4.18.5 [10] <§4.10> What is the speedup of going from a 1-issue processor to a 2-issue processor from Figure 4.69? Use your code from 4.18.1 for both 1-issue and 2-issue, and assume that 1,000,000 iterations of the loop are executed. As in 4.18.2, assume that the processor has perfect branch predictions, and that a 2-issue processor can fetch any two instructions in the same cycle.

4.18.6 [10] <§4.10> Repeat 4.18.5, but this time assume that in the 2-issue processor one of the instructions to be executed in a cycle can be of any kind, and the other must be a non-memory instruction.

4.18.1 Answer)

	Add R5,$zero,$zero
Loop:	Beq r5,r6, end
	Sll $t0,r5,2
	Add r10,r1,$t0
	Lw r11,0(r10)
	Lw r10, 4(r10)
	Sub r10, r11, r10
	Add r11,r2,$t0
	Sw r10,0(r11)
	Addi r5,r5,2
	Beq r0,r0,Loop
End	

4.18.2 Answer)

Step	Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	add r5	if	id	ex	me	wb																		
0	beq r5	if	id	id	ex	me	wb																	
1	sll $t0		if	if	id	ex	me	wb																
1	add r10		if	if	id	id	ex	me	wb															
2	lw r11					if	id	ex	me	wb														
2	lw r10					if	id	ex	me	wb														
3	sub r10						if	id	id	ex	me	wb												
3	add r11						if	id	id	ex	me	wb												
4	sw r10							if	if	id	ex	me	wb											
4	addi r5							if	if	id	ex	me	wb											
5	beq r0									if	id	ex	me	wb										
5	nop									if	id	ex	me	wb										
6	beq r5										if	id	ex	me	wb									
6	sll $t0										if	id	ex	me	wb									
7	add r10											if	id	ex	me	wb								
7	lw r11											if	id	id	ex	me	wb							
8	lw r10												if	if	id	ex	me	wb						
8	sub r10												if	if	id	id	id	ex	me					
9	add r11													if	if	if	id	ex	me	wb				
9	sw r10													if	if	if	id	id	ex	me	wb			
10	addi r5																	if	if	id	ex	me	wb	
10	beq r0																	if	if	id	ex	me	wb	
11	beq r5																			if	id	ex	me	wb
11	nop																			if	id	ex	me	wb

4.18.3 Answer)

	Add R5,$zero,$zero
	~~Add R5,$zero,$zero~~ → sll $t0,R5,2
Loop:	Beq r5,r6, end
	~~Sll $t0,r5,2~~ → add r10,r1,$t0
	~~Add r10,r1,$t0~~ → add r12,r2,$t0
	Lw r11,0(r10)
	Lw r10, 4(r10)
	~~Sub r10, r11, r10~~ → lw r10,4(r10
	~~Add r11,r2,$t0~~ → sub r10,r11,r10
	~~Sw r10,0(r11~~) → addi r5,r5,2
	~~Addi r5,r5,2~~ → sw r10, 0(r11)
	Beq r0,r0,Loop
e~~E~~nd:	

Formatted: Strikethrough
Formatted: Strikethrough
Formatted: Strikethrough
Formatted: Strikethrough
Formatted: Strikethrough
Formatted: Strikethrough
Formatted: Strikethrough
Formatted: Not Strikethrough
Formatted: Not Strikethrough

4.18.4 Answer)

Step	Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	add r5	if	id	ex	me	wb													
0	nop	if	id	ex	me	wb													
1	sll $t0		if	id	ex	me	wb												
1	beq R5		if	id	ex	me	wb												
2	add r10			if	id	ex	me	wb											
2	add r12			if	id	ex	me	wb											
3	lw r10				if	id	ex	me	wb										
3	lw r11				if	id	ex	me	wb										
4	sub r10					if	id	id	ex	me	wb								
4	addi r5					if	id	id	ex	me	wb								
5	sw r10						if	if	id	ex	me	wb							
5	beq r0						if	if	id	ex	me	wb							
6	sll $t0								if	id	ex	me	wb						
6	beq r5								if	id	ex	me	wb						
7	add r10									if	id	ex	me	wb					
7	add r12									if	id	ex	me	wb					
8	lw r10										if	id	ex	me	wb				
8	lw r11										if	id	ex	me	wb				
9	sub r10											if	id	id	ex	me	wb		
9	addi r5											if	id	id	ex	me	wb		
10	sw r10												if	if	id	ex	me	wb	
10	beq r0												if	if	id	ex	me	wb	
11	sll $t0														if	id	ex	me	wb
11	beq r5														if	id	ex	me	wb

4.18.5 Answer)

So for 2-issue, each iteration after the first takes 10 cycles. Pipline fill isn't factored because the loop is repeated a million some odd times. 1-issue, the loop will take 11 cycles to execute. And so we have:

Speedup= (1,000,000 *11)/(1,000,000 *10) = 1.1 speed up!

4.18.6 Answer)

4.18.6 [10] <§4.10> Repeat 4.18.5, but this time assume that in the 2-issue processor one of the instructions to be executed in a cycle can be of any kind, and the other must be a non-memory instruction.

So for 2-issue, each iteration after the first take 12 cycles now. While the 1-issue takes the same eleven cycles. So we have:

Speedup (1,000,000*11)/(1,000,000 *12) = .91 slow down!

1. Suppose we have a 5-stage MIPS pipeline with these latencies:
   IF 300 ps
   ID 250 ps
   EX 300 ps
   MEM 400 ps
   WB 350 ps
   a) What is the cycle time of this pipeline? What is the clock rate in GHz?
   b) Suppose we can split any stage of the pipeline into two stages, with equal latencies. Which stage would you choose? What would be the cycle time of the new pipeline?

1.a answer)
Fetch+Decode+Execute+Memory+WriteBack

300+250+300+400+350 = 1600 ps

The clock rate in GHz = 1,600*1,000 =1,600,000 Ghz

Unit Descriptions	
1 Gigahertz:	1 Cycle per Picosecond:
1 Gigahertz is exactly one billion Hertz. 1 GHz = 1 x 10^9 Hz. 1 GHz = 1000000000 Hz.	A period of 1 Picosecond is equal to 1 000 000 000 000 Hertz frequency. Period is the inverse of frequency: 1 Hz = 1 / 0.000 000 000 001 cpps.

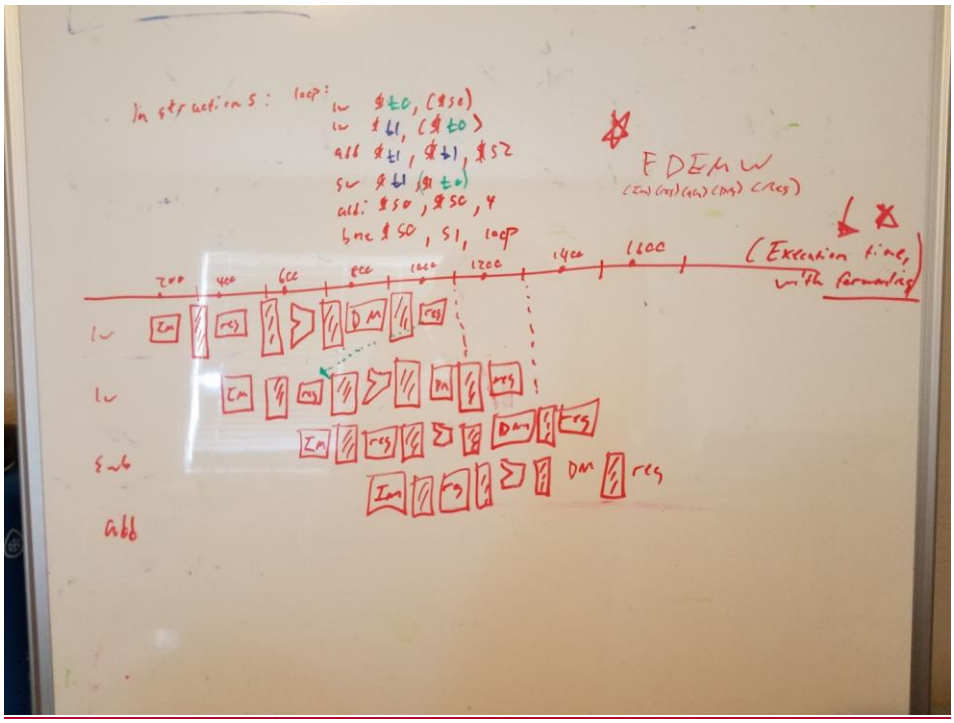
1.b answer)

I would choose the stage with the longest state, so I would pick MEM. The new cycle time of that pipeline would be 400/2 = 200 ps.

1. Suppose we have a 5-stage MIPS pipeline with all possible forwarding. Branch condition and target address are calculated in ID. Show timing in the 5-stage MIPS pipeline for one iteration of the following loop.

```
loop:lw   $t0, ($s0)
     lw   $t1, ($t0)
     add  $t1, $t1, $s2
     sw   $t1, ($t0)
     addi $s0, $s0, 4
     bne  $s0, $s1, loop
```

1.1 answer)



1. Reschedule (i.e., re-order) the instructions within the loop to improve performance. The loop must still produce the same results. Show timing of this MIPS instruction sequence through the pipeline, for two iterations of the loop. Show all stalls clearly, and mark with arrows all cases where forwarding takes place, as in our lectures.

```
loop:   lw   $t0, 0($s0)
        lw   $t1, 4($s0)
        add  $t0, $t1, $t0
        sw   $t0, 0($s1)
        addi $s0, $s0, 8
        addi $s1, $s1, 4
        bne $s0, $s5, loop
```

1.1 answer)

Instructions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
lw $t0, 0($s0)	if	id	ex	me	wb															
lw $t1, 4($s0)		if	id	ex	me	wb														
addi $s0,$s0,8			if	id	ex	me	wb													
add $t0, $t1,$t0				if	id	ex	me	wb												
sw $t0,0($s1)					if	id	ex	me	wb											
addi $s1,$s1,4						if	id	ex	me	wb										
bnq $s0, $s5, loop									if	id	ex	me	wb							
lw $t0, 0($s0)										if	id	ex	me	wb						
lw $t1, 4($s0)											if	id	ex	me	wb					
addi $s0,$s0,8												if	id	ex	me	wb				
add $t0, $t1,$t0													if	id	ex	me	wb			
sw $t0,0($s1)														if	id	ex	me	wb		
addi $s1,$s1,4															if	id	ex	me	wb	
bnq $s0, $s5, loop																if	id	ex	me	wb
Total clock: 20 for 2 interations																				
Stalls are just spaces.																				