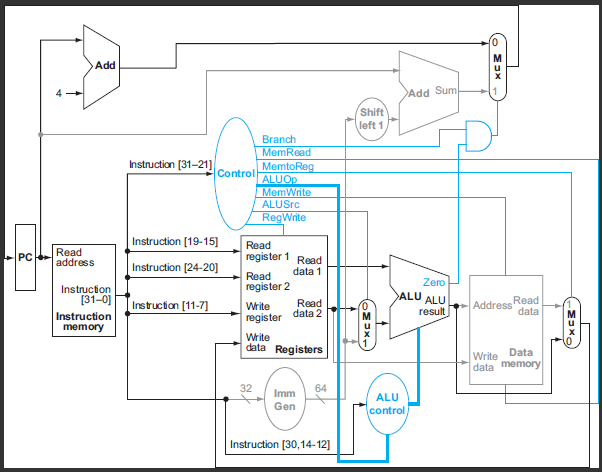
**ECEN 4593 Homework 3 – Solution Guide**

**Question 1**

****

bne x19, x3, -68

x19 = 0x000001fc

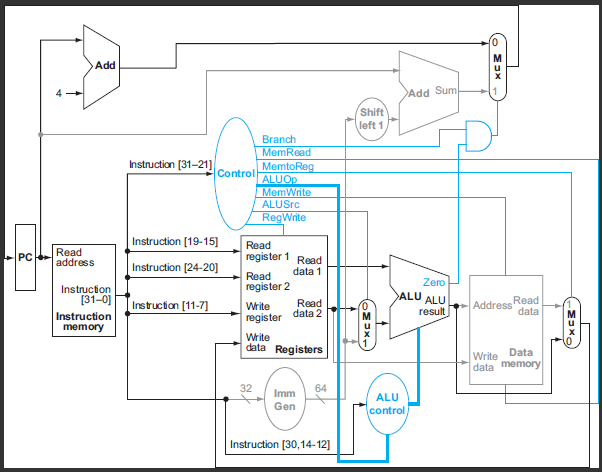
x9 = 0x00000041

Branch => Branch = asserted

0x000001fc – 0x00000041 = 0x000001bb => ALU output != 0 => Zero = not asserted

No others asserted

**Question 2**

****

sh x15, 0(x0)

x15 = 0x0005ffff

x11 = 0x00000044

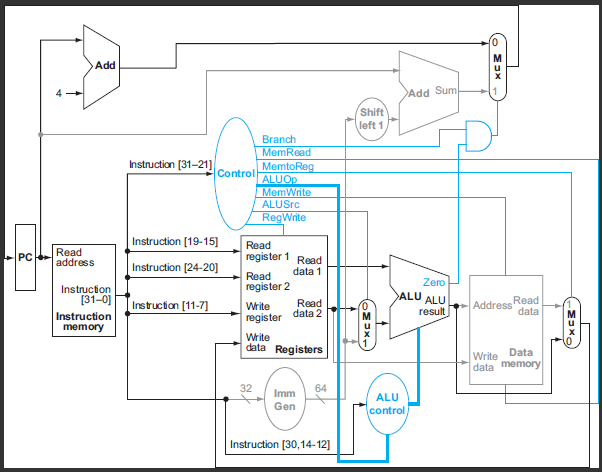
Store uses Immediate => ALUSrc = asserted

Store write to Data Memory => MemWrite = asserted

ALU output = 0x0 + 0x0 = 0x0 = 0 => Zero = asserted

No others asserted

**Question 3**

****

add x13, x16, x13

x13 = 0x00000fff

x16 = 0x00000fff

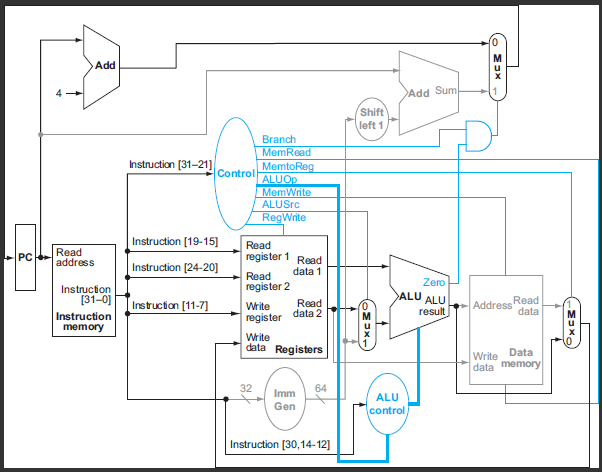
Register => ALUSrc = not asserted

Register result => RegWrite = 1

0x00000fff + 0x00000fff = 0x00001ffe => ALU output != 0 => Zero = not asserted

No others asserted

**Question 4**

****

lw x5, 32(x2)

x5 = 0x0000a5a5

x8 = 0x00000200

Load => write to Register File => RegWrite = asserted

Load uses Immediate => ALUSrc = asserted

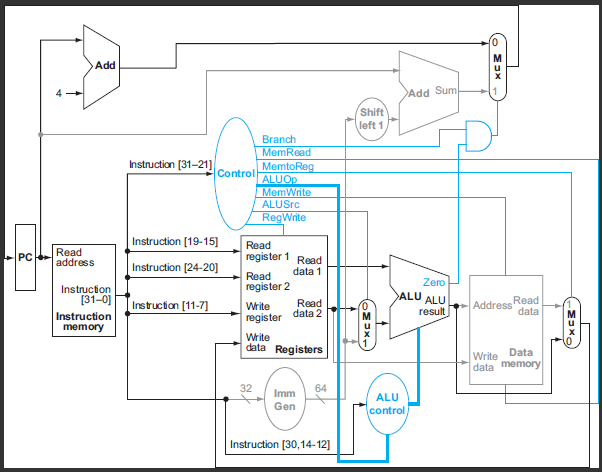
Load reads from the Data Memory => MemRead = asserted

Writeback data comes from Data Memory => MemtoReg = asserted

ALU output 32 + 512 (0x200) != 0 => Zero = not asserted

No others asserted

**Question 5**

****

xori x5, x6, 256

x5 = 0x0000a5a5

x6 = 0x0000a5a5

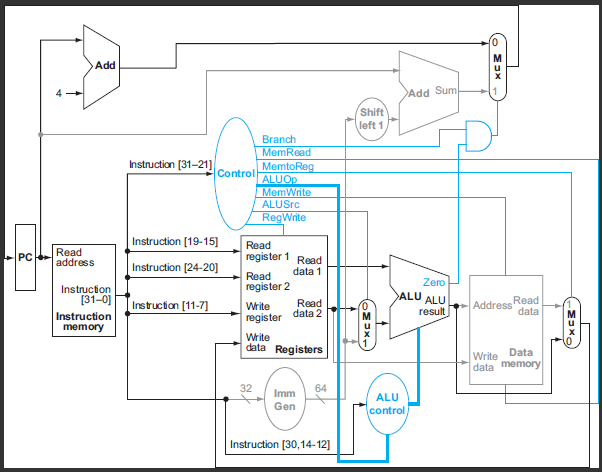
Immediate => ALUSrc = asserted

Register result => RegWrite = asserted

0x0000a5a5 XOR 0x00000100 (200) = 0x0000a4a5 => ALU output != 0 => Zero = not asserted

No others asserted

**Question 5**

****

lw x22, 0(x0)

x22 = 0x000001fc

Load => write to Register File => RegWrite = 1

Load uses Immediate => ALUSrc = 1

Load reads from the Data Memory => MemRead = 1

Writeback data comes from Data Memory => MemtoReg = 1

ALU output 0 + 0 = 0 => Zero = 1

No others asserted

**Question 6**

sub     25% <- Writes Register File

xori    15% <- Uses Immediate, Writes Register File

jalr 10% <- Uses Immediate, Writes Register File

bltu      20% <- Uses Immediate

lh        20% <- Uses Data Memory, Uses Immediate, Writes Register File

sh      10% <- Uses Data Memory, Uses Immediate

Data Memory = 20% + 10% = 30%

Immediate Generator = 15% + 10% + 20% + 20% + 10% = 75%

Write Register File = 25% + 15% + 10% + 20% = 70%

**Question 7**

Branch taken 21% of the time should use Predict not taken to get the best result (79% right)

Branch taken 83% of the time should use Predict taken to get the best result (83% right)

Branch taken 46% of the time should use Dynamic Prediction to get the best result

NOTE: A comment in the textbook indicates that Dynamic Prediction can be right 90% of the time, so I allowed the selection of Dynamic Prediction in all cases as a correct answer.

**Question 8**

The ALU operation is a function of the IF/ID.Opcode field (differentiates between the add of a load and the subtract of a conditional branch, for example), the IF/ID.funct3 field (which differentiates between the various ALU operations in the R-type instructions, for example) and the IF/ID.funct7 field (which differentiates between the SRL and SRA instructions, for example). No other fields are necessary.

**Question 9**

The only types of instructions which assert the RegWrite control line (i.e. select data to write to the Register File) are R-type, I-type, Loads, JAL and JALR. These instructions can be identified by the IF/ID.Opcode field.

**Question 10**

or x5, x5, x6  
addi x2, x2, 4  
srli x6, x6, x2  
sw x5, 36(x5) => Hazard on the instruction three earlier requires no forwarding => n/a

**Question 11**

sw x5, 32(x2)   
add x5, x5,x6  
add x5, x5,x6  
addi x9, x8, x0

sw x7, 36(x2) => Hazard on the instruction two earlier requires forwarding from MEMWB

**Question 12**

sub x5, x5, x6  
lw x5, 0(x5) => Hazard on the immediately preceding instruction forwarding from EXMEM

xor x7, x6, x19  
slli x8, x19, 5

**Question 13**

**Part 1 – No forwarding**

and x12, x14, x16

no dependency => 0 nops

beq x14, x9, -40

x12 dependency from ME => 1 nops

lw x6, 240 (x12)

x6 dependency from EX => 2 nops

sb x6, 27(x12)

Answer = 0, 1, 2

**Part 2 – MEMWB forwarding only**

and x12, x14, x16

no dependency => 0 nops

beq x14, x9, -40

x12 dependency from ME => 0 nops (use MEMWB forwarding)

lw x6, 240 (x12)

x6 dependency from EX => 1 nop (use MEMWB forwarding)

sb x6, 27(x12)

Answer = 0, 0, 1

**Part 3 – MEMWB and EXMEM forwarding**

and x12, x14, x16

no dependency => 0 nops

beq x14, x9, -40

x12 dependency from ME => 0 nops (use MEMWB forwarding)

lw x6, 240 (x12)

x6 dependency from ME => 1 nop (load hazard, use MEMWB forwarding)

sb x6, 27(x12)

Answer = 0, 1, 2

**Question 14**

IF = 300ps

ID = 400ps

EX = 200ps

MEM = 250ps

WB = 100ps

**Part 1**

5-cycle pipeline – cycle is the largest of the stages (ID) = 400 ps

**Part 2**

Single-cycle pipeline – cycle is the sum of all the stages = 1250 ps

**Part 3**

5-cycle pipeline – latency is 5 times the cycle time = 2,000 ps

**Part 4**

Single-cycle pipeline – latency is the same as the cycle time = 1250 ps

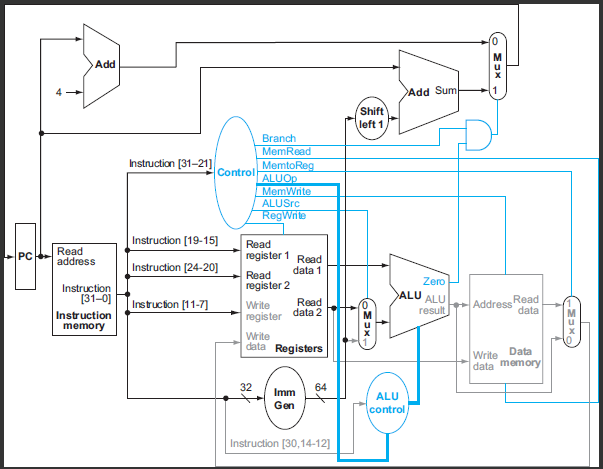
**Part 5**

If we could split a stage, we would split the longest one – ID Stage

**Part 5**

5-cycle pipeline – latency is 6 times the cycle time = 6 x 300 = 1,800 ps

**Question 15**

****

lb x13, 0x47(x17)

ALU ADD operation for the address uses ALU and the ALU unit source 2 mux

Memory offset is sign extended so uses Immediate Generator

Register File is updated from ALU output so uses Write back mux

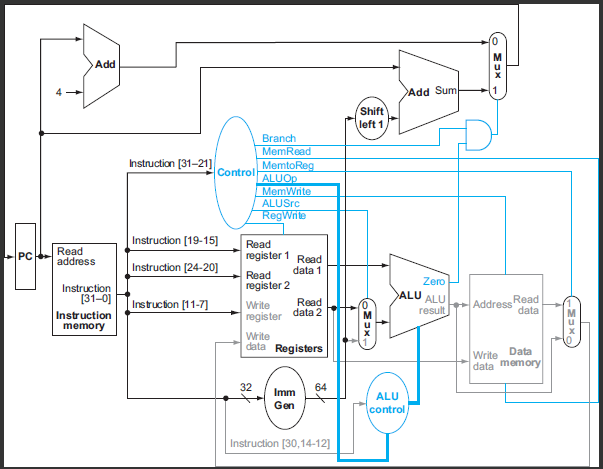
Read Data Memory so uses Data Memory

Writeback data from Data Memory so uses Write Back Mux

PC is updated using the Next PC Input Mux

No other units used

**Question 16**

****

beq x5, x0, -16

Branches always use Branch Target Address adder

Branch address is on a halfword boundary so uses Shift left 1

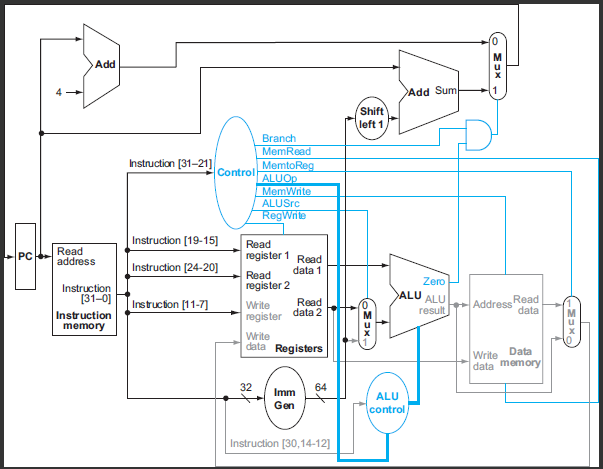
Branch offset is sign extended so uses Immediate Generator

Branch compare SUB or XOR uses ALU and the ALU unit source 2 mux

PC is updated using the Next PC Input Mux

No other units used

**Question 17**

****

xor x5, x7, x9

XOR operation uses ALU and the ALU unit source 2 mux

Writeback data from ALU pipeline so uses Write Back Mux

PC is updated using the Next PC Input Mux

No other units used