EECS2021: Practice Final exam questions

1. How many instructions could fit into a 256 megabyte memory unit, given a 32-bit architecture? **(2 marks)**

a) `256` [✓ **b)** `64`]

c) `32` **d)** `8`

2. How many address bits are needed to specify each byte in a 512 byte memory unit? **(1 mark)**

a) `512` **b)** `8`

c) `32` [✓ **d)** `9`]

3. How many bits do you shift a binary number in order to divide it by 4? **(1 mark)**

2

4. Why are the addresses of all RISC V instructions divisible by 4? **(1 mark)**

Because all instructions are four bytes long, and always starting at addresses that are a multiple of four.

5. Assuming single precision IEEE 754 format, what decimal number is represent by this word:

1 01111101 00100000000000000000000

The decimal number
= (+1)^-1* (2^(125-127))*(1.001)
= (-1)*(2^-2)*(0.25)*(0.3125)
= 0.28125

6. Using 32-bit IEEE 754 single precision floating point with one(1) sign bit, eight (8) exponent bits and twenty three (23) mantissa bits, show the representation of -11/16 (or -0.6875).

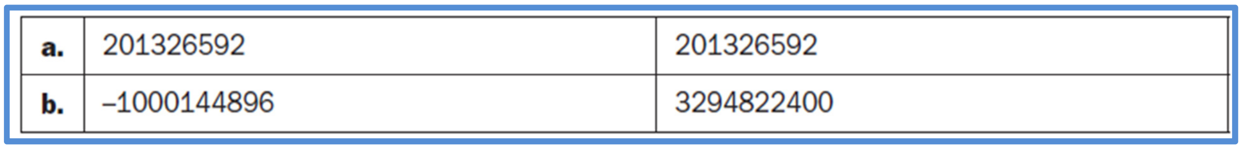
The representation of -0.6875 is:
1 01111110 01100000000000000000000000
1 01111110 01100000000000000000000

7. In a computer architecture, groups of bits have no intrinsic meanings by themselves. What a bit pattern represents depends entirely on how it is used. The following shows bit patterns expressed in hexadecimal notation.

a. 0x0C000000
b. 0xC4630000

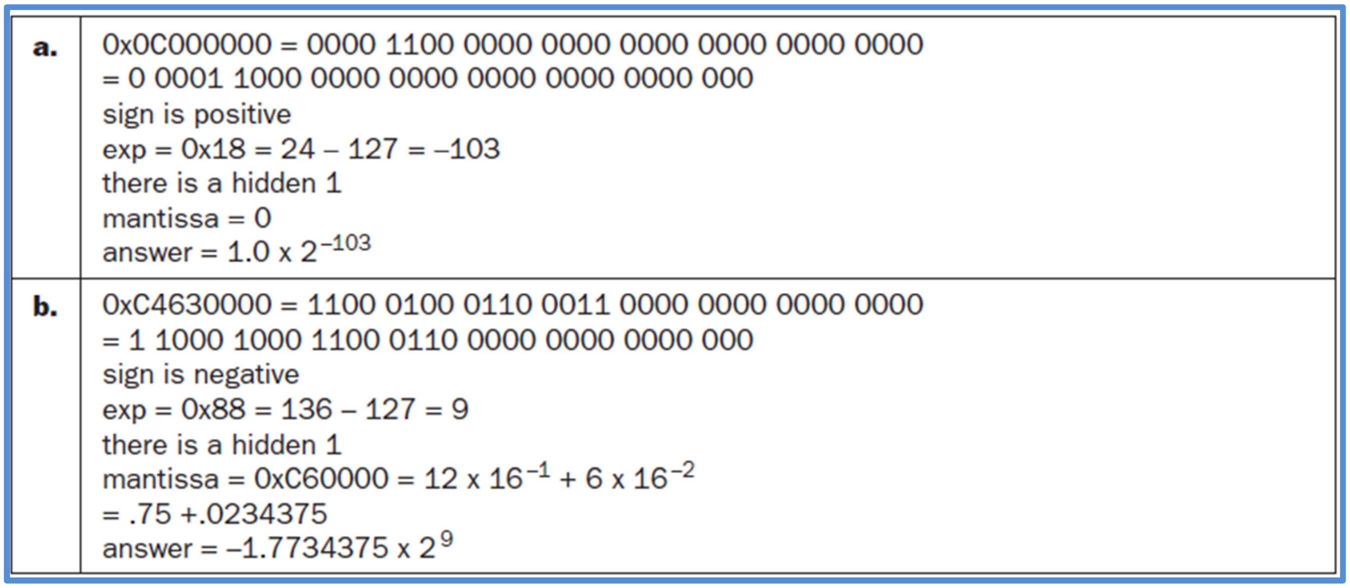
What decimal number does the bit pattern represent if it is

i. A two's complement integer?
ii. An unsigned integer?

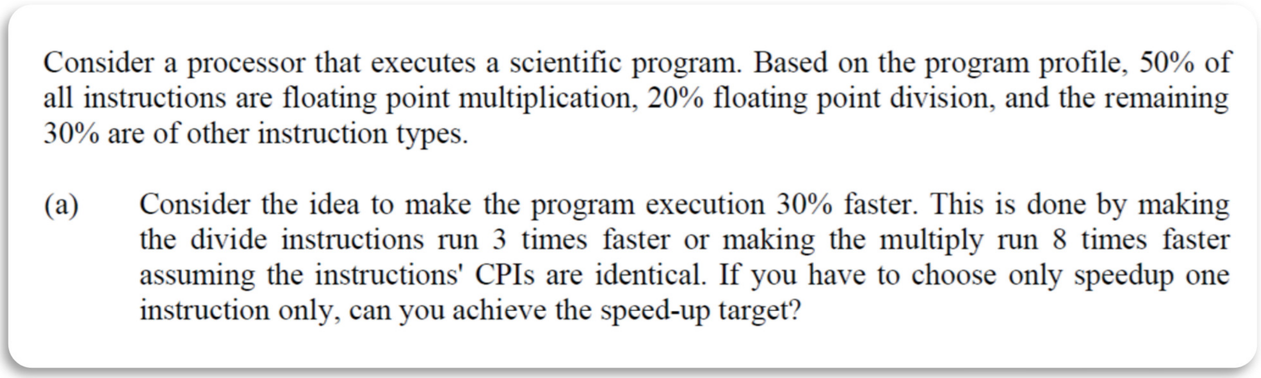


a.	201326592	201326592
b.	–1000144896	3294822400

iii. A floating point number? Use the IEEE 754 standard.



a.	0x0C000000 = 0000 1100 0000 0000 0000 0000 0000 0000 = 0 0001 1000 0000 0000 0000 0000 0000 000 sign is positive exp = 0x18 = 24 – 127 = –103 there is a hidden 1 mantissa = 0 answer = 1.0 x 2^{-103}
b.	0xC4630000 = 1100 0100 0110 0011 0000 0000 0000 0000 = 1 1000 1000 1100 0110 0000 0000 0000 000 sign is negative exp = 0x88 = 136 – 127 = 9 there is a hidden 1 mantissa = 0xC60000 = 12 x 16^{-1} + 6 x 16^{-2} = .75 +.0234375 answer = –1.7734375 x 2^{9}



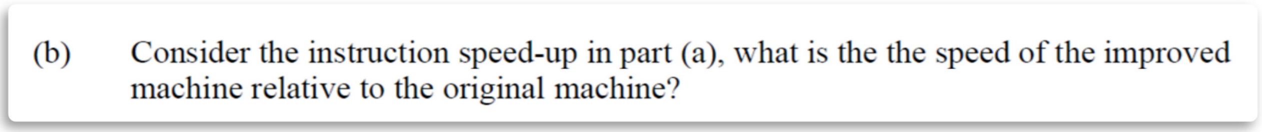
Consider a processor that executes a scientific program. Based on the program profile, 50% of all instructions are floating point multiplication, 20% floating point division, and the remaining 30% are of other instruction types.

(a) Consider the idea to make the program execution 30% faster. This is done by making the divide instructions run 3 times faster or making the multiply run 8 times faster assuming the instructions' CPIs are identical. If you have to choose only speedup one instruction only, can you achieve the speed-up target?

Amdahl's Law:

Execution time after improvement = (Execution time affected by improvement) / (Amount of Improvement) + Execution time unaffected

Execution time after Improvement for Divide = (20)/3 + (50 + 30) = 86.67
Execution time after Improvement for Multiply = (50)/8 + (20 + 30) = 66.67
The aim can be met by making the improvement with Multiply alone.



(b) Consider the instruction speed-up in part (a), what is the the speed of the improved machine relative to the original machine?

Execution time after Improvement = (50)/8 + (20)/3 + (30) = 53.33
Speedup (relative to the original machine) = (100)/(53.33) = 1.88x

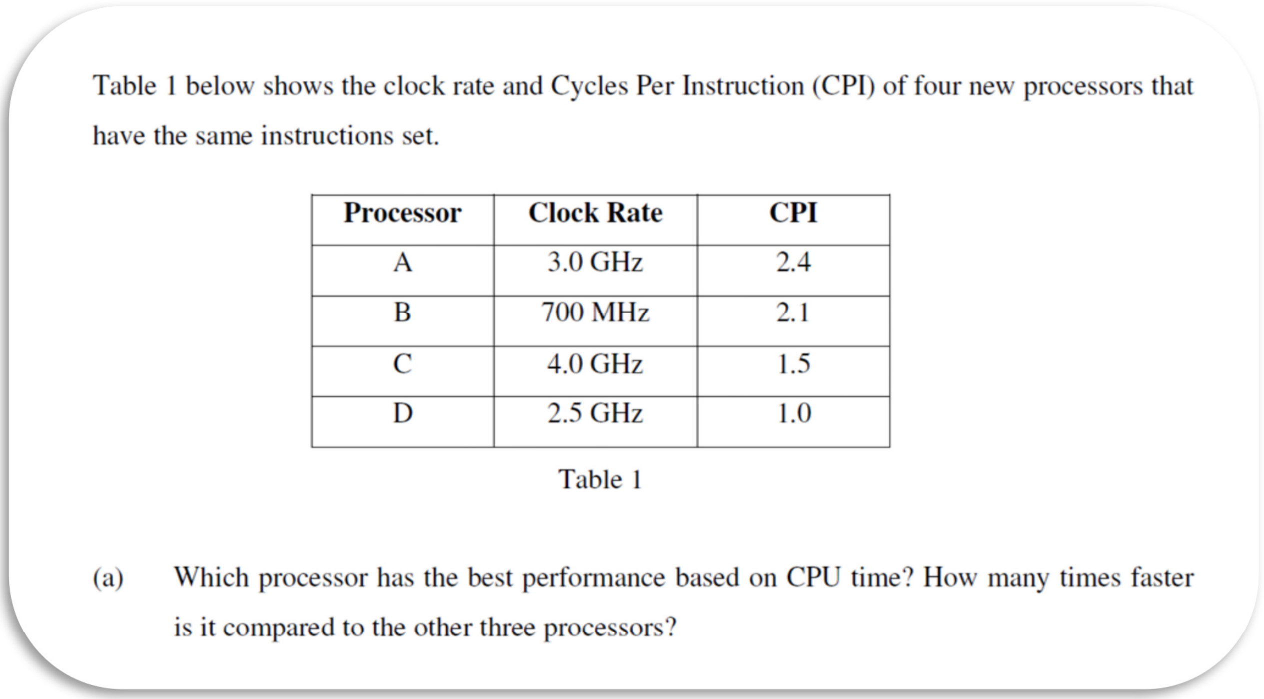


Table 1 below shows the clock rate and Cycles Per Instruction (CPI) of four new processors that have the same instructions set.

Processor	Clock Rate	CPI
A	3.0 GHz	2.4
B	700 MHz	2.1
C	4.0 GHz	1.5
D	2.5 GHz	1.0

Table 1

(a) Which processor has the best performance based on CPU time? How many times faster is it compared to the other three processors?

Answer:

CPU Time = Instructions x CPI x Cycle time

CPU Time A = I x 2.4 x 1/3.0G = 0.8n x I
CPU Time B = I x 2.1 x 1/700M = 3.0n x I
CPU Time C = I x 1.5 x 1/4.0G = 0.375n x I
CPU Time D = I x 1.0 x 1/2.5G = 0.4n x I

C is faster than A by 2.133 times
B by 8.000 times
D by 1.067 times

5. For the following assembly language instructions, write the equivalent machine code instruction in the space provided. You might find the reference information in the appendix helpful for this question.

a) `addi x2, x10, 10`

0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1

b) `lw x2, 20(x5)`

0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1

c) `jal ra, top #where top is at hexadecimal address 0xFF0`

0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1

Assume the PC is at value 0 when this instruction is performed. PC <- PC + {offset,0} << 1
$FF0_{16} = 111111110000_2$, remove two leftmost zeros. One zero is assumed by compiler because jal instruction only saves immediate value from 20:1. One zero is added by the single shift-right operation. So immediate value is 00000000011111111100_2
Immediate value for jal is arranged like so:

imm[20|10:1|11|19:12]	rd	opcode

So immediate value becomes: 0 1111111100 0 00000000
(Hint: this 'jal' is for practice and won't be asked on the final exam.)

6. For the following machine code instructions, provide the equivalent assembly language instruction in the space provided. **(6 marks)**

0000000 10100 00010 000 00011 0010011

sb x2, 0(x20)

3. In the space below, write a Verilog module called `counter` that takes in input signals called `clock`, `reset` and `enable`, and has a 4-bit output signal called `value`. **(3 marks)**

- Make the `value` output increment if `enable` is on when the clock goes high **(3 marks)**

```
module counter (clock, reset, enable, value);

input clock, reset, enable;
output reg [3:0] value

always@(posedge clock)
begin
    if (enable)
        value <= value +1;
end

endmodule
```

1. In the spaces provided below, write the assembly language instruction(s) that corresponds to each of the tasks provided. **(12 marks total)**

a) Perform a right arithmetic shift on the value in `x3`. The number of bits to shift `x3` by is `5`. The result will be stored back into `x3`. **(3 marks)**

sra x3, x3, 5

5. In the space below, write a short assembly language program that is a translation of the program on the right. You can assume that `i` has been placed on the top of the stack, and should be replaced by the return value before returning to the calling program. Make sure that you comment your code so that we understand what you're doing. **(8 marks)**

```
int make_even (int i) {
   if (i % 2 == 1)
      return i-1;
   else
      return i;
```

Answer:

```
                addi x2, x0, 1600       //init sp
                addi x5, x0, 847        //value of i
                sd x5, 0(x2)            //save to the top of stack
                jal x1, make_even       //evoke make even, x9 store
                                        //make_even(i) after return

                beq x0, x0, exit

make_even:      ld x9, 0(x2)
                andi x10, x9, 1                 //get LSB of i
                beq x10, x0, return     //if even, return
                addi x9, x9, -1         //else, i=i-1
return:         sd x9, 0(x2)            //save back to the top of stack
                jalr x0, 0(x1)
exit:
```

(a) The Multiplication Hardware shown in Figure 2 performs the sequential multiplication of the Multiplicand with the Multiplier.

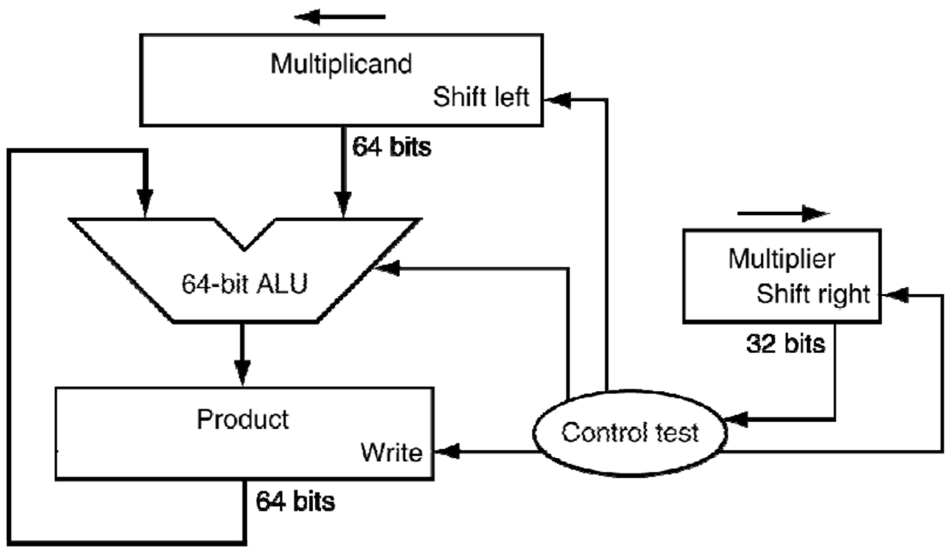
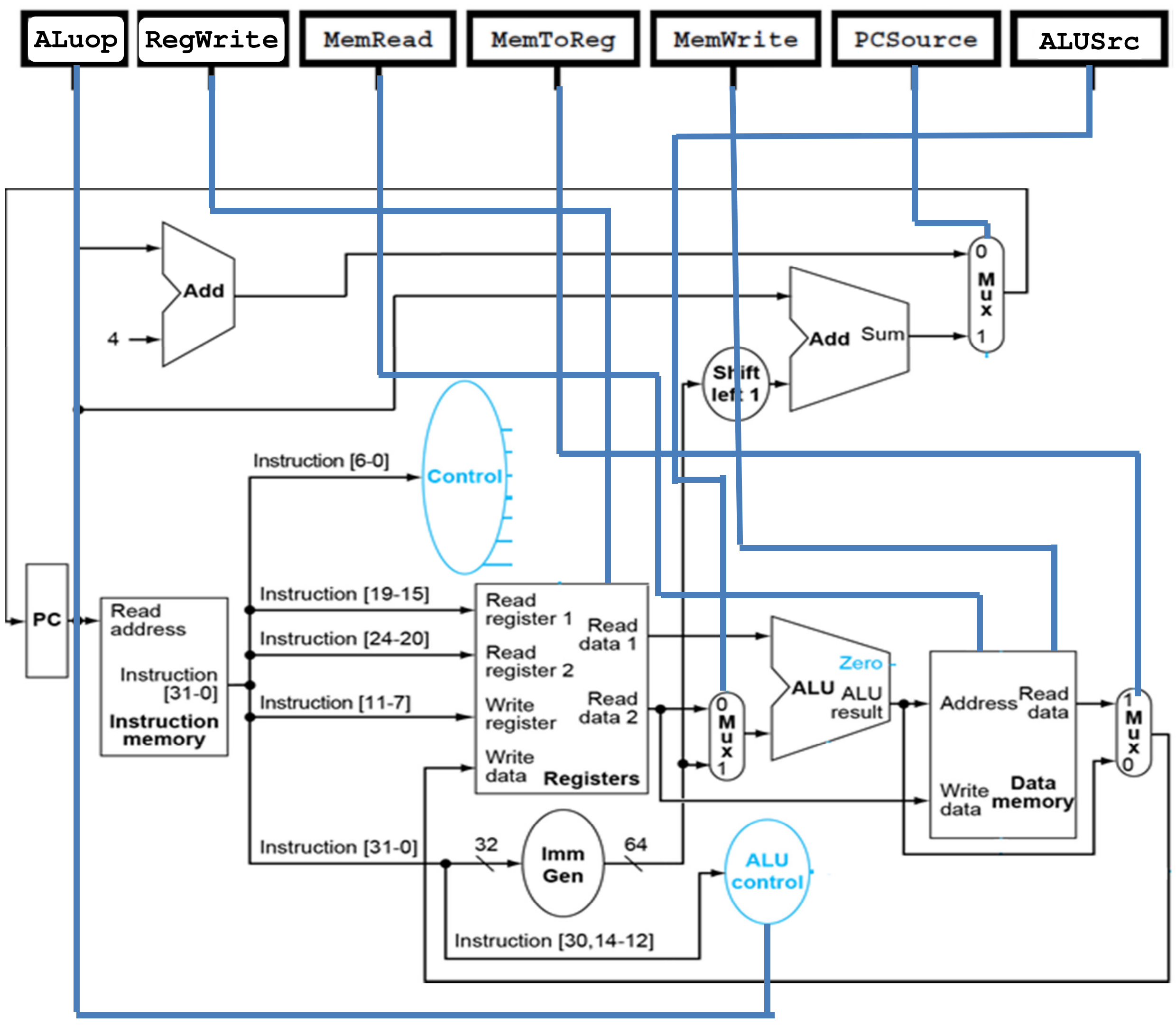


Figure 2: Multiplication Hardware

Describe the sequential multiplication process of 170 x 9 (Binary: 170_d = $1010\ 1010_b$, 9_d = 1001_b) by filling up Table 3 given below. Write the MIPS instructions performed in each step in the 'Operation' column of the table. The first operation is shown as example.

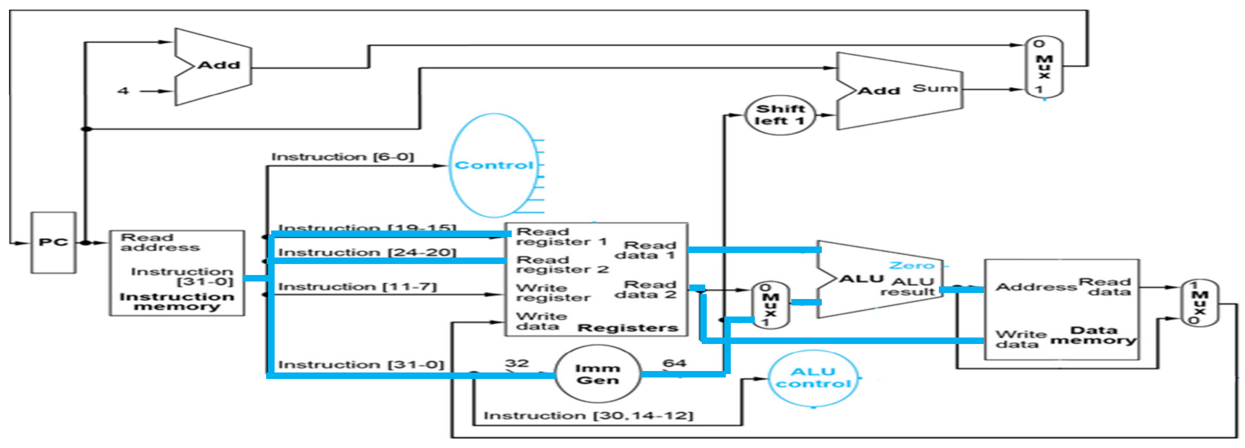
Operation	**Multiplicand (M_D)**	**Multiplier (M_R)**	**Product (P)**
Initial values	0110 1101	0110	0000 0000 0000
$P = P + M_D$ sll M_D, srl M_R	0 1101 1010	0011	0000 0000 0000
$P = M_D$ sll M_D, srl M_R	01 1011 0100	0001	0000 1101 1010
$P = P + M_D$ sll M_D, srl M_R	011 0110 1000	0000	0010 1000 1110
$P = P$ sll M_D, srl M_R	0110 1101 0000	0000	0010 1000 1110

Draw the connections between the control signals in the boxes to the corresponding datapath components.

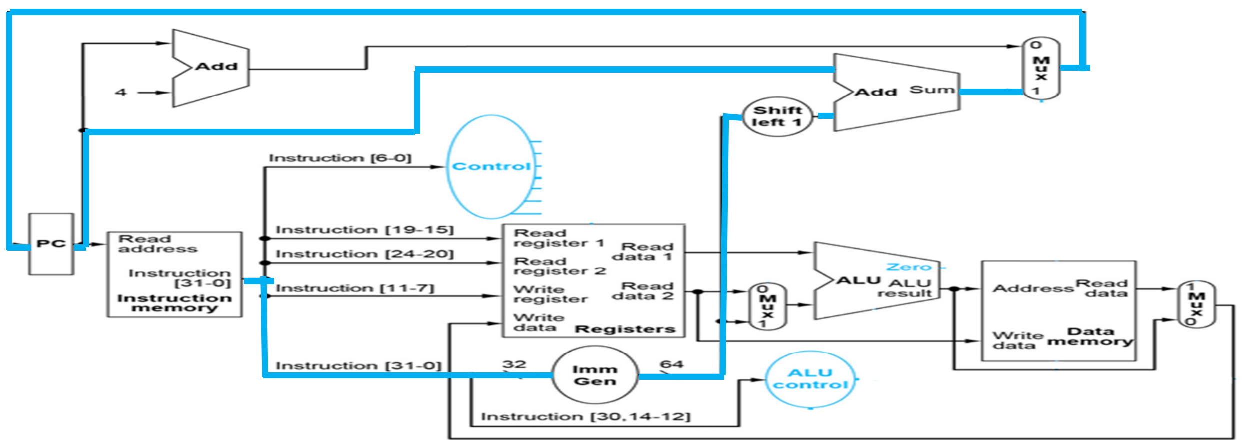


4. Consider the datapaths below. For each of the following operations, highlight the path that the data needs to take, from start to finish. **(12 marks)**

a) Store the value in x7 at the current stack pointer location.



b) Increment the program counter by the immediate value.



For each of the processor tasks below, indicate what the values of the following control unit signals will be by filling in the boxes next to each signal with the signal values. **(12 marks)**

- If a control signal doesn't affect the operation, fill in its value with an X.
- For ALUOp, if you don't know the values, just write what kind of operation is taking place.

a) Reduce the program counter by the value given in the current instruction's immediate bits.

Branch	1	MemRead	0	MemWrite	0		
MemToReg	x	ALUOp	xx	ALUSrc	x	RegWrite	0

b) Load the content of base address + offset into register x5.

Branch	0	MemRead	1	MemWrite	0		
MemToReg	1	ALUOp	00	ALUSrc	1	RegWrite	1

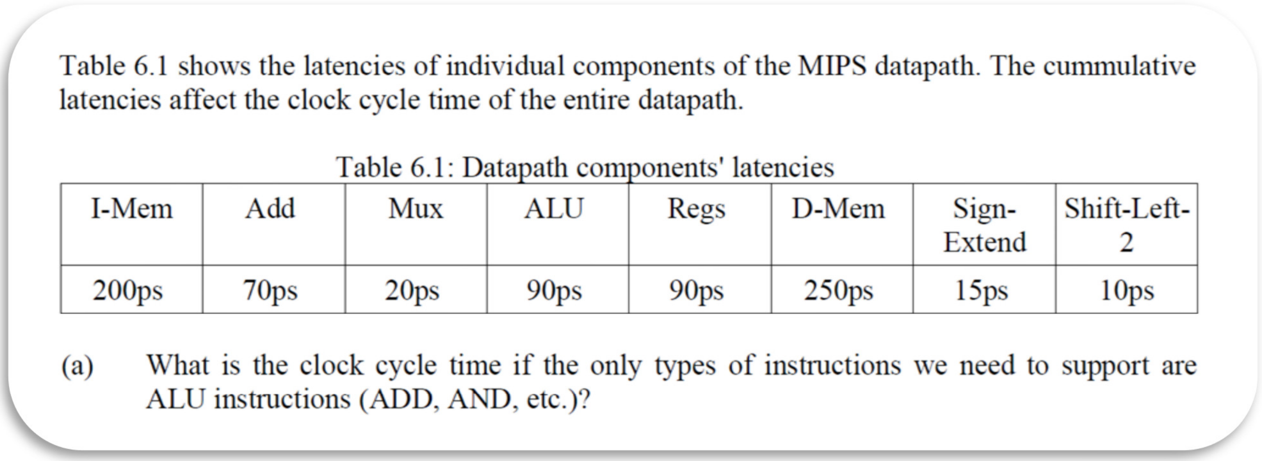


Table 6.1 shows the latencies of individual components of the MIPS datapath. The cummulative latencies affect the clock cycle time of the entire datapath.

Table 6.1: Datapath components' latencies

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

(a) What is the clock cycle time if the only types of instructions we need to support are ALU instructions (ADD, AND, etc.)?

Answer:

The longest-latency path for ALU operations is through I-Mem, Regs, Mux (to select ALU operand), ALU, and Mux (to select value for register write). Note that the only other path of interest is the PC-increment path through Add (PC + 4) and Mux, which is much shorter. So for the I-Mem, Regs, Mux, ALU, Mux path we have:

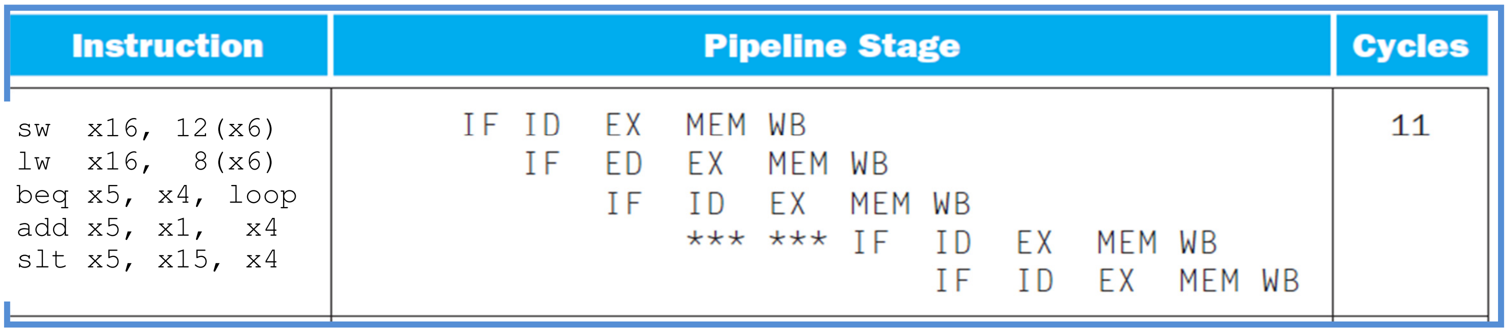
200ps + 90ps + 20ps + 90ps + 20ps = 420ps

6. Consider the following fragment of RISC V code:

```
sw   x16, 12(x6)
lw   x16,  8(x6)
beq  x5,   x4,  loop     ; Assume R5 != R4
add    x5,   x1,    x4
slt    x5,   x15,   x4
```

Assume that all branches are perfectly predicted (such that this eliminates all control hazards) and that no delay slots are used. What is the total execution time of this instruction sequence in the 5-stage pipeline?

Answer:



Instruction	Pipeline Stage											Cycles
sw x16, 12(x6)	IF	ID	EX	MEM	WB							11
lw x16, 8(x6)		IF	ED	EX	MEM	WB						
beq x5, x4, loop			IF	ID	EX	MEM	WB					
add x5, x1, x4				***	***	IF	ID	EX	MEM	WB		
slt x5, x15, x4							IF	ID	EX	MEM	WB	

Try these questions first before looking at the answers provided on the last pages.

The table below shows a series of 5 instructions that are to be performed by a RISC processor.

a) Fill in the table for each data dependency you find. For example, if instruction 2 depends on register 14 from instruction 0, you would write "x14 from I0". If there are no data dependencies, leave the table entry blank.

Instruction	Depends on (Register # from Instruction #)	Depends on (Register # from Instruction #)
I1: sub x2, x14, x15		
I2: and x12, x2, x5		
I3: lw x13, 6(x12)		
I4: andi x5, x12, 7		
I5: add x13, x12, x10		

b) Assuming a standard 5-stage RISC pipeline is used to implement the above instructions and cc0 to cc14 represent cycles (time). Fill out the schedule in the table below for each instruction and label all stalls with *.

i. Fill out the schedule in the table below for a pipeline **with no forwarding**.

Instruction	cc0	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13
I1														
I2														
I3														
I4														
I5														

ii. Complete the following table for a pipeline **with data forwarding** and draw an arrow between the stages that use the data forwarding.

Instruction	cc0	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13
I1														
I2														
I3														
I4														
I5														

Perform a binary multiplication operation on the following floating point numbers.

– 10 * 0.03125

(i) Binary conversion

– 10 in binary form = – _____________ x $2^{\boxed{}}$

0.03125 in binary form = ____________ x $2^{\boxed{}}$

(ii) Calculate result's exponent: $2^{\boxed{}}$

(iii) Multiply the significands calculated in step (i) above:

________ x _________ = ____________________

(iv) Normalize result & check for over/underflow:

(v) Rounding or any other necessary steps:

(vi) Final result in binary form: __________ x $2^{\boxed{}}$

(vii) Final result in decimal form: ________________________

Answers:

The table below shows a series of 5 instructions that are to be performed by a RISC processor.

c) Fill in the table for each data dependency you find. For example, if instruction 2 depends on register 14 from instruction 0, you would write “x14 from I0”. If there are no data dependencies, leave the table entry blank.

Instruction	Depends on (Register # from Instruction #)	Depends on (Register # from Instruction #)
I1: sub x2, x14, x15		
I2: and x12, x2, x5	x2 from I1	
I3: lw x13, 6(x12)	x12 from I2	
I4: andi x5, x12, 7	x12 from I2	
I5: add x13, x12, x10	x12 from I2	

d) Assuming a standard 5-stage RISC pipeline is used to implement the above instructions and cc0 to cc14 represent cycles (time). Fill out the schedule in the table below for each instruction and label all stalls with *.

i. Fill out the schedule in the table below for a pipeline **with no forwarding**.

Instruction	cc0	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13
I1	IF	ID	EX	MEM	WB									
I2		*	*	IF	ID	EX	MEM	WB						
I3			*	*	*	*	IF	ID	EX	MEM	WB			
I4				*	*	*	*	IF	ID	EX	MEM	WB		
I5					*	*	*	*	IF	ID	EX	MEM	WB	

ii. Complete the following table for a pipeline **with data forwarding** and draw an arrow between the stages that use the data forwarding.

Instruction	cc0	cc1	cc2	cc3	cc4	cc5	cc6	cc7	cc8	cc9	cc10	cc11	cc12	cc13
I1	IF	ID	EX	MEM	WB									
I2		IF	ID	EX	MEM	WB								
I3			IF	ID	EX	MEM	WB							
I4				IF	ID	EX	MEM	WB						
I5					IF	ID	EX	MEM	WB					

Perform a binary multiplication operation on the following floating point numbers.

– 10 * 0.03125

(viii) Binary conversion

– 10 in binary form = – _1.010_______ x $2^{\boxed{3}}$

0.03125 in binary form = _1.0000______ x $2^{\boxed{-5}}$

(ix) Calculate result's exponent: $2^{\boxed{-2}}$

(x) Multiply the significands calculated in step (i) above:

_-1.010_ x _1.00_ = _-1.01000_______

(xi) Normalize result & check for over/underflow:

(xii) Rounding or any other necessary steps:

(xiii) Final result in binary form: _-1.01000_ x $2^{\boxed{-2}}$

(xiv) Final result in decimal form: __–0 .3125________________