CSE 141 – MIDTERM EXAM (100 PTS)

The total number of points that can be earned is 110, of which 10 points are extra credit.

SUBMISSION INSTRUCTIONS (READ CAREFULLY):

1. Please type your responses in a separate document and submit the PDF file to gradescope. ONLY typed PDF documents will be accepted as a valid submission. However, if there are any questions that ask you to draw a diagram, those can be hand-drawn, but must be attached to the final PDF file. Please do not submit a separate file for hand-drawn diagrams. Only one, final PDF file will be accepted.

2. Unless you have general questions about the exam, you should not be discussing on Piazza. **If you have general questions, please email bhahn221@eng.ucsd.edu or post question to Piazza as a private question so that only instructors can see**. It is okay to refer to the class materials, but discussing with other classmates about the questions will not be allowed. It is crucial that the students do not violate the academic integrity.

3. No late submissions allowed!

1. **(12 pts)** Consider the following comparison of overall performance of machines A and B across various applications I, II and III. **"Machine A"** column shows the execution times of Machine A for applications I, II, and III**. "Machine B"** column shows the execution times of Machine B for applications I, II, and III**. "SPEEDUP"** column shows the speedups of Machine A over Machine B for applications I, II, and III. Round up all numbers to one decimal place.

	Machine A Execution Times	Machine B Execution Times	SPEEDUP
Application I	10 sec	**20 sec**	2
Application II	**8 sec**	12 sec	1.5
Application III	5 sec	11 sec	**2.2**
Average			1.9

a) **(4 pts)** Fill out the missing numbers in the table.

See above

b) **(4 pts)** Calculate the average speedup across multiple applications using (i) Geometric Mean and (ii) Arithmetic Mean?

i) 1.876

ii) 1.9

c) **(4 pts)** If the arithmetic mean and the geometric mean are different, explain where the

difference coming from and which metric would you chose?

After rounding they are the same, but the geometric mean was actually ~1.876. This slight difference is because of the geometric mean formula using products. I would choose the geometric mean as there are only 3 data points, and better represents the actual performance.

2. **(6 pts)** Consider the following figures:

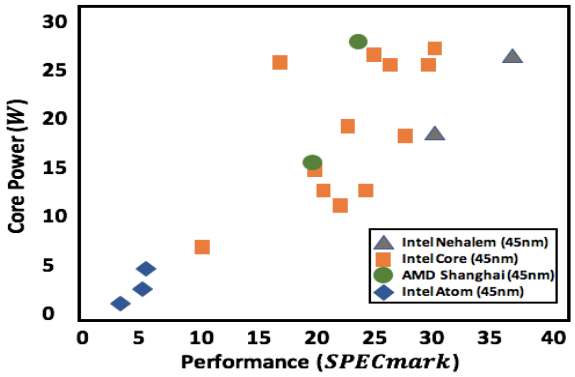


Figure A

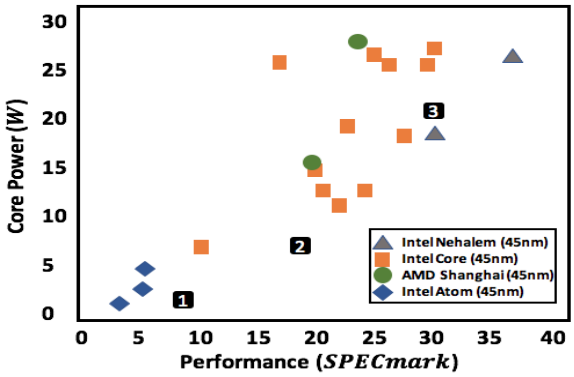
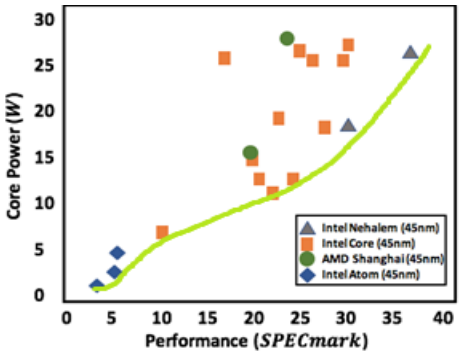


Figure B

Figure A shows the power/performance single-core design space. Each processor's performance is collected from the SPEC website. In Figure A, the x-axis is the (SPECmark) of the processor, and the y-axis is the core power budget.

a) **(2 pts)** Depict Pareto Optimal Frontier on **Figure A**.



b) **(2 pts)** New processor designs 1, 2, 3 are added to the design space as shown in **Figure B**. Which of these added designs would you keep and which would you discard from Pareto optimality viewpoint? Why?

I would remove processors 1 and 2, as they would lower the bounded curve of the pareto optimality frontier.

c) **(2 pts)** Draw the Pareto Frontier on **Figure B**?

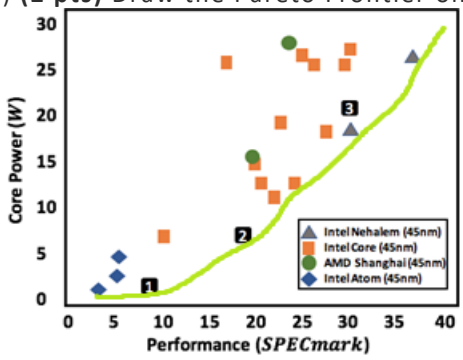


Figure B

3. **(6 pts)** Consider the following table:

Instruction Type	% Execution Time	CPI
INT	40 %	1
BR	20 %	4
LD	30 %	2
ST	10	3

Based on the above table and using Amdahl's law, which of these improvements is best?

1. Branch CPI: 4 → 3
2. Increase clock frequency: 2 → 2.3 GHz
3. Store CPI: 3 → 2

Show your calculations.

4 instructions

CPU Execution Time = .4 * 1 + .2 * 4 + .3 *2 + .1 * 3 = 2.1

1. **New time = .4 * 1 + .2 * 3 + .3 * 2 + .1 * 3 = 1.9**

   Speedup = 2.1/1.9 = 1.1x
2. **Speedup = 2.3/2 = 1.15x**
3. **New time = .4 * 1 + .2 * 4 + .3 * 2 + .1 * 2 = 2**

   Speedup = 2/1.9 = 1.05x

So, increasing clock frequency is the best improvement

4. **(6 pts)** Imagine that you are a manager of hardware implementation team. One of the hardware designers in your team proposed hardware optimizations as follows:

   - Eliminates 10% of the instructions
   - Decreases the CPI by 10% of the remaining instructions
   - Decreases the clock rate by 14%

   a) **(3 pts)** Is this set of optimizations worth implementing? Show your calculations.
   Yes,

   # New instructions = 0.9 * # old instructions

   New Cpi = 0.9 * old Cpi

   New clock rate = 0.86 * old clock rate

   New execution time = $(0.9)^2$ * 1/0.86 = 0.94

   b) **(3 pts)** What is the resulting speedup?

   1/0.94 = 1.06x

5. **(12 pts)** The following figure shows a variety of multicore chips.

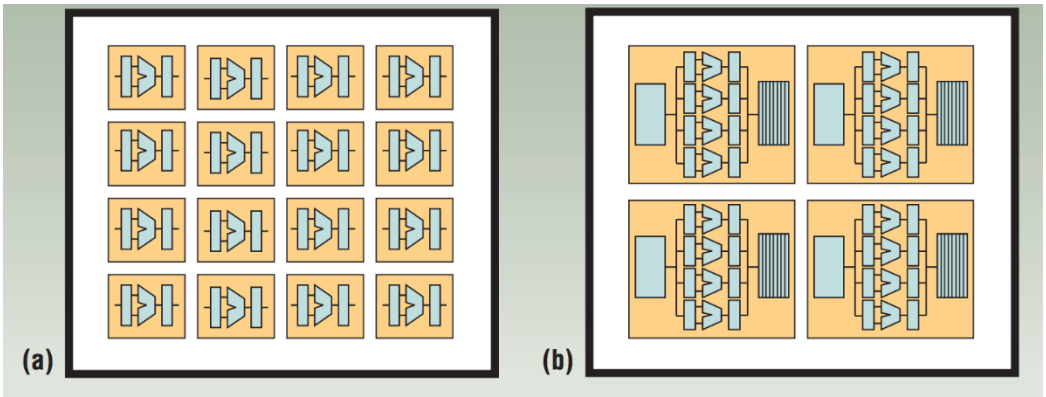


Figure (a): Symmetric multicore chip with 16 one-Base Core Equivalent (BCE) cores; **Figure (b):** symmetric multicore chip with 4 four-BCE cores

To apply Amdahl's law to a multicore chip, we need a cost model for the number and performance of cores that the chip can support.

First, we assume that a multicore chip of given size and technology generation can contain at most *n* base core equivalents (BCE), where a single BCE implements the baseline core. We are agnostic to what limits a chip to *n* BCEs. It might be power, area, or some combination of power, area, and other factors.

Second, we assume that (micro-) architects have techniques for using the resources of multiple BCEs to create a core with greater sequential performance. Let the performance of a single-BCE core be 1. We assume that architects can use the resources of *r* BCEs to create a powerful core with sequential performance perf(r).

Let's assume efforts that devote *r* BCE resources will result in sequential performance $\sqrt{r}$. Thus, architectures can double performance at a cost of four BCEs, triple it for nine BCEs, and so on. In other words, assume perf(*r*) = $\sqrt{r}$

a) **(4 pts)** What happens to the sequential and parallel execution speedups with increasing core resources when perf(*r*) > *r*?

With more core resources, sequential and parallel executions both increase.

b) **(4 pts)** What happens to the sequential and parallel execution speedups with increasing core resources when perf(*r*) < *r*?

With more core resources, sequential execution is greater but parallel execution decreases.

c) **(4 pts)** Considering the chip uses one core to execute sequentially at performance perf(*r*). And it uses all n/r cores to execute in parallel at performance perf(*r*) × *n*/*r*. Overall, we get:

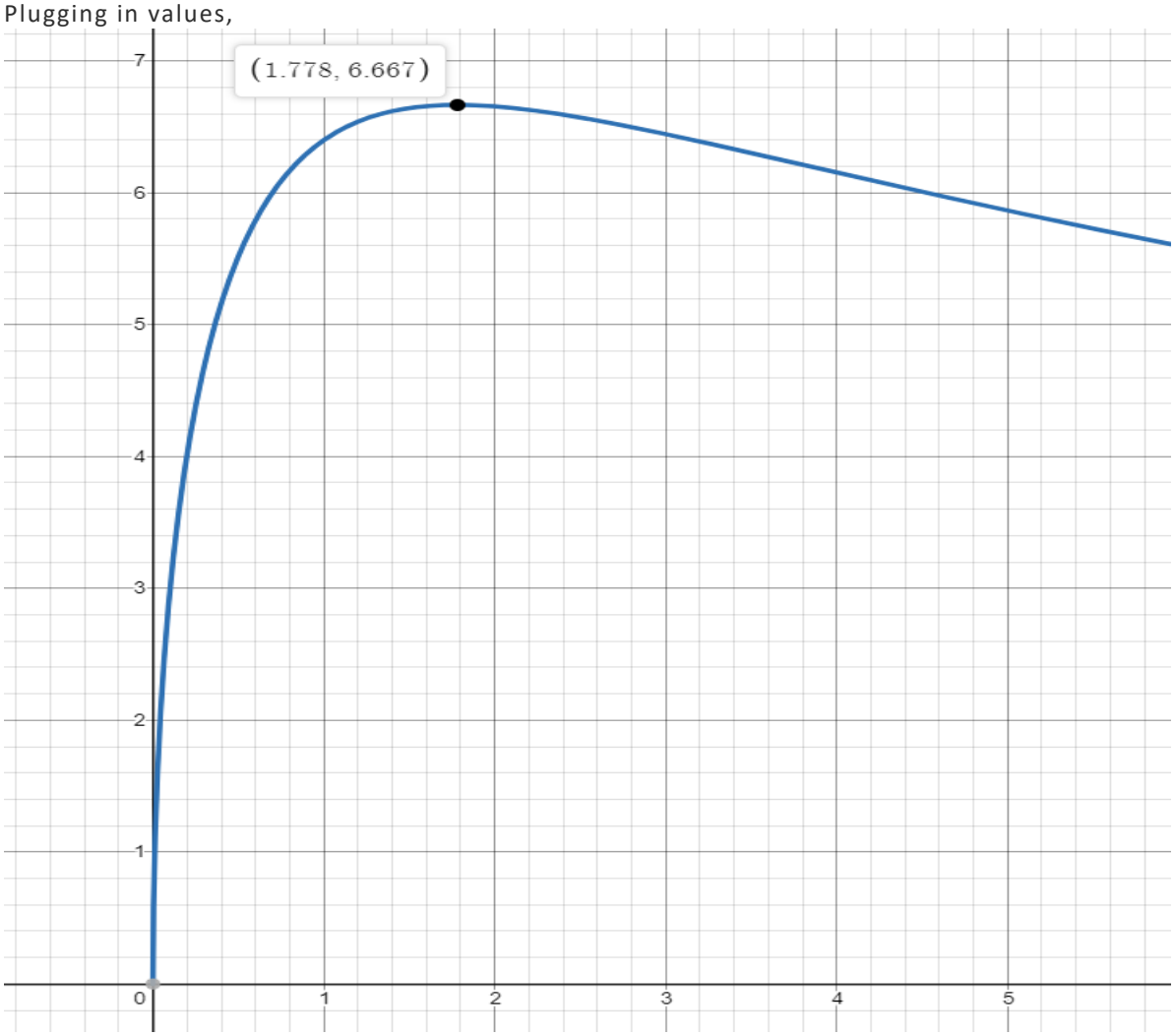
$$\text{Speedup}_{\text{symmetric}}(f,n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f \cdot r}{perf(r) \cdot n}}$$

Assuming *f*=0.9, *n*=16, calculate the optimum *r* to achieve max speedup.

1/(.1/perf(r) + .9r/(16perf(r))

= 16sqrt(r)/(1.6 + .9r)

Plugging in values,

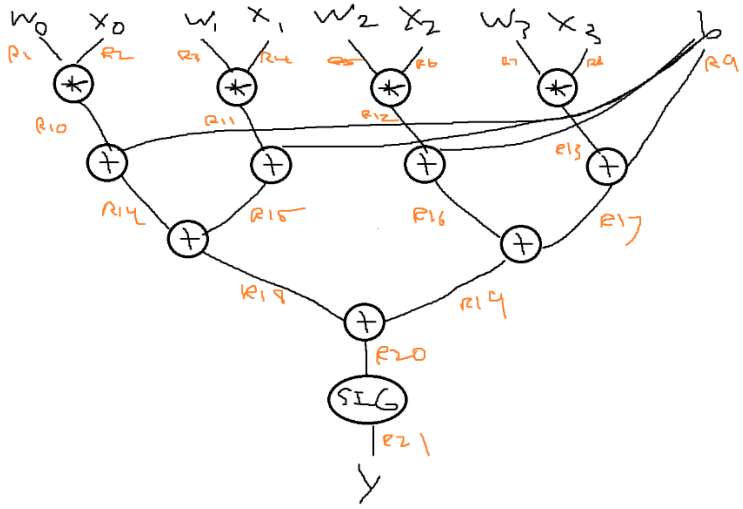


We can see that the maximum performance we can get when r = 1 is 6.5, 6.65 when r = 2, and goes down for when r >= 3. Thus our best r is 2

6. **(6 pts)** Consider the following mathematical expression:

$$y = sigmoid(\sum_{i=0}^{3} w_i * x_i + b)$$

a) **(1 pt)** Draw a DFG for the above code segment. Write the operations on the nodes and registers on the edges for each DFG. Note that the sigmoid function can be represented as a single node in the DFG.



b) **(1 pt)** Suppose we want to execute the DFG you generated in the previous question on a virtual dataflow machine. How many cycles does it take to execute the DFG? Note that each operation takes one cycle.

5 cycles.

c) **(1 pt)** For the DFG, write an instruction sequence in terms of the registers and operations you used in the previous question. Assume you have only 16 registers (R0-R15). Assume you have instruction SIG for sigmoid function.

LD R1, W0

LD R2, X0

LD R3, W1

LD R4, X1

LD R5, W2

LD R6, X2

LD R7, W3

LD R8, X3

LD R9, b

MULT R10, R1, R2

MULT R11, R3, R4

MULT R12, R5, R6

MULT R13, R7, R8

ADD R14, R10, R9

ADD R15, R11, R9

ADD R16, R12, R9

ADD R17, R13, R9

ADD R18, R14, R15

ADD R19, R16, R17

ADD R20, R18, R19

SIG R21, R20

ST (Y), R21

d) **(1 pt)** Convert each instruction sequence to static single assignment (SSA) form. Let us use the alpha notation (α) to name the registers, as we did in class. Assume you have infinite number of registers.

LD A1, W0

LD A2, X0

LD A3, W1

LD A4, X1

LD A5, W2

LD A6, X2

LD A7, W3

LD A8, X3

LD A9, b

MULT A10, A1, A2

MULT A11, A3, A4

MULT A12, A5, A6

MULT A13, A7, A8

ADD A14, A10, A9

ADD A15, A11, A9

ADD A16, A12, A9

ADD A17, A13, A9

ADD A18, A14, A15

ADD A19, A16, A17

ADD A20, A18, A19

SIG A21, A20

ST (Y), A21

e) **(1 pt)** Suppose we are generating instructions for a target architecture. Let us assume our target architecture has only two units that can perform arithmetic operations. In this case, how many cycles does it take to execute the DFG? Assume that each operation takes one cycle.

10 cycles.

f) **(1 pt)** Now suppose we want to generate instructions for stack machine instead of dataflow machine. Write the instruction sequence for a stack machine such as Java Virtual Machine that represents the expression given in the problem.

Push w0

Push x0

Push w1

Push x1

Push w2

Push x2

Push w3

Push x3

Push b

Mult ;; pop pop mult push
Mult ;; pop pop mult push

Mult ;; pop pop mult push

Mult ;; pop pop mult push

Add ;; pop pop add push

Add ;; pop pop add push

Add ;; pop pop add push

Add ;; pop pop add push

Add ;; pop pop add push

Add ;; pop pop add push

Add ;; pop pop add push

Sig ;; pop sig push

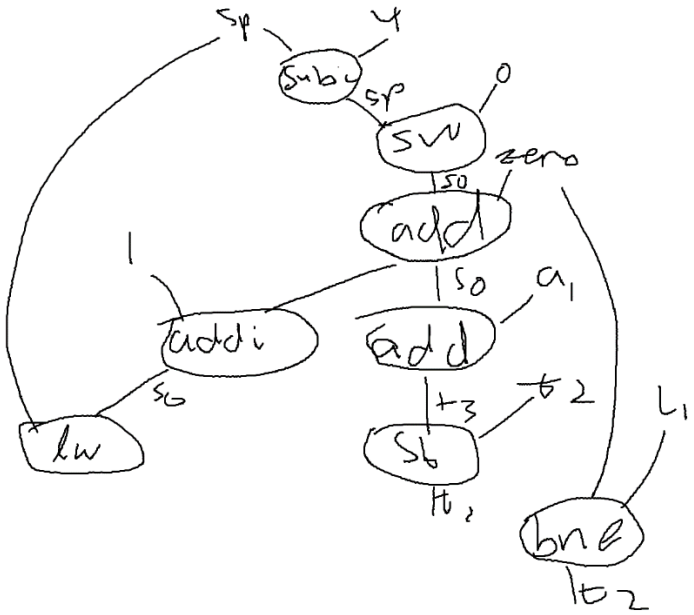
7. **(6 pts)** Consider the following C code and corresponding assembly program below.

```
// C code
void strcpy (char x[], char y[])
{
     int i;
     i = 0;
     while (x[i] = y[i]) != 0}
          i += 1;
}
```

```
;; assembly
strcpy: subi $sp, $sp, 4
        sw   $s0, 0($sp)
        add  $s0, $zero, $zero
L1:     add  $t1,$a1,$s0
        lb   t2,0($t1)
        add  $t3,$a0,$s0
        sb   $t2,0($t3)
        addi $s0,$s0,1
        bne  $t2,$zero,L1
        lw   $s0,0($sp)
        add1 $sp,$sp,4
        jr   $ra
```

source: http://www.es.ele.tue.nl/education/Computation/Cmp18.pdf

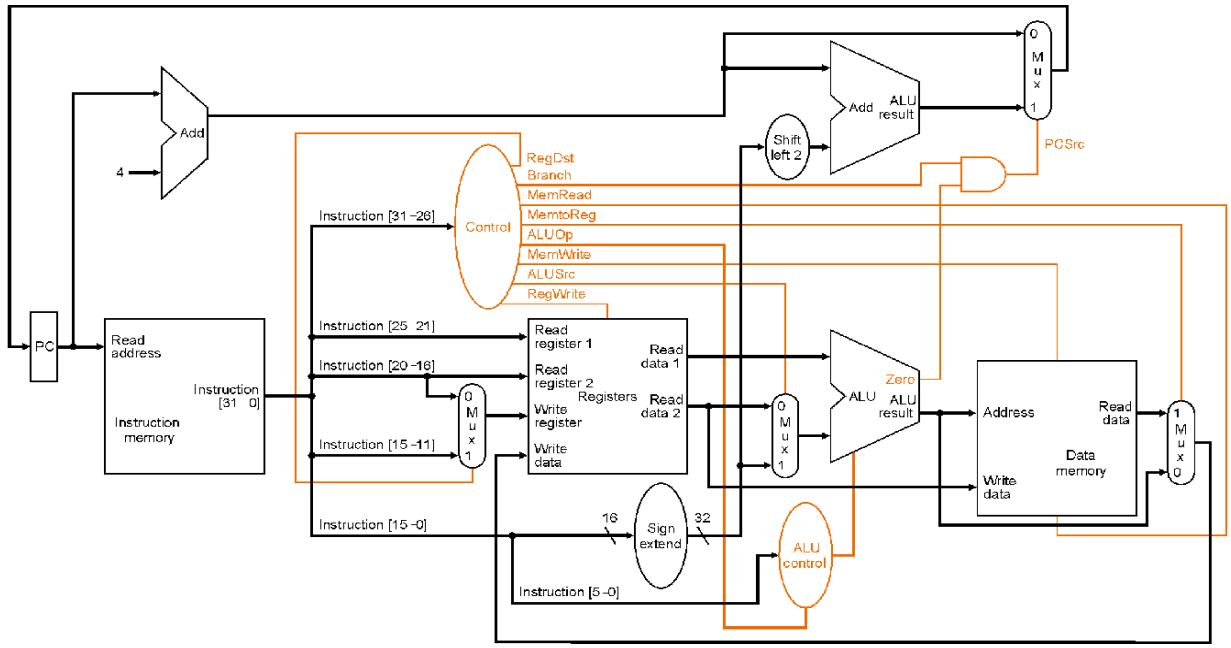
a) **(3 pts)** Convert the assembly code above to a DFG where the nodes of the graph represent the instructions.



b) **(3 pts)** Using the DFG above, convert the assembly program above to SSA form, meaning each register only gets a single assignment and each register uses the alpha notation as we did in class. This question does not require any back-edges

```
strcpy: subi $αp, $αp, 4
        lw   $α0, 0($αp)
        add  $α0, $zero, $zero
L1:     add  $α1,$a1,$α0
        lb   α2,0($α1)
        add  $α3,$a0,$α0
        sb   $α2,0($α3)
        addi $α0,$α0,1
        bne  $α2,$zero,L1
        lw   $α0,0($αp)
        add1 $αp,$αp,4
        jr   $ra
```

8. **(6 pts)** Consider the following single-cycle datapath with control signals:



Suppose the ALUSrc signal is stuck at 0. What type of MIPS instruction will no longer work properly, and why?

Any instruction that uses memory such as Load word and store word would no longer work, as the values from those instructions can't be provided to the ALU, and we wouldn't be able to determine memory addresses.

9. **(9 pts)** The latencies of the components of the single-cycle datapath are:

Memory Read: 9 ns
Memory Write: 14 ns
Register file: 5 ns
ALU: 7 ns (all other components have negligible latency)

a) **(3 pts)** If the clock-cycle time is made as small as possible, what is the clock frequency?

For slowest instruction: $1/35 * 10^9$ Hz

b) **(3 pts)** Assume a program with instruction mix of 23% loads, 13% stores, 19% branches, 2% jumps, 43% ALU. What is the average clock cycle time for this program?

1.5 * 0.23 + 1 * 0.13 + 1.5 * .19 + 2 * 0.2 + 1 * .43 = 1.59

1.59 * 35 ns = 55.65 ns.

c) **(3 pts)** We have to choose between two performance optimizations: either a new ALU (latency 5 ns) or a new register file (latency 4 ns). Which would be a better choice? Justify your answer.

The new ALU, as with a slow instruction, such as one that accesses memory, 2 ns is saved vs saving 1 ns with a new register file.

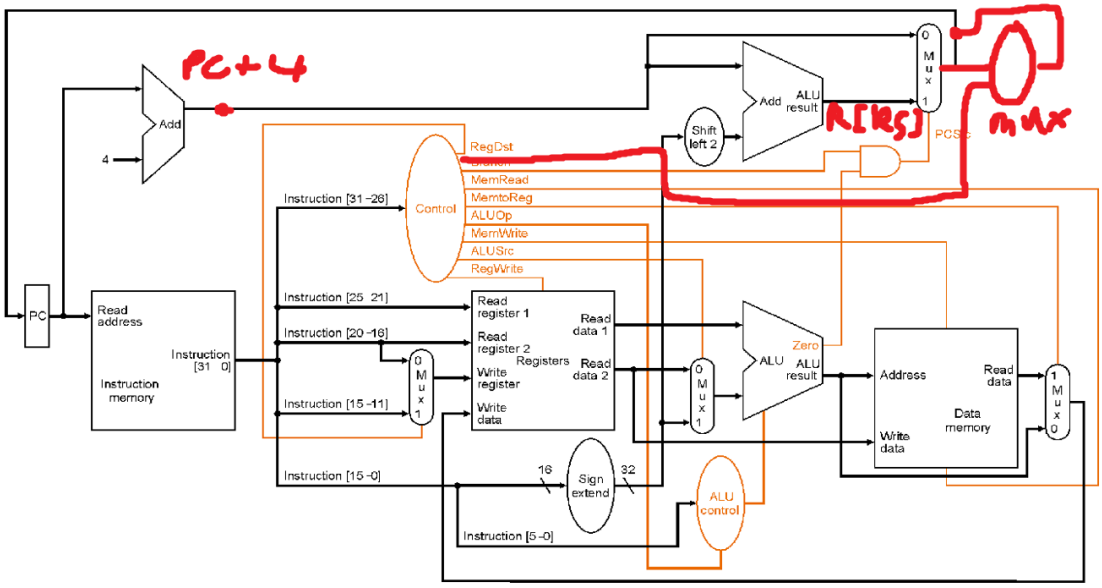
10. **(6 pts)** To perform jump to a function call, implement ***jalrn*** instruction in the MIPS single-cycle datapath. The format of ***jalrn*** instruction is shown below.

jalrn rs,rd => *# R[rd] ← (PC+4);*
 # PC ← Memory[R[rs]]

The ***jalrn*** instruction is coded as an R-type instruction.

On a single cycle datapath, show what changes are needed to support ***jalrn*** instruction. You should only add wires and muxes to the datapath; do not modify the main functional units themselves (the memory, register file and ALU). Your solution should not lengthen the clock cycle. Assume that the ALU, Memory and Register file all take 2ns, and everything else is instantaneous.

On the diagram, write (next to the signal's name) values of all control signals required for the ***jalrn*** instruction.



11. **(6 pts)** Suppose we wanted to add the auto decrement addressing mode to the MIPS ISA – e.g. lw R1, 100(R2--). This saves an instruction every time we observe a load followed by a decrement of the address register. Is this a good idea? Consider only performance, and assume that we have to increase cycle time by 11% to accommodate the new instruction, that 15% of our instructions are loads, that we can apply this change to 42% of all loads, and that the CPI doesn't change.

Original execution time = 0.85 + 0.15(.42 * 2 + .58), // Multiply by 2for the instruction not saved

= 1.063

After addressing mode time = (0.85 + 0.15(.42 + .58)) + (0.85 + 0.15(.42 + .58)) * .11

= 1.11

So it is not worth it.

12. **(6 pts)** Consider an ISA with fixed length instruction format of 16 bits. The various types of instructions supported by ISA are shown below in Table.

Instruction Type	Description
Two-operand	<OPCODE> <REG> <REG> <OPCODE> <MEM> <REG> <OPCODE> <REG> <MEM>
Three-operand	<OPCODE> <REG> <REG> <REG> <OPCODE> <REG> <REG> <IMM>

Assume processor having 15 registers and it supports 1MB of byte addressable memory. Answer the following questions based on the ISA described:

a) **(2 pts)** Show the instruction format with bit layout.

Each register takes 4 bits, as there are 15, the minimum we can have is 0000-1110

Each MEM supports up to 20 bits, for 1 mb of byte addressable memory

For <OPCODE> <REG> <REG>, there are 4 bits in each REG and 8 for OPCODE

For <OPCODE> <MEM> <REG> there are 4 bits for the REG, and 12 to split between OPCODE and MEM

<OPCODE> <REG> <MEM>, Same as above

<OPCODE> <REG> <REG> <REG>, 4 for each reg, 12 for all, that leaves 4 for the OPCODE

<OPCODE> <REG> <REG> <IMM> knowing there are 4 for each reg, that leaves 8 to split between the OPCODE and IMM.

b) **(2 pts)** How many two operand instructions are possible?

We have 8 bits for instructions in <OPCODE> <REG> <REG>, so 2^8 or 256

For <OPCODE> <MEM> <REG> and <OPCODE> <REG> <MEM>, it is possible to have 1-11 bits for OPCODE, depending on how many bits we use for MEM, so it is the summation of 2^1 + … + 2^11

So we have a grand total of 256 + 2 * the summation of 2^1 + … + 2^11

c) **(2 pts)** Assume that there are 4, three operand instructions available. What is the range of integer numbers (in decimal) can be assigned to the IMM (i.e. Immediate) operand? Assume numbers are represented in 2's complement form.

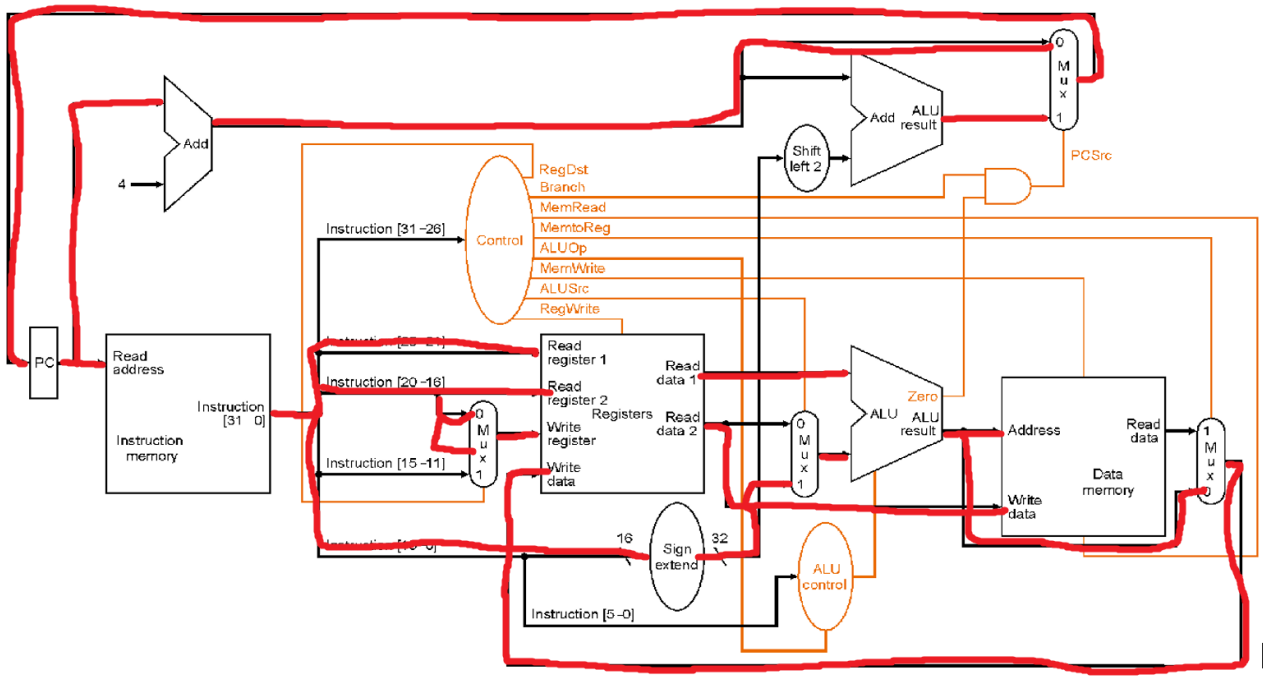
Since there are 4 instructions, we only need 2 bits to represent them. With the 8 bits for registers, that leaves 6 for the IMM. Therefore with signed integers it can be from in the range -32… 0… 31

13. **(6 pts)** We would like to supplement the MIPS single-cycle processor design to support a new instruction: 'sw_dec' as described below.

sw_dec rt, immd(rs) => #Mem[Reg[rs] - sign extended(immd)] ← Reg[rt]
#Reg[rs] ← Reg[rs] - sign extended(immd)

a) **(3 pts)** Draw the design changes on the single cycle data path and write down the control signal values when executing the newly added sw_dec instructions. Assume the processor still executes other instructions (LW, SW, BEQ).

Add a 3rd mux input



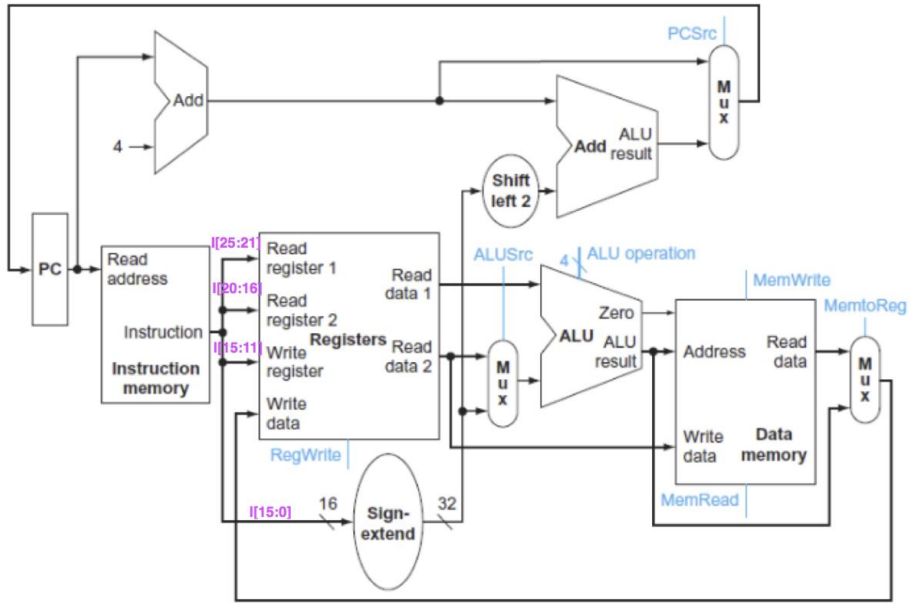
b) **(3 pts)** Would a similar instruction for 'lw_dec' be less, more, or about the same amount of difficulty to implement on this pipeline? Briefly explain in 1-2 sentences why.

Less difficulty, we can just use the base diagram.

14. **(6 pts)** In this problem, you are given an incomplete datapath – that is, there are some MIPS instructions that the processor does not properly execute. You can assume that all needed control signals are available.

(Note that the markings of the inputs into the register file indicate which bits of the instruction are used for that input. For example, I[25:21] means thae 21st thrugh 25th bits (inclusive) of the instruction)

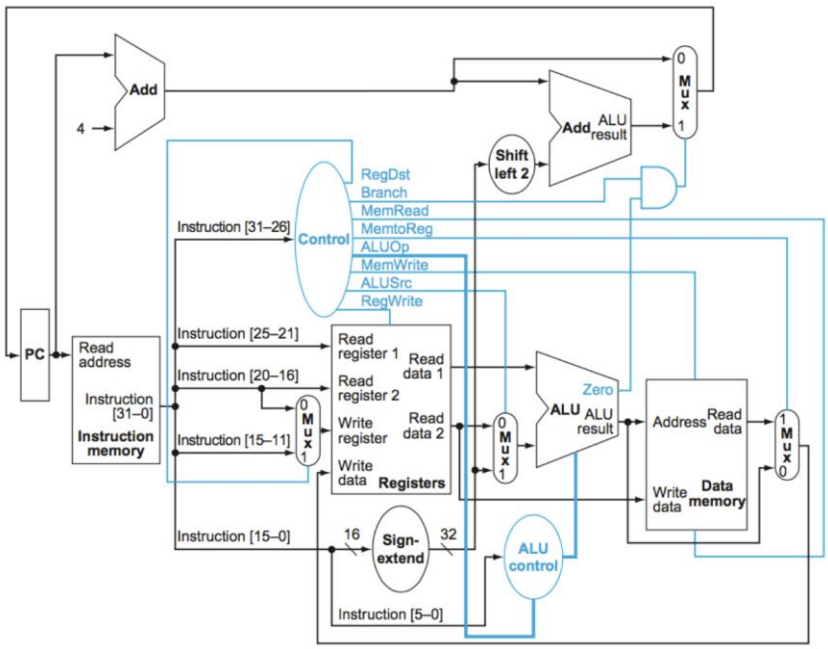
Does this processor properly execute all I-type instructions?(Please consider only the following subset of I-type instructions - lw,sw,addi,beq,ori) Separately list down the I-type instructions which can be executed properly and those which can't. If there are any problems, describe the fixes in the text box below. Assume all control signals are available.You can also assume the sign extension unit handles the zero extension for andi/ori instructions.



It can support **lw, sw, and beq** but not **addi, ori.** It needs another mux after the instruction memory path before Registers for sign extend

15. **(4 pts)** For these problems, consider the following figure of the datapath, and the given table which shows the delay of the microarchitectural components. Calculate the time it takes to execute the following instructions (assume this is the only instruction executed by the processor). You may assume that the delay of anything not mentioned in the table (including wires and control signals) is zero.

Unit	Inst. Mem.	Reg.	ALU	Data. Mem.	Mux	Add	Ctrl. Unit
Delay (ps)	220	100	160	220	30	50	80



a) **(1 pt)** Load Word (e.g. lw $t1, 8($t0))

Instruction mem + registers + mux + ALU + data memory + mux

220 + 100 + 30 + 160 + 220 + 30 = 760

b) **(1 pt)** Store Word (e.g. store $t0, 12($t1))

Instruction mem + registers + mux + ALU + data memory + mux

220 + 100 + 30 + 160 + 220 + 30 = 760

c) **(1 pt)** Add (e.g. add $t2, $t0, $t1)

Instruction mem + registers + mux + add + data memory + mux

220 + 100 + 30 + 50 + 220 + 30 = 650

d) **(1 pt)** Branch (e.g. beq $t0, $t1, loop)

Instruction mem + registers + mux + Ctrl unit + data memory + mux

220 + 100 + 30 + 80 + 220 + 30 = 680

16. **(7 pts)** Given a number *n*, how would you calculate sum of first *n* integers using a stack ISA? Apart from push, pop and add instructions, what are the other instructions that you would need? *(Hint: Think about how we handle loops in assembly)*

    Your answer should be a program written using a Stack based ISA. You can assume that the number *n* is already on the stack and that it's the only element on the stack at the beginning. By the end of your program, the answer (sum of first *n* integers) needs to be the only element on the stack.
    In simple words, you're implementing a program which pops the value of *n* from the stack and pushes the sum of first *n* integers onto the stack.

High-level

```
Procedure sum(int n):
        int ret;
        For (i = 1;  i <= n; i++) {
                ret += i
        }
        Return ret
```

Solution:

push an accumulator (0)

during each procedure call/loop: pop twice and update accumulator with n (acc = acc + n), check if n = 0, if isn't 0 push it back in with acc, otherwise just push the acc

```
push acc(0)
LOOP:
pop ;;
pop ;;
add acc, acc, n
cmp n, 0
bne END
push acc

END:
Push n
Push acc
```