Exam 1

Graded

Student

Gabriel Joseph Ragy

Total Points

79 / 100 pts

Question 1

Switch Statement & Jump Table — **10** / 10 pts

1.1 (no title) — **4** / 4 pts

✔ **– 0 pts** Correct

– 1 pt Incorrect logic on an instruction, e.g. bgt $a0, $t1, default versus bgt $t1, $a0, default

– 1 pt Misses a minor instruction, e.g. lw $t0, 0($t0) on default case

– 2 pts Misses a critical instruction, e.g. lw $t0, 0($t0) or jalr $t0, $zero

– 2 pts Incorrect logic on multiple instructions

– 2 pts Does not handle default case correctly (k > n)

– 2 pts Does not handle jump table case correctly (k < n)

– 3 pts Incorrect, but nearly correct on one case

– 4 pts Completely incorrect or empty

1.2 (no title) — **6** / 6 pts

✔ **+ 3 pts** Listed a benefit for conditional branching (e.g. no mem access, small number of branches = efficiency, varied comparison logic/non contiguous values)

✔ **+ 3 pts** Listed a benefit for jump table (e.g. large number of branches, contiguous comparison value, efficient clock cycles, reusable addresses)

+ 3 pts Two ambiguous reasons listed/need explanation

+ 1.5 pts Ambiguous reason listed/needs explanation

+ 0 pts Completely incorrect or empty

yup! :)

Question 2

Datapath

12 / 12 pts

+ 12 pts Correct

- ✔ **+ 3 pts** I1 correct **(lea $sp, Label_A)**
- ✔ **+ 3 pts** I2 correct **(lw $sp, 0($sp))**
- ✔ **+ 3 pts** I3 correct **(addi $sp, $sp, -1)**
- ✔ **+ 3 pts** I4 correct **(sw $t0, 0($sp))**

+ 1 pt Some incorrect format or item for I1 (wrong offset, swapping source and destination, adding a third or fourth argument to the instruction)

+ 1 pt Some incorrect for I2

+ 1 pt Some incorrect for I3

+ 1 pt Some incorrect for I4

+ 0 pts Incorrect

Question 3

Interrupts 1

4 / 10 pts

3.1 (no title)

1 / 4 pts

– 0 pts Correct; everything EXCEPT: EI, execute device code, $t0->stack, mode->user, $a->stack

– 1 pt 1 incorrect

– 2 pts 2 incorrect

- ✔ **– 3 pts** 3 incorrect

– 4 pts Incorrect; 4+ incorrect

3.2 (no title)

0 / 3 pts

+ 2 pts Mentioned IVT contains handler addresses / is indexed by the vector

+ 1 pt Mentioned device outputs vector / ETR

- ✔ **+ 0 pts** Incorrect / Blank

3.3 (no title)

3 / 3 pts

- ✔ **+ 2 pts** Correct RETI behavior
- ✔ **+ 1 pt** Unique property: atomicity

+ 0 pts Incorrect

Question 4

Interrupts 2 — **8** / 8 pts

4.1 (no title) — **4** / 4 pts

- ✔ **+ 4 pts** Correct
- **+ 4 pts** Correct but no or insufficient explanation
- **+ 4 pts** Incorrect

4.2 (no title) — **4** / 4 pts

- ✔ **+ 4 pts** Correct
- **+ 4 pts** Correct with error carried over from 4.1
- **+ 4 pts** Correct but insufficient or no explanation
- **+ 4 pts** Incorrect

Question 5

Performance Metrics 1 **3.5** / 10 pts

5.1 **(no title)** **2** / 2 pts

+ **0.5 pts** Incorrect: Math Error

✔ + **1 pt** 56100 nanoseconds

✔ + **1 pt** Equation/Work

+ **0.5 pts** Incorrect: Major Math Error

+ **0.5 pts** Incorrect: Answer given in wrong units

+ **0 pts** Incorrect

5.2 **(no title)** **1.5** / 2 pts

✔ + **0.5 pts** Incorrect: Math error

+ **1 pt** 49500 nanoseconds

✔ + **1 pt** Equations/Work

+ **0.5 pts** Incorrect: Major math error

+ **0.5 pts** Incorrect: answer given in wrong units

+ **0 pts** Incorrect

5.3 **(no title)** **0** / 3 pts

+ **3 pts** Correct usage of speedup formula, incorrect answer due to 5.1/5.2

+ **3 pts** Correct answer: 56100/49500 (17/15)

+ **1 pt** Equation/Work

+ **1.5 pts** Incorrect: Swapped old time and new time

+ **1.5 pts** Not in fraction form

+ **1.5 pts** Incorrect: Math Error

+ **0.5 pts** Incorrect: Major Math Error

✔ + **0 pts** Incorrect

5.4 (no title) **0** / 3 pts

+ 3 pts **(56100 - 49500) / 56100 - (6600 / 56100) - (2/17)**

+ 3 pts Correct usage of improvement formula, incorrect answer due to 5.1/5.2

+ 1 pt Equation/Work

+ 1.5 pts Incorrect: Swapped old time and new time

+ 0.5 pts Incorrect: Major Math Error

+ 1.5 pts Not in Fraction Form

✔ + 0 pts Incorrect

Question 6

Performance Metrics 2 **10** / 10 pts

6.1 (no title) **5** / 5 pts

✔ + 5 pts Correct

+ 0 pts Incorrect

+ 2 pts Did not count LEA after HALT in static calculation

+ 3 pts Minor Calculation error

6.2 (no title) **5** / 5 pts

✔ + 5 pts Correct

+ 0 pts Incorrect

+ 2 pts Counted the LEA after HALT

+ 3 pts Minor Calculation error

Question 7

Structs and Endianness **8** / 10 pts

7.1 **(no title)** **6** / 6 pts

✔ **+ 6 pts** BE EF 2D 6F
? ? ? ?
13 24 B9 CF
02 14 20 24
0C 67 ? 24

+ 6 pts BE EF 2D 6F
13 24 B9 CF
02 14 20 24
0C 67 ? 24
? ? ? ?

+ 3 pts Flipped the two rows for the long

+ 0 pts Incorrect

7.2 **(no title)** **0** / 2 pts

✔ **+ 0 pts** Incorrect

+ 2 pts Correct - considering ans from 7.1 to not penalize propagated error

7.3 **(no title)** **1** / 1 pt

✔ **+ 1 pt** Correct

+ 0 pts Incorrect

7.4 **(no title)** **1** / 1 pt

✔ **+ 1 pt** Correct

+ 0 pts Incorrect

Question 8

Datapath Delay

Resolved **7** / 10 pts

– **0 pts** Correct (PC stable + wire + memory + wire + IR hold = 314ps)

– **7 pts** Missing/including 3+ extra components

– **5 pts** Missing/including 2 extra components

✔ – **3 pts** Missing/including 1 extra component

– **2 pts** Incorrect Math

– **10 pts** Blank/Incorrect

– **7 pts** Almost correct computation of critical path, but then answer doesn't use critical path

Regrade Request **Submitted on: Mar 02**

n.8: I was removed points for 2 missing components when I am only missing one. I think I deserve 2 more points.
For n.5.3 and 5.4, I have the equations swapped... is there any way to get any points for these please ?

I have given back 2 points for q8 as it was incorrectly graded, you were correct that you were only missing 1 component.

In questions 5.3 and 5.4, using the incorrect formula is not acceptable as it will not give you the correct answer or work.

Reviewed on: Mar 02

Question 9

Stack Calling Convention **6.5** / 10 pts

9.1 (no title) **1** / 1 pt

✔ **+ 1 pt** Correct

+ 0 pts Incorrect

9.2 (no title) **1** / 1 pt

✔ **+ 1 pt** Correct Answer (no)

+ 1 pt Incorrect answer (yes) but gives correct justification

+ 0 pts Incorrect

9.3 (no title) **1** / 2 pts

✔ **– 0 pts** Correct

✔ **– 1 pt** Fails to mention that the $sp could move around unpredictably, so that is why the fixed $fp anchor point is needed.

– 2 pts Incorrect

9.4 (no title) **3.5** / 6 pts

– 0 pts Correct

– 6 pts Incorrect

✔ **– 0.5 pts** Incorrect registers - main()

✔ **– 1.5 pts** Incorrect / missing explanation - main()

– 0.5 pts Incorrect registers - foo()

– 1.5 pts Incorrect / missing explanation - foo()

✔ **– 0.5 pts** Incorrect registers - bar()

– 1.5 pts Incorrect / missing explanation - bar()

💬 main - t0
bar - s0

Question 10

CPI **10** / 10 pts

✔ **+ 10 pts** Correct

+ 8 pts Incorrect answer, work correct

+ 6 pts Correct answer, no work shown

+ 4 pts Incorrect answer, calculated execution times correct

+ 0 pts Incorrect

Question 11

Assumptions

0 / 0 pts

✔ **+ 0 pts** Correct

Question 12

Appendix

0 / 0 pts

✔ **+ 0 pts** Correct

+ 0 pts Incorrect

Question 13

What is the code from Canvas?

0 / 0 pts

✔ **+ 0 pts** Correct

+ 0 pts Incorrect

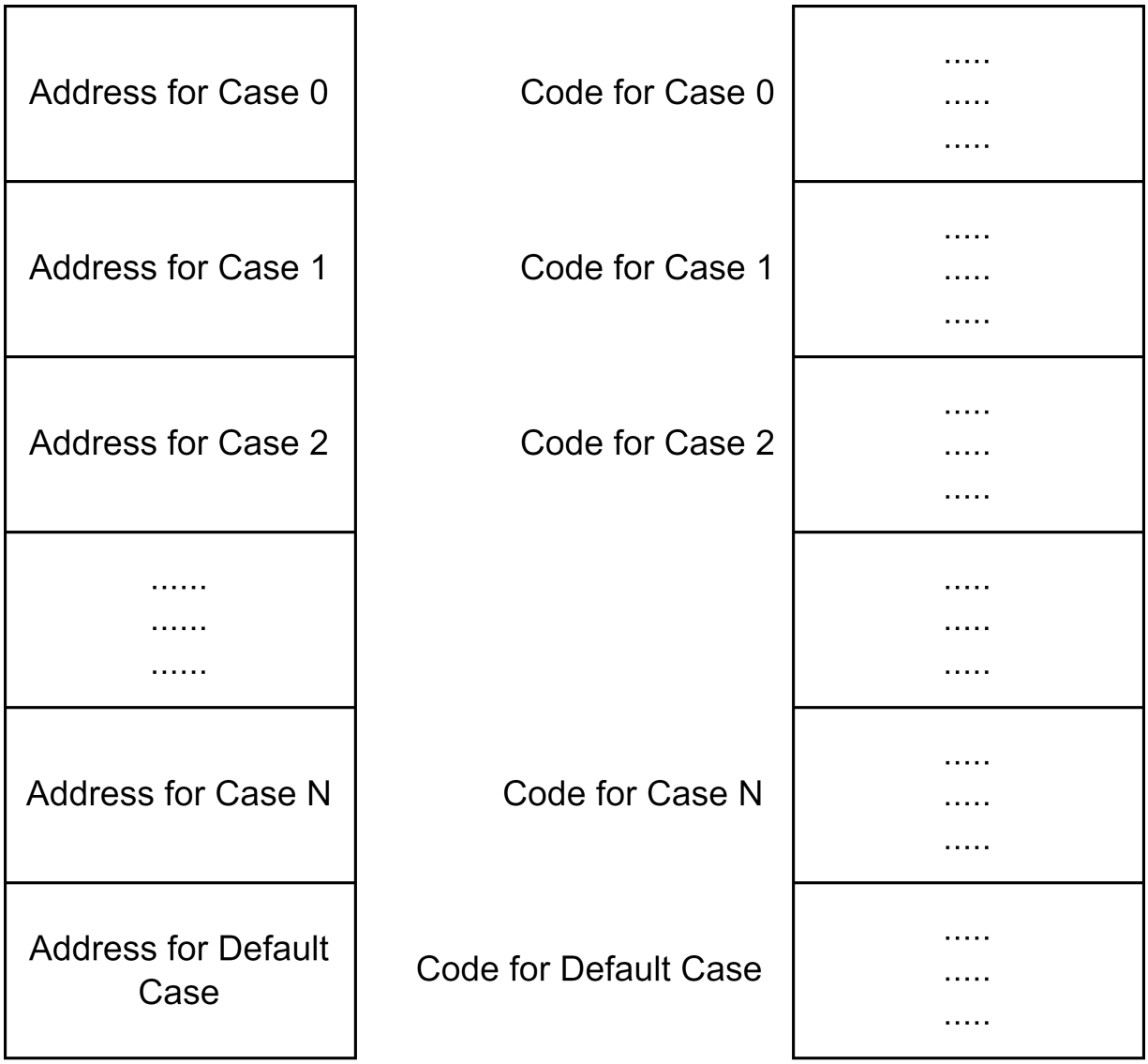
Q1 Switch Statement & Jump Table

10 Points

High-level languages provide a “switch” statement that looks as follows:

```
switch(k) {
      case 0:
      case 1:
      case 2:
      ...
      case N:
      default:
}
```

The compiler writer Kaylia knows that “k” can take non-negative contiguous integer values from 0 to N during execution, with any value greater than N going to the default case. She decides to use a jump table data structure (implemented as an array indexed by the value contained in k) to hold the start address for the code for each of the case values as shown below:



Jump Table

Assume we are using the **LC-2200 instruction set architecture**, which can be found in the appendix.

Q1.1
4 Points

Assume the **base address of the jump table** is stored in the register **$t0**, the value of **N** is stored in register **$a0**, and the **value of k** is stored in the register **$t1**. Help Kaylia out by writing a series of LC-2200 instructions that work

```
!we assume the switch statement is in main, which is why we don't back up
any registers before jumping to the case

bgt $t1, $a0, greater
beq $zero, $zero, smaller
greater:
addi $at, $a0, 1
add $at, $t0, $at
beq $zero, $zero, jump
smaller:
add, $at, $t0, $t1
jump:
lw $at, 0x($at)
jalr $at, $zero

case k: !example case
! code
beq $zero, $zero, done



done:
!code after switch statement
```

Q1.2
6 Points

Instead of using a jump table, her friend suggests using a series of conditional branch instructions to compile the switch statement. List **one reason** why conditional branches could be better, and **one reason** why jump table could be better. Explain your answer, clearly stating your assumptions.

One reason why a jump table could be better is because it can compute the jump address in constant time O(1) instead of having to test if k is equal to every single case using if statement logic. The second method takes O(n) time. However, if there aren't many cases, it would be better to use a series of conditional branch instructions as Kaylia wouldn't have to make a jump table for a couple of cases. It all depends on the amount of cases!

Q2 Datapath

12 Points

You are given 26 clock cycles of control signals which is a subsection of the execution macro states for some assembly code. Fill out the 4 lines of LC-2200 assembly instructions that correspond to the control signals. Note that the clock cycles corresponding to the fetch macrostate have been omitted.

```
I1: _____
I2: _____
I3: _____
I4: _____
Label_A .fill 0x2200
```

Cycle 4: DrPC, LdA
Cycle 5: DrOFF, LdB [Offset = Label_A]
Cycle 6: ALU=add, DrALU, WrREG, RegSel = $sp

Cycle 10: DrREG, LdA, RegSel = $sp
Cycle 11: DrOFF, LdB [Offset = 0]
Cycle 12: ALU=add, DrALU, LdMAR
Cycle 13: DrMEM, WrREG, RegSel = $sp

Cycle 17: DrREG, LdA, RegSel = $sp
Cycle 18: DrOFF, LdB [Offset = -1]
Cycle 19: ALU=add, DrALU, WrREG, RegSel = $sp

Cycle 23: DrOFF, LdA [Offset = 0]
Cycle 24: DrREG, LdB, RegSel = $sp
Cycle 25: ALU=add, DrALU, LdMAR
Cycle 26: DrREG, WrMEM, RegSel = $t0

```
lea $sp,  Label_A
```

```
lw $sp, 0x0($sp)
```

```
addi $sp, $sp, -1
```

```
sw $t0, 0x0($sp)
```

Q3 Interrupts 1
10 Points

Q3.1
4 Points

After the execute macrostate, you can either enter the INT macrostate or continue execution as normal. Assuming you enter the INT macrostate, select from the following the steps that occur in the INT macrostate.

- [x] $k0 ← PC
- [x] Save current mode in the system stack
- [ ] Enable Interrupts
- [x] Execute device code
- [x] Retrieve address of handler from interrupt vector table
- [x] ACK INT by asserting INTA
- [x] Switch mode to kernel
- [ ] Save $t registers to stack
- [ ] Disable interrupts
- [x] PC ← handler address received from the vector table
- [ ] Switch mode to user
- [ ] Save $a registers to stack
- [ ] Receive interrupt vector from device on the data bus
- [x] Update $sp to point to the system stack

Q3.2
3 Points

Describe how we retrieve the interrupt handler code after an interrupt has been acknowledged.

First it goes into the INT macrostate, then to the INT handler. We need to do many things such as disabling interrupts while we handle delicate tasks such as saving PC to $k0, then enable them once again before RETI back to the previous state of the machine.

Q3.3
3 Points

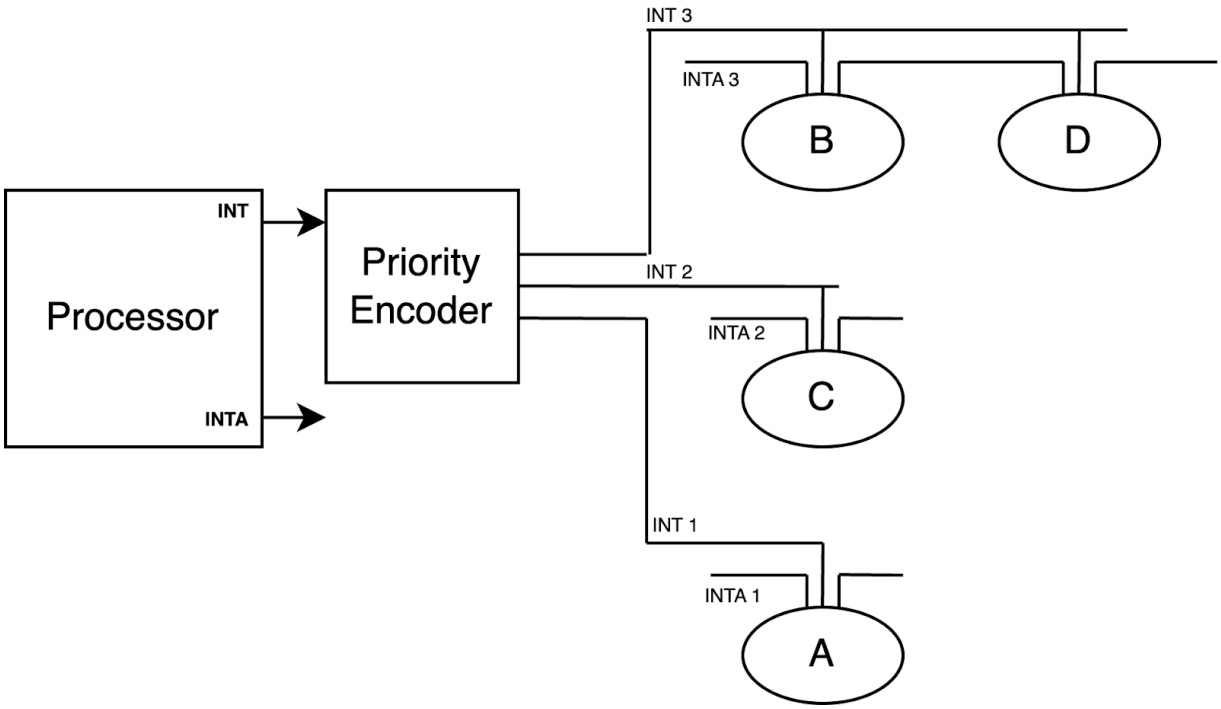
What does RETI do, and what is its unique property that necessitated its introduction as a new instruction?

It makes the INT handler return from the interrupt and makes the mode change to the previous mode. It is an atomic instruction and cannot be cut off by another interrupt. Its like return but the machine version of return.

Q4 Interrupts 2
8 Points

Consider an architecture that has 3 interrupt priority levels. The interrupt handler for every device enables interrupts before executing the device-specific code.

The below diagram shows four devices. Smaller numbers represent a higher priority level. B is electrically closer to the processor than D. The INTA line is chained through devices B and D.



A schedule of signals is shown below. Each handler takes 1000 cycles.

Time	Interrupted By	INTA
0	C	
100	D	
200		INTA
300	B	
400	A	

Time	Interrupted By	INTA
500		INTA
600		INTA
....		
900		INTA

Q4.1
4 Points

In what order are the three devices acknowledged? Explain your answer.

CABD. First, C and D interrupt then INTA is asserted and C is acknowledged as it has higher priority. Then, B and A interrupt and INTA is asserted thereafter but since A has higher priority it is acknowledged first. Then, INTA is asserted again, and we now go in order of electrical proximity which is B then D. So the final order is CABD

Q4.2
4 Points

Assuming that these are the only interrupts, in what order do the interrupt handlers complete? Explain your answer.

The interrupt handlers complete in order of priority so A then C, then in order of electrical distance so B then D. Hence, the order in which they complete is ACBD.

Q5 Performance Metrics 1

10 Points

Kaylia has a program that consists of the following 4 lines, but repeats 550 times.

```
lw $t0, 0($a0)
nand $t1, $t0, $t1
addi $t1, $t1, 8
sw $t1, 0($a0)
```

In this implementation, the lw instruction takes 5 clock cycles. The nand instruction takes 3 clock cycles. The addi instruction takes 4 clock cycles. The sw instruction takes 5 clock cycles. Each clock cycle takes 6 nanoseconds. Show your work for the following questions.

Q5.1

2 Points

What is the execution time of this program?

Execution time = (5+3+4+5) * 550 * 6ns = 17 * 550 * 6 = 56100ns

Q5.2

2 Points

If Kaylia changes her implementation so the lw instruction now takes 3 clock cycles, what is the new execution time? Round to the nearest nanosecond.

Instead of taking 17 cycles per loop, it takes 14 cycles. ex time = 14 * 550 * 6 = 46200ns

Q5.3

3 Points

What is the speedup achieved after this change? Leave your answer as a fraction.

speedup = (oldtime-newtime)/oldtime = (17-14)/17 = 3/17 (I simplified and cancelled out all the common terms)

Q5.4
3 Points

What is the improvement in execution time after this change? Leave your answer as a fraction.

improvement in exec time = oldtime/newtime = 17/14 (I simplified and cancelled out all the common terms)

Q6 Performance Metrics 2

10 Points

Consider the following program that contains 600 instructions:

```
I1:
I2:
I3:
.....
I330:
I331: LEA
I332:
.....
I343:
I344: COND BR to I330
I345:
I346:
.....
I599: HALT
I600: LEA
```

LEA instruction occurs exactly twice in the program as shown. Instructions 330-344 constitute a loop that gets executed 150 times. All other instructions execute exactly once.

Q6.1

5 Points

What is the static frequency of LEA instruction? Show your work for credit and leave your answer as a fraction.

We can see LEA twice in the code, so the static freq of LEA is 2/600 = 1/300

Q6.2

5 Points

What is the dynamic frequency of LEA instruction? Show your work for credit and leave your answer as a fraction.

First we find the total lines executed = 329 + 15 * 150 + 255 = 2834. LEA executes 150 times only because the LEA on line 600 doesn't execute with HALT preceding it.
Dynamic Freq of LEA = 150/2834

Q7 Structs and Endianness

10 Points

For the struct defined below, show how a smart compiler might pack the data to minimize wasted space, whilefollowing alignment restrictions. Pack in such a way that each member is naturally aligned based on its data type. Assume the compiler will not reorder fields of the struct in memory. Assume a char is 1 byte, an int is 4 bytes, and a short is 2 bytes, and a int64_t is 8 bytes. Moreover, assume the CPU architecture is little-endian and supports load word (LW), load byte (LB), and load half word (LHW), where a memory word is 4 bytes.

```
struct x {
   int a;
   int64_t b;
   char d;
   short e[1];
}

data = {
    0xBEEF2D6F,
    0x021420241324B9CF,
    0x24,
    {0x0C67}
};
```

Q7.1
6 Points

Assume struct data is saved at 0x1000. Fill in the table for the following:

+3	+2	+1	+0	Starting Address
				0x1000
				0x1004
				0x1008
				0x100c
				0x1010

a.) What hex values are stored in the addresses 0x1003 to 0x1000 (the first row)?

0xBE 0xEF 0x2D 0x6F

b.) What hex values are stored in the addresses 0x1007 to 0x1004 (the second row)?

? ? ? ?

c.) What hex values are stored in the addresses 0x100B to 0x1008 (the third row)?

0x13 0x24 0xB9 0xCF

d.) What hex values are stored in the addresses 0x100F to 0x100C (the fourth row)?

0x02 0x14 0x20 0x24

e.) What hex values are stored in the addresses 0x1013 to 0x1010 (the fifth row)?

0x24 ? 0x0C 0x67

Q7.2
2 Points

CPU issues: LW R1, MEM[0x1008]. Show the content of R1.

0xCFB92413

Q7.3
1 Point

You are a clever programmer who knows the architectural details and how the compiler will pack the variables to optimize on space. Reorder the elements of struct x such that it will result in optimal space usage.

```
int64_t b;
int a;
short e[1];
char d;
```

Q7.4
1 Point

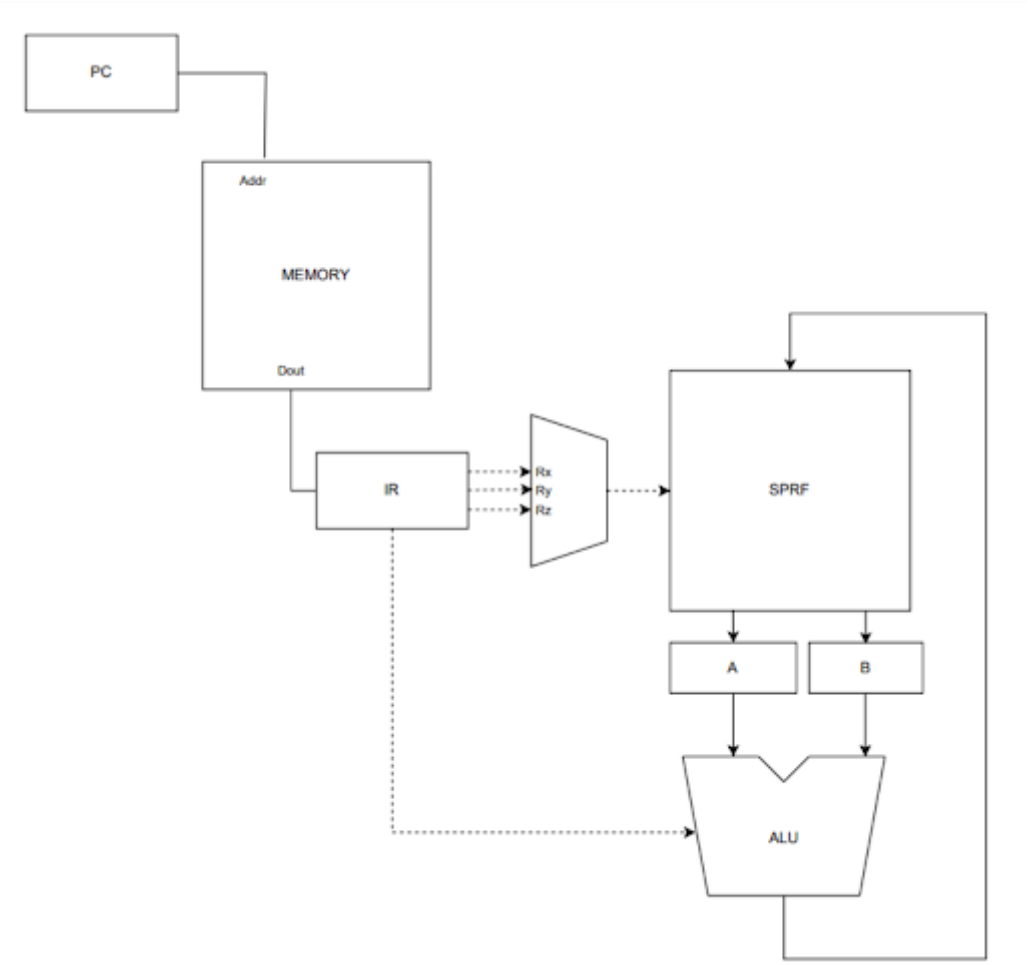
How many bytes do you save by reordering the elements of struct x?

We save 4 bytes by swapping int64_t and int, then another byte by swapping short and char, so 5 bytes are saved in total

Q8 Datapath Delay

10 Points

Consider the datapath with the delays shown below. Solid lines are data lines; dashed lines are control lines. All times are given in picoseconds - e.g., reading from or writing to the register file takes 2 picoseconds (2x10^-12s). What is the minimum clock cycle in ps?



Component	Delay (ps)	Notes
Wire	3	
Alu	4	Combinational
Register hold + setup	5	For IR, PC, A, B
Register output-stable	3	For IR, PC, A, B
Memory	300	Level Logic

Component	Delay (ps)	Notes
SPRF Access (read or write)	4	
MUX	3	

minClock = max(clockcyle1, clockcyle2, clockcyle3) = Register hold + setup 5 + Register output-stable 3 + Wire 3 + Memory 300 = 311ps. I didn't bother calculating the other clock cycles because they take significantly less time than 300 seconds, so it is without a doubt that the cycle where memory is used will be the bottleneck affecting the minClock.

Q9 Stack Calling Convention

10 Points

Quick reminder of the LC-2200 software convention for registers:

```
$a0-$a2: parameter passing
$s0-$s2: callee saves if need be
$t0-$t2: caller saves if need be
$v0: return value
$ra: return address
$at: target address
$sp: stack pointer
$fp: frame pointer
```

Q9.1

1 Point

What is a frame pointer?

The frame pointer is a register that holds the address on a "local stack" of the values saved by a subroutine/method. It is similar to the head of an array in memory where the array holds the registers saved to the stack but only for that specific program/subroutine. Frame pointer is in essence similar to stack pointer, but on a deeper level within the stack. Different functions will have frames all within the stack, with their variables saved there.

Q9.2

1 Point

Does the value of $fp change during the execution of the current procedure?

No typically you don't change $fp's value unless changing procedures/methods/subroutines. This is because you want your local variables to be accessible. However, before calling a new routine/method, you save your $fp on the stack then branch.

Q9.3
2 Points

Given the existence of the stack pointer, what is the benefit provided by the introduction of the frame pointer?

Frame pointer allows for subroutines to make nested calls because we now have a reference point for all the values saved by a previous subroutine/method in the stack. This reference is the frame pointer, and allows for more flexibility in how registers are backed-up when calling subroutines/methods.

Q9.4
6 Points

We have a program where main() calls foo() and foo() calls bar(). Assume that main() is the root function.

main() uses $s0, $s1 and $t0.
foo() uses $s0 and $t0.
bar() uses $s0, $t0, $t1, and $t2.

These functions use these registers before and after they make their function calls. If there are no other function calls, and the $s# and $t# registers are only used as indicated above, what registers will each of main(), foo(), and bar() save to the stack? Explain why.

The registers that callees save are the registers that they will use. foo uses $s0 and $t0, and will therefor save them before writing to them. bar uses $s0, $t0, $t1, and $t2, and will save them on the stack before writing to them. Similar to the professors analogy of using the pans that nobody knows if they're clean or dirty, you back up the registers you want to use before using them! If the caller backs up its used registers instead of the callee doing it, this would be inefficient because maybe the callee doesn't even need those registers!

Q10 CPI
10 Points

The following table shows the CPI for three instructions A, B, C, implemented by a processor. An architect determines that she can reduce the CPI of A to 5, with no change to the CPIs of the other two instruction types, but with an increase in the clock cycle time of the processor.

Assume that all the workloads that execute on this processor use 20% of A, 50% of B, and 30% of C types of instructions.

Type	CPI
A	10
B	8
C	3

What is the maximum permissible increase in clock cycle time that will make this architectural change still worthwhile? Show your work for credit.

First we must find the improvement in execution time after this change = (0.2*10+0.5*8+0.3*3)/(0.2*5+0.5*8+0.3*3) = 1.1695
We can therefor increase the clock cycle time by 16.95% and the architectural change will still be worthwile.

Q11 Assumptions
0 Points

State any assumptions you made here.

8 byte based addressing for int64_t data type.

Q12 Appendix

0 Points

LC 2200 ISA with an additional LEA and BGT instruction		
Mnemonic Example	**Opcode (Binary)**	**Action Register Transfer Language**
add add $v0, $a0, $a1	0000	Add contents of reg Y with contents of reg Z, store results in reg X. RTL: $v0 ← $a0 + $a1
nand nand $v0, $a0, $a1	0001	Nand contents of reg Y with contents of reg Z, store results in reg X. RTL: $v0 ← ~($a0 && $a1)
addi addi $v0, $a0, 25	0010	Add Immediate value to the contents of reg Y and store the result in reg X. RTL: $v0←$a0+25
lw lw $v0, 0x42($fp)	0011	Load reg X from memory. The memory address is formed by adding OFFSET to the contents of reg Y. RTL: $v0 ← MEM[$fp + 0x42]
sw sw $a0, 0x42($fp)	0100	Store reg X into memory. The memory address is formed by adding OFFSET to the contents of reg Y. RTL: MEM[$fp + 0x42] ← $a0
beq beq $a0, $a1, done	0101	Compare the contents of reg X and reg Y. If they are the same, then branch to the address PC+1+OFFSET, where PC is the address of the beq instruction. RTL: if($a0 == $a1) PC ← PC+1+OFFSET
jalr jalr $at, $ra	0110	First store PC+1 into reg Y, where PC is the address of the jalr instruction. Then branch to the address now contained in reg X. Note that if reg X is the same as reg Y, the processor will first store PC+1 into that register, then end up branching to PC+1. RTL: $ra ← PC+1; PC ← $at Note that an **unconditional jump** can be realized using **jalr $ra, $t0,** and discarding the value stored in $t0 by the instruction. This is why there is no separate jump instruction in LC-2200.
halt	0111	Halt the machine
bgt bgt $a0, $a1, done	1000	Compare the contents of reg X and reg Y. If the value in reg X is greater than the value in reg Y, then branch to the address PC+1+OFFSET, where PC is the address of the bgt instruction. RTL: if($a0 > $a1) PC ← PC+1+OFFSET
lea lea $a0, stack	1001	An address is computed by sign-extending bits [19:0] to 32 bits and adding this result to the incremented PC (address of instruction + 1). It then stores the computed address into register DR. RTL: $a0 = MEM[stack]

Q13 What is the code from Canvas?
0 Points

Not answering this may result in a zero for this exam!

HonorlockWorks!