HW03: RISC-V ISA and Function Calls
CSE 30321 Computer Architecture
Names: Aaron Wang and Ethan Little

Preamble:

1. Copy this assignment to your Google Drive folder and enter your answers in the boxes provided; **each empty box should be filled in with an answer**. You can either type your answers directly or insert images as long as they are legible. For solutions that require code, use a fixed-width font (Consolas preferred) no smaller than 10 points with proper indentation. Save your solutions as a single PDF file and upload them to Gradescope.

2. You are encouraged to work in groups of 2. Please type your names at the top of this document.

Problem 1: RISC-V Encoding and Instructions

Here is a supplement with instructions explaining the steps to encode or decode an SB- or UJ-type instruction. You don't have to use the supplement, but you might at least find helpful the explanations of how to test your work using QtRVSim.

Question 1
Translate the following RISC-V instructions to their hexadecimal encoding. The first one is done for you as an example.

Tip 1: check your work using QtRVSim – instruction encodings are in hex in the Memory window. Make sure you are comfortable doing this by hand using the RISC-V Reference Data Sheet ("Green Sheet") and/or zyBooks Table, since you will be doing this on exams. You will have both as a reference, but otherwise the exams are closed notes/book/computer.

Important reminder: per the syllabus, you are allowed to consult (but not copy from) generative AI tools like ChatGPT *if you cite it as a source* and do not ask it for solutions directly. However, be aware these tools often give *garbage answers* when asked about RISC-V instructions (and other Comp Arch topics). For example, for the `addi` below, GPT-4 made the following mistakes: 1) it converted the immediate (32) to 0b000001000000 (64) in one place, and to 0b000000000100 (4) shortly thereafter; 3) it converted the resulting binary to an intermediate binary (in groups of 4) that did not match; 4) finally, the hex encoding it gave at the end did not match (2) or (3). I told GPT-4 it was wrong, and it doubled down, making further mistakes. **Be very careful when consulting these resources!**

Q1 Table

RISC-V Human Readable Instruction	Hexadecimal Encoding
`addi x9, x9, 32`	`0x02048493`

`sw x25, 0xCC(x10)`	`0x0D952623`
`lb x4, 0x33(x10)`	`0x03350203`
`add x11, x0, x19`	`0x013005B3`
beq x9, x20, LBL LBL is 15 instructions before the beq (e.g., if beq is at address 0x1000 (4096), LBL is at address 0xFC4 (4036)).	`0xFD4482e3`

Question 2
Translate the following hex-encoded RISC-V instructions back into their human readable form. For any SB- or UJ-type instructions, express your answer like the beq in the table in Q1.

Q2 Table

RISC-V Human Readable Instruction	Hexadecimal Encoding
`xor x12, x5, x8`	`0x0082c633`
`auipc x20, 0x0270a`	`0x0270aa17`
`srli x11, x9, x4`	`0x0044d593`
`jal x1, LBL` LBL is at x1+0xc8	`0x0c8000ef`

Question 3
Explain what the following code does. Write comments next to the code explaining what the original, high-level language code would do. A line-by-line translation of each instruction will not receive full credit.

Note 1: please attempt this problem without first running it in the simulator, and then check your reasoning with the simulator

Note 2: labels and variable names are deliberately obfuscated so as not to give away what the code does, this is not generally good coding practice.

Q3 Annotated Code

```
.text
// code to initialize variables, x5 is a counter, x3 address of Z, x4 = Y =
9 and x10 is the sum which starts at 0
_start:
            li x5, 0
            la x3, Z
            la x4, Y
            lw x4, 0x0(x4)
            li x10, 0
// Increases the pointer by 1, loads it and add it to the sum
// for loop range(0:9) add each byte to the sum (x10)
LBL1:
            add x6, x3, x5
            lb x7, 0x0(x6)
            add x10, x10, x7
            addi x5, x5, 1
            bne x4, x5, LBL1

            ebreak

.data
Y:
      .word 9
Z:
      .word 0x23161548, 0x38063242, 0x00000027
```

Problem 2: Performance Practice

Question 1
Complete Exercises 1.15.14 a, b, and c in zyBook Ch 1.15 *using Amdahl's Law*. **Use the form of Amdahl's Law presented in class**:

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \dfrac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

You should be able to see the zyBooks solutions for this problem, so check your calculations using Amdahl's Law against those calculations.

Q1 Answer with Work
Answer in the box below, and show all work (don't forget to include units). You can work by hand and scan or take a photo of your work (just make sure it is large enough to read clearly).

a)

$Fraction_{enhanced} = \frac{70}{250} = 0.28$ fp instruction fraction

$Speedup_{enhanced} = \frac{1}{1-0.2} = 1.25$x speed up

$Speedup_{overall} = \frac{1}{(1-0.28)+\frac{0.28}{1.25}} = 1.059$x speed up

$Time_{reduced} = 1 - \frac{1}{1.059} = 5.6\%$

b)

$Fraction_{enhanced} = \frac{250-70-85-40}{250} = 0.22$ INT instruction fraction

$Speedup_{overall} = 1.25 = \frac{1}{(1-0.22)+\frac{0.22}{x}}$

$x = \frac{0.22}{0.8-(1-0.22)} = 11$x speed up

$Time_{reduced} = 1 - \frac{1}{11} = 90.9\%$

c)

$Fraction_{enhanced} = \frac{40}{250} = 0.16$ br instruction fraction

$Speedup_{enhanced} = \infty$x speed up

$Speedup_{overall} = \frac{1}{(1-0.16)+0} = 1.19$x speed up

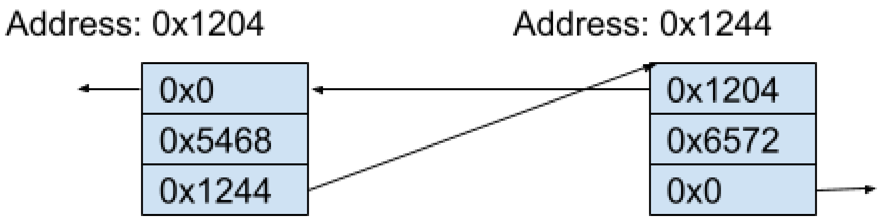
$Time_{reduced} = 1 - \frac{1}{1.19} = 16\% < 20\%$ So, NO

Problem 3: RISC-V Function Calls

For this problem, you will write a short program to implement support functions for the upcoming doubly linked list sort.

Functions and Requirements

1. Swap
   a. Eventually, you will write code to sort the list using a simple sorting algorithm. The algorithm will swap adjacent list elements – adjacent meaning they point to each other, they may not be adjacent in memory.
   b. You must pass the address of the first element to swap into the function.
      i. Additional arguments and any return value are up to you.
      ii. Consider the figure from HW02. To swap these two elements, the address 0x1204 would be passed into the swap function.



   c. The swap **must** occur via next/previous pointer manipulation, you may not move the “data” field between elements.
   d. Be sure to pay careful attention to the order of previous/data/next in each element (same as HW02) and that you are consistent with this ordering throughout your code.
   e. You must handle the following edge cases, both of which should have no effect:
      i. When there is only a single list element,
      ii. When swap is called on the last list element.
   f. And this edge case:
      i. When swap is called on the head - your “global” head pointer should be updated when this happens (i.e., to what was previously head→next).

2. Delete
    a. In a future assignment, you will write code to delete specific elements in the linked list using this delete function.
    b. You must pass the address of the element to delete into the function.
        i. Additional arguments and any return value are up to you.
    c. You must handle the following edge cases:
        i. When there is only a single list element. Your "global" head pointer should be updated to point to 0x0000 when this happens.
        ii. When there are > 1 list elements and delete is called on the head. This results in a new head; your "global" head pointer should be updated.
        iii. When there are > 1 list elements and delete is called on the tail. This results in a new tail (you do not need to maintain a tail pointer).

Testing your Functions

Each test case will be in a **separate .S file**. You will duplicate your code 3 times, and modify each program to match Test Cases 1-3 below.

Option 1: start with your HW02 code.

Option 2: copy the .data section from Using linked_list_print_skeleton.S, which will automatically place the linked list in memory.

Test Case 1 (multiple tests on complete HW02 list)

Write test code to call swap on the element with value 0x5468[1], followed by the element with value 0x206E, and finally on the element with value 0x7472. Call delete on the element with value 0x6572, followed by the element with value 0x6973, and finally on the element with value 0x7472.

Test Case 2

First, update the .data section: set num_elements to 1 and remove all but the first element from the array (or from the dummy list if using Option 2). Call swap on the single list element.

Test Case 3

First, update the .data section: set num_elements to 1 and remove all but the first element from the array (or from the dummy list if using Option 2). Call delete on the single list element.

[1] I am referring to elements by value, since their addresses may change, **but your functions should take in an address as described earlier.** Tip: work the swaps and deletes out on paper so you know what addresses to pass to the functions.

Problem 3 Deliverables (Code and Screenshots)

RISC-V assembly program

Copy your assembly code to one group member's dropbox on the student machines.
*List the **full path** to the directory where your code is. Place all 3 test case .S files in the same directory:*

```
/escnfs/home/awang27/esc-courses/sp25-cse-30321.01/dropbox/HW03
```

*In addition, **copy and paste** the entirety of the code in the box below **for test case 1 only**.*
***This should not include any of the code to print the list to the terminal**.*

```
   la x8, linked_list_head
   or x10, x8, x8 // set head
   // Write test code to call swap on the element with value 0x5468,
   li x12, 0x1204 // set arg for element to swap
   jal x1, swap
   // followed by the element with value 0x206E,
   li x12, 0x1304 // set arg for element to swap
   jal x1, swap
   // and finally on the element with value 0x7472.
   li x12, 0x1384 // set arg for element to swap
   jal x1, swap
   // Call delete on the element with value 0x6572,
   li x12, 0x1244 // set arg for element to delete
   jal x1, delete
   // followed by the element with value 0x6973,
   li x12, 0x12c4 // set arg for element to delete
   jal x1, delete
   // and finally on the element with value 0x7472.
   li x12, 0x1384 // set arg for element to swap
   jal x1, delete


swap:
      // new head returned at x10
      lw x11, 0x0(x12) // prev
      lw x13, 0x8(x12) // next
      beq x13, x0, end_swap // if next is null, no swap to be done return
      lw x14, 0x8(x13) //next_next
      beq x11, x0, skip_head //if prev is null (aka head) skip prev.next =
next
      sw x13, 0x8(x11) //prev.next = next
skip_head:
      beq x13, x0, skip_tail //if next_next is null (aka adjacentis tail)
skip next_next.prev = cur
      sw x12, 0x0(x14) //next_next.prev = cur
skip_tail:
      sw x12, 0x8(x13) // next.next = cur
      sw x14, 0x8(x12) // cur.next = next_next
      sw x13, 0x0(x12) // cur.prev = next
      sw x11, 0x0(x13) // next.prev = prev
      bne x11, x0, end_swap // if prev is not null we good
      or x10, x13, x13 // else head = next
end_swap:
      jr x1

delete:
      // need element to delete at x12
      // need head at x10
      lw x11, 0x0(x12) // prev
      lw x13, 0x8(x12) // next
      bne x11, x0, del_not_head //branch if prev is not null
      or x10, x13, x13 // since prev null, head = next
      beq x0, x0, del_tail //skip to next part
del_not_head:
      sw x13, 0x8(x11) // otherwise prev.next = next
del_tail:
      beq x13, x0, end_delete // if next is null return
      sw x11, 0x0(x13) // otherwise next.prev = prev
end_delete:
      jr x1
```

Print the List, Screenshot Output

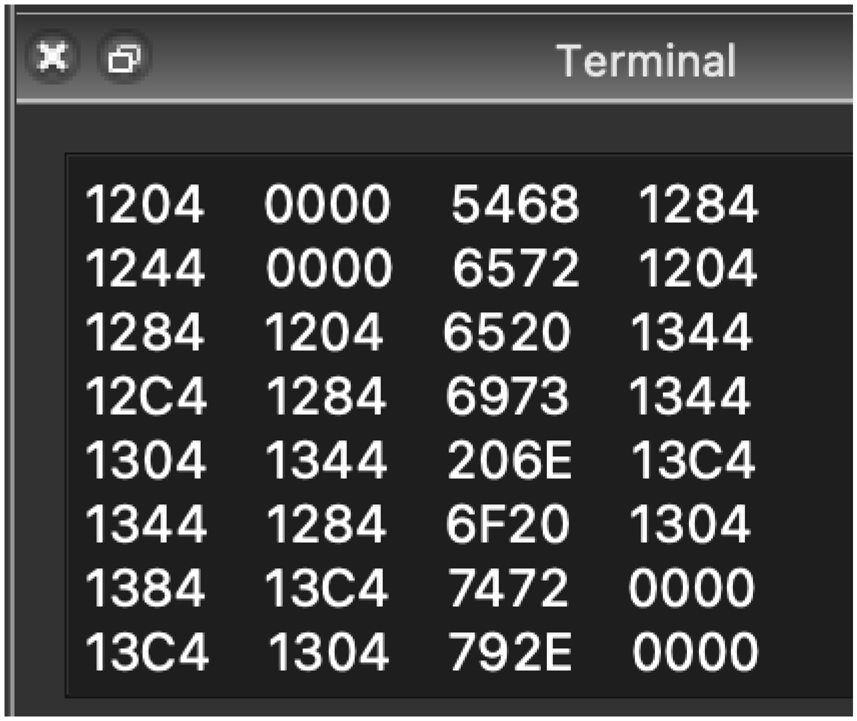
To demonstrate your code works, combine it with the same skeleton code from HW02 that will print the contents of the linked list for each test case. Instructions are here:
Using linked_list_print_skeleton.S

You should create each test case as a separate file. The code will largely be the same, differing only in the memory organization and test cases. Though tedious, this will be the simplest way to demonstrate your code works correctly (and make it easy for graders to test your code).

*List the **full path** to the directory where your code is:*

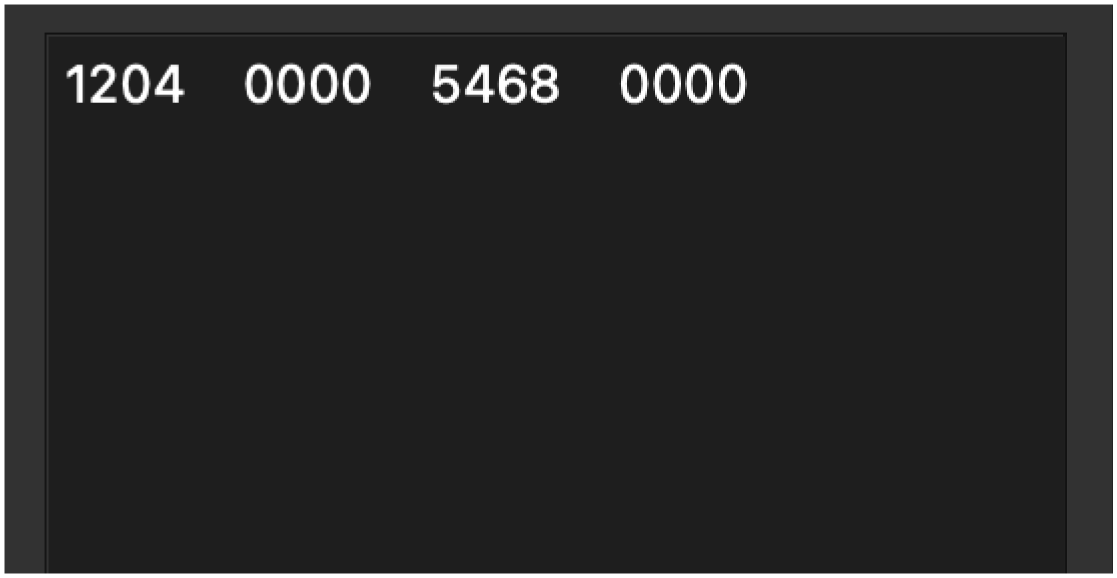
```
/escnfs/home/awang27/esc-courses/sp25-cse-30321.01/dropbox/HW03
```

Copy a screenshot of the terminal window from QtRVSim showing the final list state for Test Case 1:



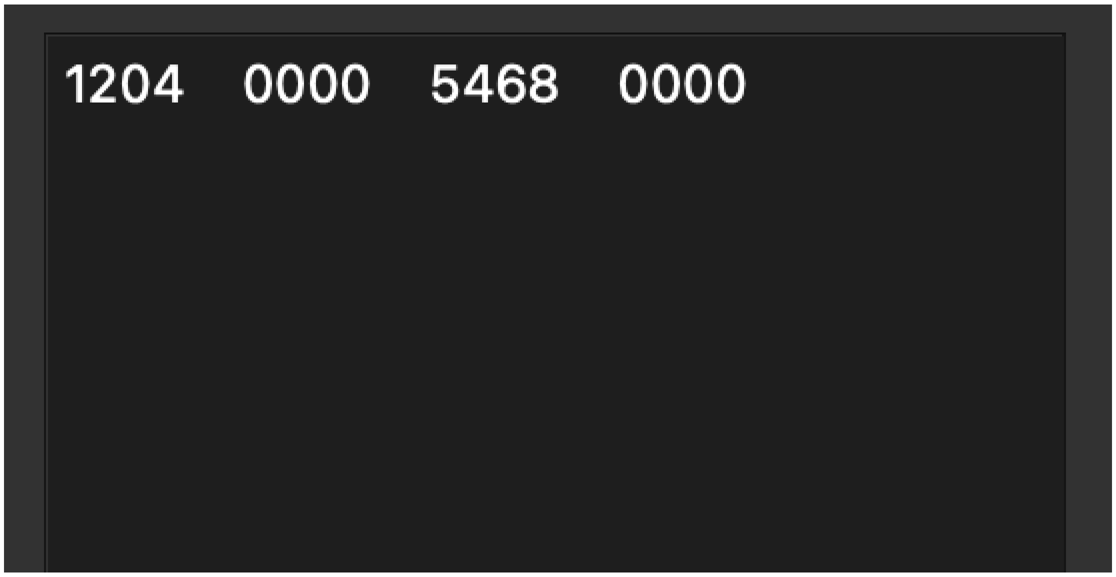
```
1204   0000   5468   1284
1244   0000   6572   1204
1284   1204   6520   1344
12C4   1284   6973   1344
1304   1344   206E   13C4
1344   1284   6F20   1304
1384   13C4   7472   0000
13C4   1304   792E   0000
```

Copy a screenshot of the terminal window from QtRVSim showing the final list state for Test Case 2:



```
1204   0000   5468   0000
```

Copy a screenshot of the terminal window from QtRVSim showing the final list state for Test Case 3:



```
1204    0000    5468    0000
```

What to Turn In

Fill in the boxes on the previous pages with answers to all the questions. Save your Google Doc as a PDF and upload it to **Gradescope** for grading. Note Gradescope can be accessed via our Canvas site, or by visiting gradescope.com. Make sure all code you write has sufficient comments so that the graders can clearly identify the parts of the program. Use a fixed-width font (Consolas is preferred) for your code with proper indentation. Below is a checklist for this assignment:

Problem 1 (44 points)

	Deliverable	Points
1.	Q1 table	15
2.	Q2 table	15
3.	Q3 Annotated Code	14

Problem 2 (15 points)

	Deliverable	Points
1.	Q1 Answer with Work, 5 points for each calculation	15

Problem 3 (65 points)

	Deliverable	Points
1.	RISC-V assembly program	40
2.	Screenshots	25