HW02: Performance and Assembly Programming
CSE 30321 Computer Architecture
Name(s): 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 with a partner (though it is not required). DM @adingler if you are looking for a partner or post in #cse-30321-sp25
    a. Please type your name(s) at the top of this document.

Problem 1: Performance

Question 1
Assume an instruction set with 3 classes of instructions; each instruction class requires a different number of clock cycles to complete as defined in the table below.

Integer	4 cycles
Memory	9 cycles
Branches	3 cycles

The datapath is used to execute 3 programs sequentially with instructions mixes per the table below:

	Integer instructions	**Memory instructions**	**Branch instructions**
Program 1	3×10^9	3×10^9	0.5×10^9
Program 2	2×10^9	8×10^9	7×10^9
Program 3	6×10^9	2×10^9	1.5×10^9

The processor's clock rate is 3.75 GHz (recall, giga is 10^9)

Assume that we want to reduce the total time it takes to execute these 3 programs by 2.5%. Is it possible to achieve this performance target by reducing only the CPI for the memory instructions? Justify your answer quantitatively by calculating the CPI for the memory instructions required to achieve this speedup. Recall our discussion from class about CPI for instructions or instruction classes, namely, they must be whole numbers and cannot be negative. Assume the clock rate will not change.

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).

$3 \times 10^9 + 2 \times 10^9 + 6 \times 10^9 = 11 \times 10^9$ Total Integer instructions
$3 \times 10^9 + 8 \times 10^9 + 2 \times 10^9 = 13 \times 10^9$ Total Memory instructions
$0.5 \times 10^9 + 7 \times 10^9 + 1.5 \times 10^9 = 9 \times 10^9$ Total Branch instructions

$4 \times 11 \times 10^9 + 9 \times 13 \times 10^9 + 3 \times 9 \times 10^9 = 188 \times 10^9$ Total OG Clock Cycles

If CPI for memory = 8
$4 \times 11 \times 10^9 + 8 \times 13 \times 10^9 + 3 \times 9 \times 10^9 = 175 \times 10^9$ Total Clock Cycles

$\frac{188}{175} = 1.074.$

By reducing CPI for memory instructions from 9 to 8, we have reduced the total time by more 7.4% (which is more that 2.5%). Thus it is possible to achieve this speed up by only reducing CPI for memory.
(Sidenote, clock rate was ignored, because it doesn't change.)

Question 2
Assume a program exists with the following mix of instructions:

Instruction Class	**% of Instruction Mix**	**Cycles per Instruction**
ALU	40%	4
Branch	15%	3
Memory	45%	6

For a given program:
- 5×10^{12} instructions are executed in total
- The processor's clock rate is 5.5 GHz (recall, giga is 10^9)

Computer architects are considering a processor redesign where the register file size will increase. This would reduce the number of Memory instructions to 75% of the original mix (e.g., if there are 1,000,000 memory instructions, there would now be 750,000). However, the reduction will come at the expense of increased CPI for ALU instructions. For example, in this situation perhaps we are removing some optimized datapath from the ALU to increase the register file size, causing all ALU operations to slow down, and we want to see if it is a worthwhile design change *for this benchmark.*

We wish to obtain a speedup of at least 1.25X for this program. What is the minimal *increased* CPI for ALU instructions required to achieve this speedup? If the CPI of ALU instructions cannot be increased, note that in your answer – this means the design change is not feasible.

Q2 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).

40% x 5 x 10^{12} = 2 x 10^{12} ALU Instructions
15% x 5 x 10^{12} = 0.75 x 10^{12} Branch Instructions
45% x 5 x 10^{12} = 2.25 x 10^{12} Memory Instructions

4 x 2 x 10^{12} + 3 x 0.75 x 10^{12} + 6 x 2.25 x 10^{12}
= 23.75 x 10^{12} Total OG Clock Cycles

For a 1.25x speed up, we want
23.75 x 10^{12} /1.0125 = 19 x 10^{12} Total Clock Cycles

MODIFIED
2.25 x 10^{12} x .75 = 1.6875 x 10^{12} Memory Instructions

C x 2 x 10^{12} + 3 x 0.75 x 10^{12} + 6 x 1.6875 x 10^{12}
= 19 x 10^{12} Total Clock Cycles
so
C = (19 x 10^{12} - 3 x 0.75 x 10^{12} - 6 x 1.6875 x 10^{12})/(2 x 10^{12})
C = 3.3125 CPI → 3 CPI (b/c CPI is whole number and we want faster).

As the required ALU CPI is faster than the original, this design change is not feasible.
(Sidenote, clock rate was ignored, because it doesn't change.)

Question 3

Consider the mix of instructions shown below.

Instruction Class	**Instruction Count**	**Cycles per Instruction**
Integer ALU	4 x 10^9	5
Branch	1 x 10^9	4
Memory	5 x 10^9	10

Floating point ALU	2×10^9	8

Computer architects are considering two options to improve the performance of this workload:

- **Option 1:** Lower the clock period by 1.3X at the expense of Memory instructions that will now require 12 clock cycles each instead of 10. (Lowering clock period by 1.3X means, for example, if the clock period is initially 1ns it becomes 0.769ns.)

- **Option 2:** Reduce the number of clock cycles required for each FP ALU instruction to 6 and reduce the number of FP instructions to 1.5×10^9, but at the expense of a clock period that is *increased* by 1.25X. (Increasing the clock period by 1.25X means, for example, if the clock period is initially 1ns it becomes 1.25ns.)

Which option, if any, is better than the original design? If any of the options are better than the original design, what speedup is obtained with the best option?

Q3 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).

Option 1
Integer ALU Instructions: 4×10^9 instructions
Branch Instructions: 1×10^9 instructions
Memory Instructions: $5 \times 10^9 \times 6 / 5 = 6 \times 10^9$ instructions
Floating Point ALU Instructions: 2×10^9 instructions
Integer ALU CPI: 5 Cycles
Branch CPI: 4 Cycles
Memory CPI: 10 Cycles
Floating Point ALU CPI: 8 Cycles
Clock Speed increase: 1.3x

$$\frac{10^9(4\cdot5+1\cdot4+5\cdot10+2\cdot8)C}{10^9(4\cdot5+1\cdot4+6\cdot10+2\cdot8)\frac{C}{1.3}} = 1.17\text{x speed up}$$

Option 2
Integer ALU Instructions: 4×10^9 instructions
Branch Instructions: 1×10^9 instructions
Memory Instructions: 5×10^9 instructions
Floating Point ALU Instructions: 1.5×10^9 instructions
Integer ALU CPI: 5 Cycles
Branch CPI: 4 Cycles
Memory CPI: 10 Cycles
Floating Point ALU CPI: 6 Cycles
Clock Speed decrease: 1.25x

$$\frac{10^9(4\cdot5+1\cdot4+5\cdot10+2\cdot8)C}{10^9(4\cdot5+1\cdot4+5\cdot10+1.5\cdot6)1.25\cdot C} = 0.87\text{x speed up}$$

Consequently, Option 1 is better than the original and option 2 is worse.
Option 1 is the best option.

Problem 2: RISC-V Programming
For this problem, you will write a short program to build a doubly linked list using elements from an existing array. You will write RISC-V assembly and test your implementation in QtRVSim.

Creating the Array
The values for your list will come from an 8-element array defined in memory via assembler directives. Add the following to your `.data` section to create this array. This is one of the test cases we will use to test your code, but you are strongly encouraged to test with other arrays. Use a ".org" assembler directive to place `num_elements` at memory address **0x1000**.

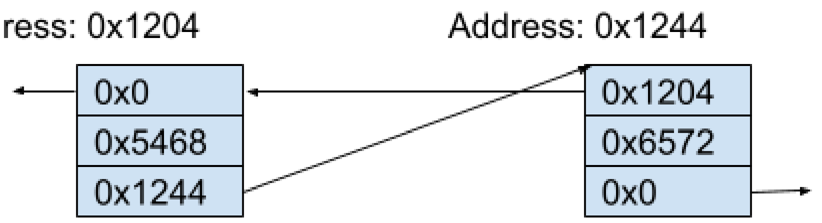
```
num_elements:
      .word  0x8
array_data:
      .word  0x5468, 0x6572, 0x6520, 0x6973, 0x206E, 0x6F20, 0x7472, 0x792E
```

Linked List Review and Requirements
If you are not familiar with doubly linked lists – or it's been awhile – please review using your previous coursework or online resources. The instructor and TAs are also happy to review with you in office hours.

A few requirements, **crucial for this assignment**:

1. We assume the list elements[1] may not be in contiguous memory locations. They can even be "out of order", spread throughout memory in any location. For example, the head of the list may be at a higher memory address (e.g., 0x800) than the next element (e.g., 0x700).
    a. You will use non-contiguous (but increasing) addresses for the list elements. Make the start of your linked list 0x200 bytes (i.e., 512 bytes) after the start of the array. Then, place each new list element 0x40 bytes (i.e., 64 bytes) after the start of the previously-created list element. For example, the array starts at 0x1004, so the head will be at 0x1204, the next element at 0x1244, the next element at 0x1284, etc.
2. We assume *within* a list element, its three components *are in contiguous memory locations*. Said differently: each list element will comprise three adjacent words.
3. Our doubly linked list will be organized in the following way: (i) the address of the previous element, (ii) the data associated with the element, and (iii) the address of the next element. Below is a graphical depiction of this organization with a doubly linked list of two elements. The first element is at address 0x1204 and has value 0x5468, and the second element is at address 0x1244 with value 0x6572. A value of 0 is used to indicate a null pointer. Note that all addresses are 32 bits, but leading 0's are omitted here to save space.

[1] You may see element, node, record, or struct when reviewing linked lists. I will try to be consistent with "element", where an element comprises three pieces: a pointer to the previous element, the data stored in the element, and a pointer to the next element.



You must follow the "previous element, data value, next element" ordering for the structure of your doubly linked list elements.

Building the List
Your main task for this assignment is to build the doubly linked list from the existing array. You will follow a structured approach to do this – this is intended both to give you a starting point, but also to make it easier for us to follow and grade.

1. Start by initializing registers for: i) the number of list elements; ii) any variables you need; iii) any constant values you need; iv) a pointer to the head of the list.
    a. You **must** maintain a copy of the head pointer in a register and leave it unchanged throughout the program. Make a copy of this immutable head pointer when you need to modify it. This will be important when printing the list, and also in future assignments when the list code is expanded.

2. Loop: you should use a loop that iterates over the array elements, adding one at a time to the linked list following the rules above.
    a. Use the label `insertLoop` for your loop.
    b. Your loop will repeatedly call an `insert` function[2] to insert each list element to the list, one at a time.
    c. Make sure the head element points back to NULL (address 0), and the tail element points forward to NULL (address 0).

3. `insert` function: your `insert` function should be defined below the loop. Pass the following into the function via three registers: i) the address of the array element to insert; ii) the address of the list element that is being created; iii) the address of the most recently inserted (previous) element.
    a. Register choice is up to you, and you are not required to follow register conventions on this assignment.
    b. All list element modifications (i.e., setting next, data, previous) must be done in the `insert` function, *not* directly in `insertLoop`, `main`, etc.

4. You must include detailed comments in your code. Explain what every few lines of code do. A direct translation is not appropriate, you should instead explain what the code does at a high level. For example, see the comments in this albaCore code we looked at in class.

[2] Recall: a function is created with a label and called using "jal xN, funName", and "jr xN" is used to return from the function. Using "j funName" (and/or a j to return) is not sufficient, as the function is not reusable.

5. You must **not** at any point hard code the number of array elements, the array elements themselves, or the address of either of these.
    a. In other words, this is not allowed:
    ```
    li x5, 0x1000          //x5 is address of num_elements
    ```
    Nor is this:
    ```
    addi x6, x0, 0x8       //x6 holds num_elements
    ```
    Instead, you can obtain the address of the number of array elements (or the array itself) using the `la` pseudoinstruction.

    For example:
    ```
    la x5, num_elements
    //x5 now contains the address of the number of array elements
    ```
    Now that we have the address of the number of elements, we can use a `lw` instruction to load the number of elements into a register:
    ```
    lw x6, 0x0(x5)
    //x6 now contains the number of array elements
    ```
6. Similar to #5, you must **not** hard code the address of any list element
    a. For example, do not hard code address 0x1204 (list head).
    b. You *may* hard code the various offsets: 0x40 for the inter-list spacing; 0x200 for the spacing between array and list.

Tips

1. *Start early*. As you know, writing assembly can be tricky, so budget enough time to complete the performance problems and the assembly coding problem.
2. Make a plan for writing your assembly code. Don't just sit down and start writing assembly. An outline with comments is a great way to start.
3. Try to keep it simple. Ignoring comments and assembler directives, my example solution is under 25 lines of assembly. You will not be graded on writing concise code for this assignment, but if you find yourself exceeding this by a large amount, rethink your approach and/or see the instructor or TAs.
4. See the last deliverable, below, and accompanying instructions explaining how to print the contents of your list to the QtRVSim terminal. This is a good way to test your code.

Problem 2 Deliverables

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:*

```
/escnfs/home/awang27/esc-courses/sp25-cse-30321.01/dropbox/HW02
```

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

```
.globl _start
.option norelax
.text
_start:
      la x1, num_elements
      lw x2, 0x0(x1) # num elements
      la x1, array_data # address of array_data
      addi x3, x1, 0x200 # pointer to head of list
      li x5, 0x0 # for loop i=0

      or x8, x1, x1 # address of array element to insert
      or x9, x3, x3 # address where list element is created
      li x10, 0x0 # address of last list element / pointer to tail
insertLoop:
      beq x2, x5, end # if len == i end loop
      jal x12, insert # call func insert
      addi x5, x5, 1 # update counter
      addi x8, x8, 0x4 # p = p+i; pointer for array of words increased
      or x10, x9, x9 # prev = curr
      addi x9, x9, 0x40 # curr = next element creation address
      j insertLoop # jump to top of loop
insert:
      lw x11, 0x0(x8) # set val to the word
      sw x11, 0x4(x9) # this.val = val
      sw x0, 0x8(x9) # this.nex = 0
      sw x10, 0x0(x9) # this.prev = prev
      beq x5, x0, first # if first element, dont do following
      sw x9, 0x8(x10) # prev.next = this
      jr x12 # jump back to part in loop
first:
      jr x12 # jump back to part in loop
end:
      ebreak
.data
.org 0x1000
num_elements:
      .word  0x8
array_data:
      .word  0x5468, 0x6572, 0x6520, 0x6973, 0x206E, 0x6F20, 0x7472, 0x792E
```

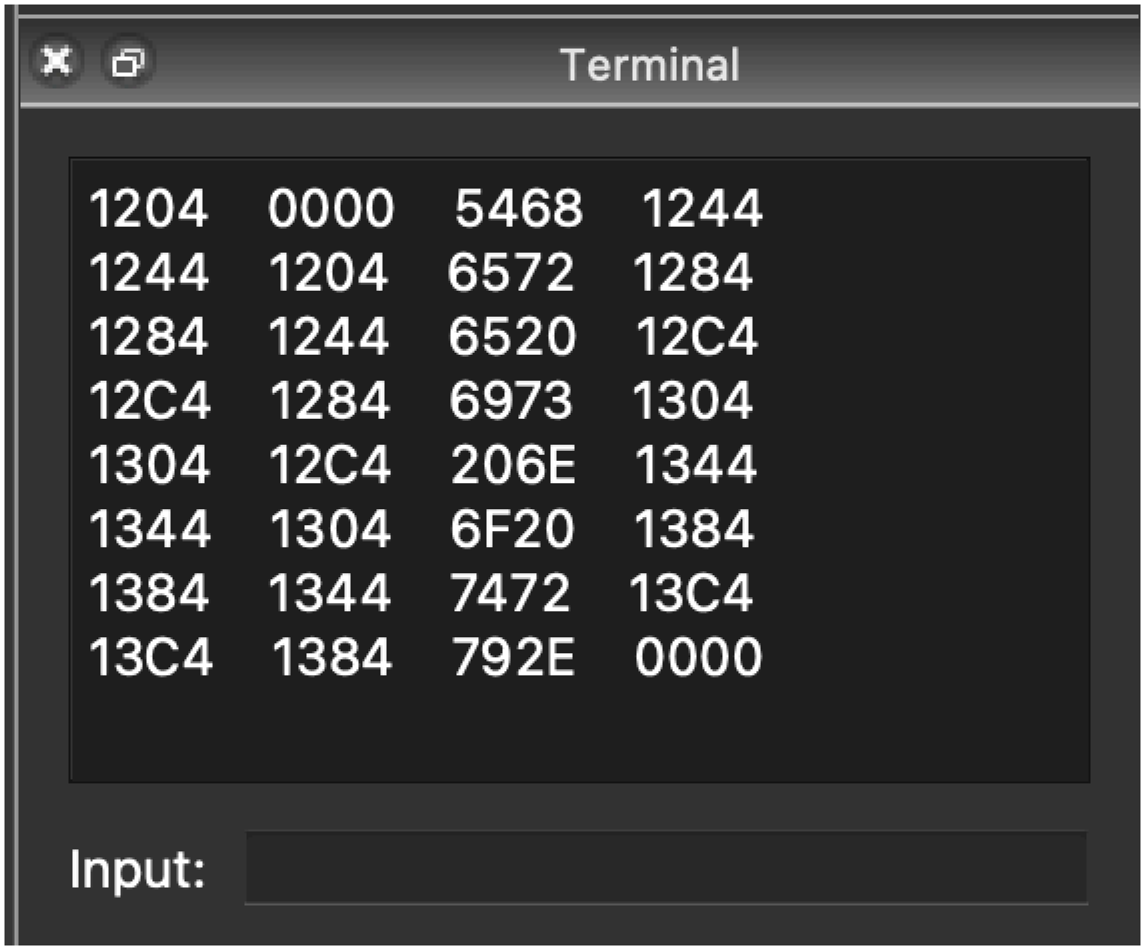
Print the List, Screenshot Output
To demonstrate your code works, you will combine it with some skeleton code that will print the contents of the linked list. Instructions for how to combine the "print_list" skeleton and your code, and how to run it, will be posted in the Drive Homeworks directory. This method will be used for future assignments as well.

You should maintain the original copy of your code, and create this as a separate file.

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

```
/escnfs/home/awang27/esc-courses/sp25-cse-30321.01/dropbox/HW02
```

Copy a screenshot of the terminal window from QtRVSim showing the list was built correctly:



```
1204   0000   5468   1244
1244   1204   6572   1284
1284   1244   6520   12C4
12C4   1284   6973   1304
1304   12C4   206E   1344
1344   1304   6F20   1384
1384   1344   7472   13C4
13C4   1384   792E   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 you showed your work for the performance problems and that all code you wrote has sufficient comments, as noted above, 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 (35 points)

	Deliverable	Points
1.	Question 1 Answer with Work	10
2.	Question 2 Answer with Work	10
3.	Question 3 Answer with Work	15

Problem 2 (50 points)

	Deliverable	Points
1.	RISC-V assembly program (path and code)	35
2.	Print the List, Screenshot Output (path and screenshot)	15