HW1: Performance and MIPS

Due February 13th , 2023, 11:59pm

extra credit available for early submissions!

There are 2 parts to this assignment: part 1: written exercises for Ch1; part 2: MIPS programming. Basic rules regarding assignment implementation and submission:

- Total points of this assignment: 100.
- **Team Allowed. Maximum 2 per Team**. You must include names and G#'s of **BOTH** group members in **ALL** submitted files**.**

Submission Instructions

- Submit to **gradescope**. A link to gradescope will be available from Blackboard. There will be two assignments on gradescope, one for part 1 and one for part 2. You will lose "off the top" points if you fail to follow the submission instructions.
- For Part1, follow the submission instruction from gradescope. You will need to upload a **PDF** and **specify** on which page we can find your answer for each question. Make sure your answer is legible.
    - **Up to -5 points** for incorrect format or not specifying pages.
- For Part 2, name your MIPS source code as **`cs465_hw1_username1[_userame2].asm`**
    - Here username is the first part of your GMU email address. Make sure to include both members' names if you have two people in the team.
    - For example: cs465_hw1_yzhong.asm
    - Do **NOT** submit it as a .pdf.
- For teams with two people:
    - For each part, only **one** team member should submit to Gradescope.
    - You won't be able to specify the team when you submit. Instead, you must **include names and G#'s of both group members** in all submitted files**.** We will use the provided group information to assign the same grade to both members.
- **Plagiarism is not permitted in any form. I enforce the university honor code.**

Part 1. Written Exercise for Performance (50%)

Notes:

- **A large portion of (or all) points will be taken off if you do not include detailed calculation in your answer.** You must show steps to justify your answer.
- **Answers must be legible**, especially if you scan to generate your submission.

1. (15 pts) Assume that we compile a program and get two different executables (A and B) for the same ISA.

1.1. (8 pts) The table below describes the performance of A and B when running on a 2GHz (i.e., 2x10^9 Hz) machine. Find the CPI for A and B respectively.

Version	Instruction Count	CPU Execution Time (sec)
A	5 x 10^9	4.5
B	4 x 10^9	5.6

1.2. (7 pts) We further run A on machine M1 and B on another machine M2. Assume they keep the same CPIs as **Q1.1**. Also assume that we measure the CPU time: A on M1 uses T1 seconds while B on M2 uses T2 seconds. If T1==T2, which machine has a shorter clock cycle time? Use calculation details to justify your answer.

2. (15 pts) Consider a machine running a program with four classes of instructions: A/B/C/D. The program requires 500 seconds, with the number of seconds executing each class as shown in the table below:

Instruction Class	**A**	**B**	**C**	**D**
Time (sec)	120	80	200	100

2.1. (5pts) By how much is the total execution time changed if the time for class D is reduced by 30% and the time for class C is increased by 20% (assuming no other changes)? Answer in seconds.

2.2. (5pts) If we want to reduce the total execution time by 20% by optimizing class A only, what is the target execution time of class A instructions in order to achieve the reduction (assuming all other instructions are not changed)?

2.3. (5pts) Can we reduce the execution time by 50% if we can only optimize **one** class of instructions (assuming no other changes)? If so, which class(es) can we consider as the optimization candidates? If not, why? Use calculations to justify your answer.

3. (20 pts) Assume for arithmetic, load/store, and branch instructions, a 4GHz processor has CPIs of 1, 10 and 5, respectively. Also assume that we are checking a program that requires the following number of instructions of different categories if run on a single processor:

- Arithmetic instructions: 2.8×10^9
- Load/store instructions: 1.4×10^9
- Branch instructions: 0.8×10^9

3.1 (7 pts) What is execution time for this program on a single processor as described above? Answer in seconds.

3.2 (5 pts) What is the average CPI of the giving program?

3.3 (8 pts) Assume that we try to parallelize this program to run over multiple processors of the same CPI (1, 10, and 5) and clock rate (4GHz). Suppose the parallelized version requires the same number of branch instructions as the original version *per processor*, while the number of arithmetic and load/store instructions *per processor* is divided by `p` (where `p` is the number of processors). Find the speedup for this program on 4 processors over the original execution from **Q3.1**.

Part 2. MIPS Programming (50%)

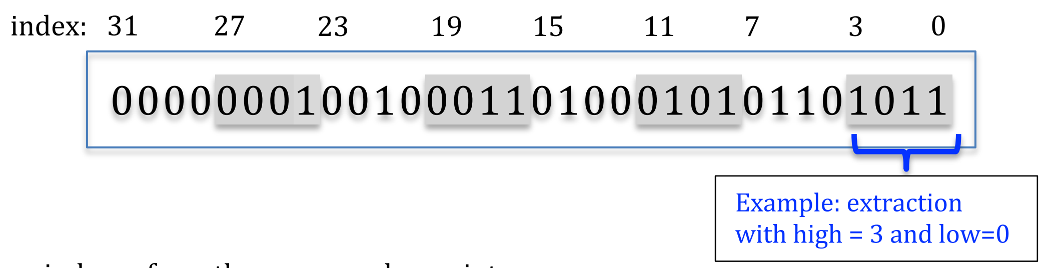
MIPS atoi and bit extracting. There is no built-in facility in MIPS that can convert a string into an integer as `atoi()` function in C. For this task, you will need to start from the provided template file and implement a program to extract the integer value from a string typed in by the user. You program must perform the following steps:

1. Accept a hexadecimal number from the standard input **as one string**. You must use system call with service number `8` to accept the input. This is already implemented in the provided file: do NOT change it.

2. Convert the string into an integer and print out (as both hexadecimal and unsigned decimal values). You can use service number **`34`** and **`36`** for printing.
3. If there is any invalid character in the input, report the error then immediately terminate the execution. (No more step 4 and 5)
4. Accept two integers as **`high`** and **`low`** indexes respectively.
5. For valid indexes, extract the bits located between **`high`** and **`low`** indexes (inclusively) and print out the extracted bits (again, as both hexadecimal and unsigned decimal values). For invalid indexes, report the index error and terminate.

Assumptions:

- All input hexadecimal strings have a fixed length of 8 characters. It will be left-padded with zeros if needed.
- Only digits '0' through '9' and capital letters 'A' through 'F' are considered as valid symbols. All other characters from the input string should trigger an error report.
- We use the indexing system as shown below:


index: 31 27 23 19 15 11 7 3 0
00000001001000110100010101101011
Example: extraction with high = 3 and low=0

- The two indexes from the user are always integers.
  - The valid range of inputs is: **`0<=low<=high<=31`**.
  - If user inputs are not in the valid range, an error report should be triggered.

Multiple sample runs (user input in blue):

`Note:` `Sample runs are for command-line execution in a prompt. Newline and user input display will be different if you run directly in MARS but all numbers/strings should match.`

```
Enter a hexadecimal number: 0123456B↵
Input: 0x0123456b = 19088747
Specify the high bit index to extract (0-LSB, 31-MSB): 3↵
Specify the low bit index to extract (0-LSB, 31-MSB, low<=high): 0↵
Extracted bits: 0x0000000b = 11

Enter a hexadecimal number: 0123456B↵
Input: 0x0123456b = 19088747
Specify the high bit index to extract (0-LSB, 31-MSB): 23↵
Specify the low bit index to extract (0-LSB, 31-MSB, low<=high): 15↵
Extracted bits: 0x00000046 = 70

Enter a hexadecimal number: 0123456B↵
Input: 0x0123456b = 19088747
Specify the high bit index to extract (0-LSB, 31-MSB): 7↵
Specify the low bit index to extract (0-LSB, 31-MSB, low<=high): 7↵
Extracted bits: 0x00000000 = 0
```

`Note: high==low, extract one bit.`

```
Enter a hexadecimal number: ABCDEFGH↵
Error: Input has invalid digits!
```

```
Note: Invalid digits.
```

```
Enter a hexadecimal number: 0123456B↵
Input: 0x0123456b = 19088747
Specify the high bit index to extract (0-LSB, 31-MSB): 32↵
Error: Input has incorrect index(es)!
```

```
Note: Invalid index (high).
```

```
Enter a hexadecimal number: 0123456B↵
Input: 0x0123456b = 19088747
Specify the high bit index to extract (0-LSB, 31-MSB): 15↵
Specify the low bit index to extract (0-LSB, 31-MSB, low<=high): 16↵
Error: Input has incorrect index(es)!
```

```
Note: Invalid index (low
cannot go beyond high).
```

Notes:

- Your submission must be able to assemble and run with MARS. We cannot grade it otherwise.
- As in all programming assignments, your code must be very well commented.
- **A template file has been provided to you as your starting point.** Do not modify the part marked in the template file.
- Your code will need to deal with standard input/output with system calls. The template file includes some examples and defines useful prompt strings that you should use.
- You can find more helpful system calls here: http://courses.missouristate.edu/KenVollmar/mars/Help/SyscallHelp.html. **Hint:** there are system calls that can help you to print an integer as binary, decimal, and hex.
- You might find ASCII encoding useful: http://en.wikipedia.org/wiki/ASCII#Printable_characters.

Grading rubric:

Code submission and documentation	5/50
○ Make sure your code is well commented as always	
○ Make sure to follow the required format, including file names	
Code assembled with no error	5/50
Required standard input/output	5/50
String to integer conversion	15/50
Bits extraction	15/50
Error checking and report	5/50

- We will be comparing the generated output and the expected output for grading. The template file defines useful prompt strings that you should use.
- **No late work allowed after 24 hours.**
  - Late submission automatically uses one of your tokens.
- **Extra credit for early submissions:**
  - **1%** extra credit rewarded for every 24 hours your submission made before the due time.
  - Up to **3%** extra credit will be rewarded.
  - Your latest submission will be used for grading and extra credit checking. You CANNOT choose which one counts.
- You will need to make two separate submissions for Part1 and Part2 on gradescope. The **later** one will be used to determine whether your HW1 is early / normal/ late.