CSE 220: Systems Fundamentals I

Stony Brook University
Homework Assignment #5
Spring 2024
Due: **Friday**, May 3rd, 2024 by 9:00 pm EDT

Updates to this Document

- None yet!

Learning Outcomes

By the end of this assignment you should be able to do the following in MIPS Assembly:

- Perform memory, arithmetic, and bitwise operations.
- Iterate through strings and arrays using loops.
- Implement functions and follow proper register conventions.
- Implement basic data structures.

Preliminaries

- You must use the Stony Brook version of MARS provided in the template repository. Do not use the version of MARS posted on the official MARS website. The Stony Brook version has a reduced instruction set, added tools, and additional system calls you might need to complete the homework assignment.
- When writing assembly code, try to stay consistent with your formatting and to comment as much as possible. It is much easier for your TAs and the professor to help you if we can quickly figure out what your code does.
- You personally must implement the programming assignments in MIPS Assembly language by yourself. You may not write or use a code generator or other tools that write any MIPS code for you. You must manually write all MIPS assembly code you submit as part of the assignments.
- Do not copy or share code. Your submissions will be checked against other submissions from this semester and from previous semesters.

How to Run MARS

MARS has a GUI feature, so if you want to use the GUI, you must run it natively on Windows, MacOS or Linux. You can also run MARS in text-only mode, which is how we will use it to grade your work in CodeGrade and how you must run it in Codespaces. The assignment repo

includes a copy of MARS so that the unit tests will execute properly. The repo also contains C files that begin `tests_`, which will run test cases against your code (but in text-only mode).

In Windows and MacOS, you should be able to double-click the JAR file to run the program. In the Linux VM (or any other OS), type `java -jar MarsFall2020.jar` to run it in GUI mode.

How Your MIPS Code Will Be Tested

As with the C assignments, testing scripts will execute your code with input values (e.g., command-line arguments, function arguments) and will check for expected results (e.g., print-outs, return values, etc.) For this assignment, your program will generate output that will be checked for exact matches by the grading scripts. Therefore, it is imperative that your code does not generate extraneous output.

Each test case must execute in 500,000 instructions or fewer. Efficiency is an important aspect of programming. To find the instruction count of your code in MARS, go to the **Tools** menu and select **Instruction Statistics**. Press the button marked **Connect to MIPS**. Then assemble and run your code as normal.

All work you do must be in the file `hw5.asm`. To test your work in GUI mode, open one of the files given in the `tests` directory, like `init_print_01`.asm, and simply run it. To run the same test case in command-line mode, such as under Codespaces, type `java -jar MarsFall2020.jar --noGui ./tests/init_print_01.asm`. To see what the expected output is, consult the file `./tests/expected_outputs/init_print_01.txt`.

To run all tests, build the project as you normally would using CMake, and then execute `./build/run_all_tests` This is the testing method that will be used on CodeGrade.

Finally, TA Brian created a program in `run_and_compare.c` which you can use to compare your code's output with the expected output. You'll need to build the file yourself. Run the resulting executable for a short help message on how to use it.

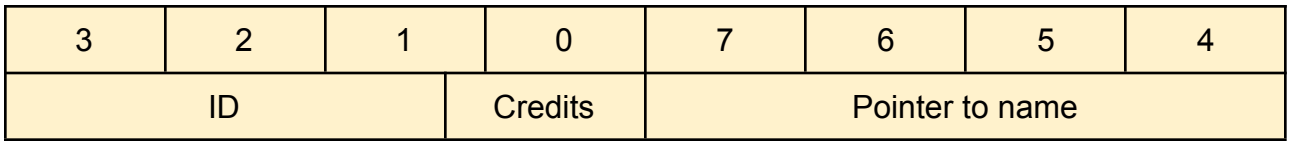
Assignment Overview

In the first part of the assignment, you will be working with a student record struct that contains the following fields:

1. Student ID: 22 bits
2. Number of credits: 10 bits
3. Pointer to the student's name: 32 bits

Numerical fields are stored as unsigned integers.

Byte #



You will implement functions that involve student records, such as initializing a student record struct and printing its fields. Then, you will implement a hash table with linear probing that stores pointers to student records.

Part 1: Initialize a Student Record Given the Fields

```
void init_student(int id, int credits, char *name,
                  struct student *record)
```

Given the fields of a student, the `init_student` function stores the fields in `record` in the order shown in the previous diagram. `record` is an uninitialized region of memory 8 bytes long.

Part 2: Print the Fields of a Student Record

```
void print_student(struct student *record)
```

Given a pointer to a student struct, the `print_student` function prints the fields of the student in the format `<ID> <Credits> <Name>`, separated by a space. When printing the name, print the actual string, not its address. Do not print a newline at the end.

Example

Consider a student record with the following fields:

```
id: 3126375
credits: 314
name: 0x10010008
*name: "Kevin T. McDonnell"
```

This data would be represented as a student record struct as the following bytes:
`10111110 11010001 10011101 00111010 00010000 00000001 00000000 00001000`

And, as printed:
`3126375 314 Kevin T. McDonnell`

Part 3: Initialize an Array of Student Records

```
void init_student_array(int num_students, int id_list[],
                        int credits_list[], char *names,
                        struct student records[])
```

Given arrays of student fields, the `init_student_array` function initializes an array of student structs.

Arguments:

- `num_students`: the number of students to initialize. It is guaranteed that the arrays passed into this function have at least `num_students` elements.
- `id_list` and `credits_list` are the arrays of IDs and credits of each student, respectively.
- `names` is a pointer to the first character of the first student name, where the names are separated by a single null terminator. For example:
  - `Wolfie Seawolf\0Kelly Chen\0Michael Borczuk\0`
- `records` is a pointer to an uninitialized region of memory, `8 * num_students` bytes long. Note that this argument is stored on the top of the stack.

Your implementation should call `init_student`.

Hash Table with Linear Probing

Now, you will implement a hash table to quickly access and modify student records. The hash table is implemented as an array of pointers to student records, using `record->id % table_size` as the array index.

An empty item is denoted by a NULL pointer (`0x0`), and when a student record is deleted, the item in the hash table is replaced by -1 (or equivalently, `0xFFFFFFFF`), which we will refer to as the "tombstone" value.

The following are some examples to demonstrate the linear probing algorithm. For simplicity, we will denote a student record by only its ID. An empty item is denoted as 'X' and a tombstone is denoted as 'T'. During the loop, the starting item is marked as red, and the items that are accessed are marked bold.

Example #1: Collision while Inserting
Before inserting 23:
`X X X` **`13`** **`X`**
After inserting 23:
`X X X 13 23`

Example #2: Collision while Inserting, with Looping Around
Before inserting 18:
X X X **13** **23**
After inserting 18:
18 X X 13 23

Example #3: Inserting into Tombstone
Before inserting 7:
X X **2** **T** 4
After inserting 7:
X X 2 7 4

Example #4: Inserting into Full Hash Table
Before inserting 7:
0 **1** **2** **3** **4**
After inserting 7:
(Unchanged)

Example #5: Collision while Searching (Found)
Searching for 2:
X X **7** **2** 4

Example #6: Collision while Searching (Not Found)
Searching for 2:
X 1 **7** **X** 4

Example #7: Searching over Tombstone
Searching for 2:
X X **7** **T** **2**

The logic for deletion is the same as searching, but if the element is found, it is replaced with the Tombstone value.

Part 4: Insert a Student Record into a Hash Table

```
int insert(struct student *record, struct student *table[],
           int table_size)
```

Given a pointer to a student record, the `insert` function inserts the pointer to the hash table. If the calculated array index is already occupied, use linear probing to find an empty index.

Returns:

- `-1` If the hash table is full and the record could not be inserted

- Otherwise, the array index that the record was inserted in

Tip: use your `print_student` function to print the contents of the hash table and debug your program.

Part 5: Search for a Student Record in the Hash Table

```
(struct student*, int) search(int id, struct student *table[],
                              int table_size)
```

Given a student ID, the `search` function searches for the matching student record using the linear probing algorithm and returns the pointer. When searching for the matching student record, make sure to skip over the tombstone value.

Returns in `$v0`:

- The pointer to the student record if it was found.
- `NULL (0)` if a matching record was not found.

Returns in `$v1`:

- The array index in the hash table where the student record was found.
- -1 if a matching record was not found.

Part 6: Delete a Student Record in the Hash Table

```
int delete(int id, struct student *table[], int table_size)
```

Given a student ID, the `delete` function searches for the matching student record and replaces it with the tombstone value. When searching for the matching student record, make sure to skip over the tombstone value.

Returns in `$v0`:

- The array index in the hash table where the student record was found.
- -1 if a matching record was not found.

Your implementation should call `search`.

Testing Your Code

The tests directory contains numerous scripts

Grading Notes

The correctness of your code will be tested using extensive print statements (in MIPS). Your code must not produce any extraneous output; otherwise, most or all of the test cases will fail.

The number of instructions per test is capped at 500,000. If your program gets caught in an infinite loop or is extremely inefficient, MARS will terminate the test case once this instruction limit is reached.

Function arguments will be passed exclusively through the $a registers, with the sole exception of `init_student_array`, which takes its final argument from the top of the stack. Do not attempt to grab the function arguments from the .data section of the testing scripts.

Note: any evidence of hard-coding test case input or output will automatically result in a score of zero for the assignment. This kind of behavior is borderline academic dishonesty.

Generative AI

Generative AI may not be used to complete this assignment.

Academic Honesty Policy

Academic honesty is taken very seriously in this course. By submitting your work for grading you indicate your understanding of, and agreement with, the following Academic Honesty Statement:

1. I understand that representing another person's work as my own is academically dishonest.
2. I understand that copying, even with modifications, a solution from another source (such as the web or another person) as a part of my answer constitutes plagiarism.
3. I understand that sharing parts of my homework solutions (text write-up, schematics, code, electronic or hard-copy) is academic dishonesty and helps others plagiarize my work.
4. I understand that protecting my work from possible plagiarism is my responsibility. I understand the importance of saving my work such that it is visible only to me.
5. I understand that passing information that is relevant to a homework/exam to others in the course (either in lecture or even in the future!) for their private use constitutes academic dishonesty. I will only discuss material that I am willing to openly post on the discussion board.
6. I understand that academic dishonesty is treated very seriously in this course. I understand that the instructor will report any incident of academic dishonesty to the University's Academic Judiciary.

7. I understand that the penalty for academic dishonesty might not be immediately administered. For instance, cheating on a homework assignment may be discovered and penalized after the grades for that homework have been recorded.
8. I understand that buying or paying another entity for any code, partial or in its entirety, and submitting it as my own work is considered academic dishonesty.
9. I understand that there are no extenuating circumstances for academic dishonesty.