CS 241 — Winter 2022 — Assignment 1

Assignments for CS 241

Assignment 1
Wednesday, Jan 19th at 5:00pm

Assignment 2 →
Friday, Jan 28th at 5:00pm

P1 • P2 • P3 • P4 • P5 • P6 • P7

In this assignment, you will begin writing very simple programs in MIPS machine language on the CSCF Linux environment.

You must prepare and test your solutions using tools available on the CSCF Student Linux Computing Environment. You must submit your solutions to the Marmoset automatic grading system available at `https://marmoset.student.cs.uwaterloo.ca`, which will run them on a number of test inputs and grade them automatically. You may submit your solutions as many times as you wish prior to the submission deadline (though the number of times you can view release test results is limited; see below).

Important Note: For this and future assignments, be sure to enter the command `source /u/cs241/setup` to gain access to the CS 241 tools.

Marmoset Information

Marmoset tests your submission with three kinds of tests: public tests, release tests, and secret tests. If you have used Marmoset in past courses you are probably familiar with the first two, and you may also have seen the third.

Test your submissions yourself before submitting them to Marmoset. You should not rely on Marmoset as a means of submission testing.

Public tests have their results made visible as soon as the testing process is finished. They are generally very simple test cases, or test cases that are given as examples on the assignment page. Once you have passed the public tests, Marmoset will give you the option to view release test results. Some problems might only have public tests (such as Problem 1 on this assignment), in which case the option to view release test results will not appear.

Release tests have their results hidden until you pass the public tests and spend a **release token** to view the results. To encourage you to start the assignment early, you are given only three release tokens per problem, and each release token takes 12 hours to regenerate after use. If you run out of release tokens, you will not be able to check your program against the release tests, and must rely purely on your own testing to give yourself confidence in your program.

In general, release tests check for common errors, but do not constitute a comprehensive test suite. Release tests are meant to check that your program is not fundamentally broken in some way, rather than to thoroughly check its correctness.

Spending a release token will not show you every release test result. If you passed all the release tests, you will simply see a screen showing the combined mark you received for public and release tests. If you failed any release tests, you will see a screen showing the results of at most two of the failed release tests.

Secret tests have not only their results but their *existence* hidden until they are manually made visible by course staff after the deadline. Before secret tests are made visible, you will not see any indication that they exist on Marmoset; the mark values shown on Marmoset only include public and release tests. The assignment page will indicate which problems have secret tests and how many marks the secret tests are worth, but will not give any details beyond that.

Secret tests are intended to fill in the gaps left by public and release tests. Edge cases and stress tests (i.e., tests of program efficiency) will usually be secret tests rather than release tests. To ensure you pass the secret tests, you should write your own thorough test suites for your program. As mentioned, release tests are not meant to be comprehensive, so passing all the release tests does not mean you are "done" with a problem unless it has no secret tests.

For this assignment, secret tests are worth **10%** of the Marmoset test marks. Note that the Marmoset test marks do not necessarily make up the entire assignment mark, since a small number of questions throughout the term will be hand-marked to give feedback on coding style. We do not announce which questions will be hand-marked ahead of time to encourage you to always write quality code.

You can use the `marmoset_submit` command to submit to Marmoset directly from the command line, avoiding the need to copy files back and forth between your computer and the Linux environment.

Part I. Creating binary files with `cs241.wordasm`

We have provided the tool `cs241.wordasm` that may be used to create a file whose binary content is specified using hexadecimal notation. `cs241.wordasm` reads the hexadecimal representation of several 32-bit words on standard input. On standard output, it outputs a file containing, for each of the input words, the four bytes containing the 32 bits represented by the word (8 bits per byte). On each line of input, any characters after a semicolon (;) are assumed to be comments and ignored. Example:

```
% cat > input
.word 0x43533234  ; C(43) S(53) 2(32) 4(34)
.word 0x3120726f  ; 1(31) space(20) r(72) o(6f)
.word 0x636b730a  ; c(63) k(6b) s(73) newline(0a)
% cs241.wordasm < input
CS241 rocks
% (end of example)
```

To examine the individual bytes in a binary file, you can use the tool `cs241.binview`. By default, it prints the binary representation of each byte, with 4 bytes (one 32-bit word) per line.

```
% cs241.wordasm < input > output
% cs241.binview output
01000011 01010011 00110010 00110100
00110001 00100000 01110010 01101111
01100011 01101011 01110011 00001010
% (end of example)
```

By using the command line option `-ha` (show hex and ASCII) we can see that `cs241.wordasm` encoded the file as expected. A full list of command line flags for `cs241.binview` can be obtained by running `cs241.binview --help`.

```
% cs241.binview -ha output
0x43 0x53 0x32 0x34
C    S    2    4

0x31 0x20 0x72 0x6F
1    (SP) r    o

0x63 0x6B 0x73 0x0A
c    k    s    ^LF
% (end of example)
```

Problem 1 — 10 marks of 70 (filename: `helloworld.hex`)

Write a hexadecimal representation (using the .word notation) of a file with contents:

```
Hello
from Linux!!!
```

To be precise, there should be two newline characters (ASCII 10) in the file, one immediately after the "o" on the first line, and one immediately after the third "!" on the second line. There is a single space character between "from" and "Linux". There should be no other whitespace in the file.

You may find it useful to use the Linux command `man ascii` to get an ASCII code chart. Before submitting to Marmoset, run the `cs241.wordasm` tool with your solution as standard input to check that your output is correct.

You must name the file you submit to Marmoset `helloworld.hex` for the testing scripts to work correctly.

Part II. MIPS Machine Language Programming

The `mips.twoints` tool loads a MIPS machine language program from a file into memory starting at location 0. It then reads two integers, stores them into registers 1 and 2, runs your program, and, when your program returns to the instruction whose address is stored in register 31, prints the values of all the registers and exits. Run this tool using the command `mips.twoints` ***`mycode.mips`***, replacing ***`mycode.mips`*** with the name of a file containing your machine code.

Since MIPS machine language is encoded in binary, not text, you cannot create it directly using an editor like vim. You must specify your machine language program in hexadecimal and use `cs241.wordasm` to translate it to (binary) machine language.

For each of the following problems, create the hexadecimal representation of a MIPS machine language program that solves the problem. You will need to use the MIPS Machine Language Reference Sheet to find the binary encodings of the MIPS instructions you want to use, then convert these binary encodings to hexadecimal (since `cs241.wordasm` does not support binary input).

Test your program using `cs241.wordasm` and `mips.twoints` as follows:

```
% vim mycode.hex
% cs241.wordasm < mycode.hex > mycode.mips
% mips.twoints mycode.mips
Enter value for register 1: 1
Enter value for register 2: 2
Running MIPS program.
...
```

Each problem specifies a filename for the file that you will submit. You must use the specified filename for the testing scripts to work correctly.

Scratch registers used to hold temporary results may be used and do not need to be set back to 0.

Problem 2 — 8 marks of 70 (filename: `a1p2.hex`)

Write the hexadecimal notation for a MIPS machine language program that returns to the address saved in register 31. This is the only thing your program should do.

Example of running the program:

```
Enter value for register 1: 1
Enter value for register 2: 2
Running MIPS program.
MIPS program completed normally.
...
```

Problem 3 — 8 marks of 70 (filename: `a1p3.hex`)

Write the hexadecimal notation for a MIPS machine language program that copies the value in register 1 to register 3, then adds the values in register 1 and 3 placing the result in register 4 and then returns.

Example of running the program:

```
Enter value for register 1: 2
Enter value for register 2: 3
Running MIPS program.
MIPS program completed normally.
$01 = 0x00000002   $02 = 0x00000003   $03 = 0x00000002   $04 = 0x00000004
...
```

Problem 4 — 12 marks of 70, 2 secret marks (filename: `a1p4.hex`)

Write the hexadecimal notation for a MIPS machine language program that determines the minimum of the two's complement signed integers in registers 1 and 2, places it in register 3, and returns.

Example of running the program:

```
Enter value for register 1: 3
Enter value for register 2: 5
Running MIPS program.
MIPS program completed normally.
$01 = 0x00000003   $02 = 0x00000005   $03 = 0x00000003   ...
```

Problem 5 — 8 marks of 70 (filename: `a1p5.hex`)

Write the hexadecimal notation for a MIPS machine language program that adds 42 to the value in register 2 placing the result in register 3 and then returns.

Example of running the program:

```
Enter value for register 1: 2
Enter value for register 2: 3
Running MIPS program.
MIPS program completed normally.
$01 = 0x00000002   $02 = 0x00000003   $03 = 0x0000002D    ...
```

Problem 6 — 12 marks of 70, 1 secret mark (filename: `a1p6.hex`)

Write the hexadecimal notation for a MIPS machine language program that interprets the value in register 1 as the address of a word `w` in memory, and places in register 3 the address of the next word after `w` in memory, and then returns.

You may assume the value in register 1 is a multiple of 4 when interpreted as an unsigned integer (that is, the address is word-aligned) and is strictly less than $2^{32} - 4$.

Problem 7 — 12 marks of 70, 4 secret marks (filename: `a1p7.hex`)

An integer m **divides** an integer n if there exists an integer k such that $mk = n$. Write the hexadecimal notation for a MIPS machine language program that determines whether the value in register 1 divides the value in register 2, treating both values as unsigned integers. The program should place the value 1 in register 3 if the value in register 1 divides the value in register 2, and otherwise it should place the value 0 in register 3. Afterwards it should return.

Hint: The above definition of "divides" applies to *all* pairs of integers. If you are unsure how certain pairs of integers should be handled, read the definition more carefully. Be sure to test your program thoroughly and think about edge cases.

Click here to go back to the top of the page.