COS 217: Introduction to Programming Systems

Assignment 5

PRINCETON UNIVERSITY

1

The wc command

If the file named `proverb` contains these characters:

```
Learning␣is␣a␣n
treasure␣which␣n
accompanies␣its␣n
owner␣everywhere.␣n
--␣Chinese␣proverb␣n
```

then the command:

```
$ wc < proverb
```

writes this line to `stdout`:

```
       5      12      82
```

2

2

Part 1a Task

The given `mywc.c` file contains a C program that implements the subset of the `wc` command described above.

Translate that program into ARMv8 assembly language, thus creating a file named `mywc.s`. Your `mywc.s` program must be an accurate translation of `mywc.c`.

3

3

Part 1b Task

Compose data files (patterned `mywc*.txt`) that, when read by your `mywc.s` program, perform:

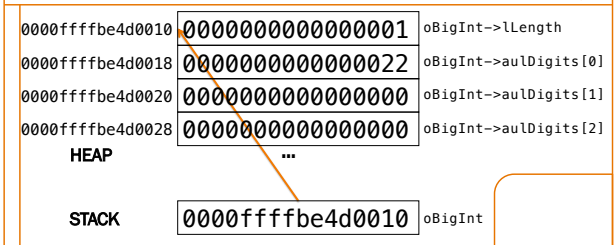
- boundary tests
- statement tests
- stress tests

Describe your test files' testing characteristics and the corresponding lines in `mywc.c` that they exercise.

4

4

Part 2: `BigInt` objects



5

5

Part 2a: Unoptimized C `BigInt_add` Implementation

Study the given code.

Then build a `fib` program consisting of the files `fib.c`, `bigint.c`, and `bigintadd.c`, without the `-D NDEBUG` or `-O` options.

Run the program to compute `fib(250000)`. In your readme file note the amount of CPU time consumed.

6

6

Part 2b/c: Optimized C `BigInt_add` Implementation

Rebuild a `fib` program consisting of the files `fib.c`, `bigint.c`, and `bigintadd.c`, **with** the `-D NDEBUG` and `-O` options.

Run the program to compute `fib(250000)`. In your readme file note the amount of CPU time consumed.

Profile the code with `gprof`. (More on this next lecture.)

7

7

Part 2d: Translate to Assembly Language

Suppose, not surprisingly, your `gprof` analysis shows that most CPU time is spent executing the `BigInt_add` function. In an attempt to gain speed, you decide to code the `BigInt_add` function manually in assembly language…

Manually translate the C code in the `bigintadd.c` file into ARMv8 assembly language, thus creating the file `bigintadd.s`. Do not translate the code in other files into assembly language.

Your assembly language code must **store all parameters and local variables defined in the `BigInt_larger` and `BigInt_add` functions in memory, on the stack**.

8

8

Part 2e: Optimize to use registers, not the stack

Suppose, to your horror, you discover that you have taken a step backward: the CPU time consumed by your assembly language code is approximately the same as that of the non-optimized compiler-generated code! So you decide to optimize your assembly language code…

Manually optimize your assembly language code in `bigintadd.s`, thus creating the file `bigintaddopt.s`. Specifically, perform this optimization:

- Store all parameters and local variables defined in the `BigInt_larger` and `BigInt_add` functions in callee-saved registers instead of in memory.

9

9

Part 2f (Challenge Portion): Optimize All You Want

Finally, suppose you decide to optimize your assembly language code even further, moving away from a statement-by-statement translation of C code into assembly language…

Further optimize your assembly language code in `bigintaddopt.s`, thus creating the file `bigintaddoptopt.s`. Specifically, perform these optimizations:

- Use the assembly language *guarded loop* pattern described in Section 3.2 of Chapter 5 of the Pyeatt with Ughetta book instead of the simpler but less efficient loop patterns described in precepts.
- "Inline" the call of the `BigInt_larger` function. That is, eliminate the `BigInt_larger` function, placing its code within the `BigInt_add` function.
- Use the `adcs` ("add with carry and set condition flags") instruction effectively. The `adcs` instruction computes the sum of its source operand, its destination operand, and the C condition flag, places the sum in the destination operand, and assigns 1 (or 0) to the C condition flag if a carry occurred (or did not occur) during the addition. Effective use of the `adcs` instruction will use the C condition flag instead of a `ulCarry` variable to keep track of carries during addition.

"Feel free to implement any additional optimizations"

"This part is challenging. We will not think unkindly of you if you decide not to do it."

10

10