CISS360: Computer Systems and Assembly Language
Assignment a10

Name: jdoe5@cougars.ccis.edu Score:

Open `main.tex` and enter answers (look for `answercode`, `answerbox`, `answerlong`). Turn the page for detailed instructions. To rebuild and view pdf, in bash shell execute `make`. To build a gzip-tar file, in bash shell execute `make s` and you'll get `submit.tar.gz`.

Objectives

- Perform computations involving number representations for signed int for fix width using two's complement.
- Translate between assembly code and machine code

Some general instructions:

- Once you have unpacked the download, run `make` to check if your LaTeX is missing any files. If your `make` is successful (i.e. no error messages), you should see `main.pdf`. If there's an error, let me know ASAP.
- The main file is `main.tex`. The questions are in the files `q01.tex`, `q02.tex`, etc. You type up your answers in `q01s.tex`, `q02s.tex`, etc. These files are included in `main.tex` by the commands such as `\input{q01.tex}` (sort of like `#include`). So for sure you want to first look at `main.tex`. There are also files names `how-to-*.tex` that gives you examples on how to typeset certain computations in LaTeX. Look at the examples in these files and make full use of the LaTeX code for copy-paste-modify. As in all things, you want to work in "baby steps" – you want to do `make` frequently and check that the pdf generated is OK.
- I encourage discussion and whiteboarding and checking each others work in hangout. Or you can discuss on our discord server in the ciss360 channel. However when you write up the answer, it must be your own work. Sharing of LaTeX work is plagiarism will result in an immediate -1000% .
- Show all your work. All computations should be done by hand, i.e., do not use your TI calculators or any calculators. And then you typeset in LaTeX.
- As for LaTeX specific questions, again goto hangout or chat in discord. You can go to my website http://yliow.github.io and look for `latex.pdf` in the Tutorials section – but it's a lot easier to chat in hangout or discord.
- As usual submit using alex.

LaTeX

In general for regular English text, you just type regular English text. When you want to type math, you can type `$x = 1$` (i.e., enclose your math with `$` symbols). And this is what you see: $x = 1$. Notice that mathematical symbols are then in italics. Or, to emphasize your math you can also do `\[ x = 1 \]` (i.e., enclose in `\[ ... \]`). This is what it looks like:

$$x = 1$$

Notice that this math sentence is then centered in a separate paragraph.

Most of the LaTeX typesetting commands for this assignment is very basic. Most of the LaTeX commands in the files here should give you an idea of what to do. So just look at the LaTeX files here and copy-and-paste whatever you need. If you want more sophisticated LaTeX commands, you can look at my latex tutorial on my website for more information.

LaTeX is probably the most sophisticated scientific typesetting system in the world and is used by people in CS, math, physics, etc. It's now even used in general publishing. LaTeX is built on top of the original TeX language which was written by the world's most famous living computer scientist Donald Knuth. He is also writing the world's most famous multi-volume treatise on algorithms called The Art of Computer Programming.

How to write a conversion from base 10 another base

Problem: Convert 19 to base 2.
Answer:

2	19		
2	9	r	1
2	4	r	1
2	2	r	0
2	1	r	0
	0	r	1

Therefore $19 = 10011_2$.

Make sure you look at the LaTeX code:

```
\begin{longtable}{r|rrr}
  2 & 19 &   &   \\ \cline{2-4}
  2 &  9 & r & 1 \\ \cline{2-4}
  2 &  4 & r & 1 \\ \cline{2-4}
  2 &  2 & r & 0 \\ \cline{2-4}
  2 &  1 & r & 0 \\ \cline{2-4}
    &  0 & r & 1 \\ \cline{2-4}
\end{longtable}
Therefore $19 = 10011_{2}$.
```

The `&` is a separator of values in the table. The `\cline{2-4}` draws a line from column 2 to column 4.

When you need to do something similar to the computations below, just look for the LaTeX code in this file, copy-paste-and-modify.

How to write multi-column addition

Problem: Compute $423_8 + 636_8$.
Answer:

		1		1
			4	2	3
+			6	3	6
		1	2	6	1

Therefore $423_8 + 839_8 = 1261_8$.

Make sure you look at the LaTeX code:

```
\begin{longtable}{ccccc}
  & 1 &   & 1 &   \\
  &   & 4 & 2 & 3 \\
+ &   & 6 & 3 & 6 \\ \hline
  & 1 & 2 & 6 & 1 \\ \hline
\end{longtable}
Therefore $423_{8} + 839_{8} = 1261_{8}$.
```

The `{ccccc}` in the LaTeX says there are 5 columns, `&` in the LaTeX are column separators for each row, the `\\` is a newline, the `\hline` draws a line through all columns.

(Note: Show the carries.)

For subtraction you don't have to show the borrows, because it would be too messy:

	1	2	3
−	1	0	8
		1	5

How to write multi-column multiplication

Problem: Compute $123_8 \times 456_8$.
Answer:

$$
\begin{array}{cccccc}
 & & & 1 & 2 & 3 \\
\times & & & 4 & 5 & 6 \\
\hline
 & & & 7 & 6 & 2 \\
 & & 6 & 3 & 7 & \\
+ & 5 & 1 & 4 & & \\
\hline
 & 6 & 0 & 7 & 5 & 2
\end{array}
$$

Therefore $123_8 \times 456_8 = 60752_8$

Take a look at the LaTeX code:

```
\begin{longtable}{cccccc}
         &   &   & 1 & 2 & 3 \\
$\times$ &   &   & 4 & 5 & 6 \\ \hline
         &   &   & 7 & 6 & 2 \\
         &   & 6 & 3 & 7 &   \\
  +      & 5 & 1 & 4 &   &   \\ \hline
         & 6 & 0 & 7 & 5 & 2 \\ \hline
\end{longtable}
Therefore $123_{8} \times 456_{8} = 60752_{8}$
```

The `{cccccc}` means there are 6 columns (with values centered). The `&` is a separator of column values. The `\\` is newline. The `\hline` is to draw a horizontal line.

Note that for this problem, there are actually two multicolumn additions. You don't have to show the work for these multicolumn additions.

How to write fast conversions

Problem: Convert 1011110_2 to base 4.
Answer:

$$\begin{aligned} 1011110_2 &= (01_2|01_2|11_2|10_2)_4 \\ &= (1_4|1_4|3_4|2_4)_4 \\ &= 1132_4 \end{aligned}$$

Therefore $1011110_2 = 1132_4$.

In general when doing a sequence of aligned computations, you do this

$$\begin{aligned} a + b + c &= d \times e \cdot f \\ &= g_h + i \\ &= j + k^l \end{aligned}$$

Notice that the = symbols are aligned. Take a look at the LaTeX code:

```
\begin{align*}
a + b + c &= d \times e \cdot f \\
          &= g_{h} + i \\
          &= j + k^{l}
\end{align*}
```

The `&` is the alignment character and the `\\` is newline. The aligned computations must be enclosed in `\begin{align*}` and `\end{align*}`.

How to write a conversion from base 10 float to another base

Problem: Convert 19.1 to base 2 up to 8 places after the binary point.
Answer:

$19.1 = 19 + 0.1$.

2	19		
2	9	r	1
2	4	r	1
2	2	r	0
2	1	r	0
	0	r	1

Therefore $19 = 10011_2$.

$$2 \times 0.1 = 0.20$$
$$2 \times 0.2 = 0.40$$
$$2 \times 0.4 = 0.80$$
$$2 \times 0.8 = 1.61$$
$$2 \times 0.6 = 1.21$$
$$2 \times 0.2 = 0.40$$
$$2 \times 0.4 = 0.80$$
$$2 \times 0.8 = 1.61$$
$$2 \times 0.6 = 1.21$$

Therefore $0.1 = 0.000110011_2$ (to 9 places). With rounding, $0.1 = 0.00011010_2$ (to 8 places). Therefore $19.1 = 10011.00011010_2$.
(Check: $10011_2 = 1 + 2 + 16 = 19$ and $0.00011010_2 = 1/16 + 1/32 + 1/128 = 0.1015625$.)

How to write multi column float operation

Problem: Compute $1.1_2 + 0.111_2$.
Answer:

$$\begin{array}{cccccccc} & & 1 & 1 & & 1 & & \\ & & & 1 & . & 1 & & \\ + & & & 0 & . & 1 & 1 & 1 \\ \hline & & 1 & 0 & . & 1 & 1 & 1 \end{array}$$

Therefore $1.1_2 + 0.111_2 = 10.\overline{111}_2$.

Problem: Compute $1.1_2 \times 0.11_2$.
Answer:

Ignoring the binary point,

$$\begin{array}{cccccc} & & & & 1 & 1 \\ \times & & & & 1 & 1 \\ \hline & & & & 1 & 1 \\ + & & & 1 & 1 & \\ \hline & & 1 & 0 & 0 & 1 \end{array}$$

Therefore $1.1_2 \times 0.11_2 = 1.001_2$
(Check: $1.1_2 \times 0.11_2 = 1.5 \times (0.5 + 0.25) = 1.5 \times 0.75 = 1.125$ and $1.001_2 = 1 + 1/8 = 1.125$.)

Problem: Compute $10.1011_2/0.11_2$ to eight places after binary point.

(You don't have to show your work for floating point division because the long division is tedious to type. But you do need to know how to do long division. If you want to do it, see below. You have to comment out `\begin{answerlong}` and `\end{answerlong}`.)
Answer: $10.1011_2/0.11_2 = 1010.11_2/11_2$

```
      011 10010101010101010
   +----------------------
11 |  1010.11000000000000000
    - 00
     ----------------------
       101
      - 11
      ----------------------
       100
      - 11
      ----------------------
        01 1
       - 1 1
       ---------------------
         0 01
        -  00
        ---------------------
           010
          - 00
          ------------------
            100
           - 11
           -----------------
             010
            - 00
            ----------------
              100
             - 11
             ---------------
               010
              - 00
              --------------
                100
               - 11
               -------------
                 010
                - 00
                ------------
                  100
                 - 11
                 -----------
                   010
                  - 00
                  ----------
                    100
                   - 11
                   ---------
                     010
                    - 00
                    --------
                      100
                     - 11
                     -------
                       010
                      - 00
                      ------
                        100
                       - 11
                       -----
                         010
                        - 00
                        ----
                          10
```

Therefore $10.1011_2/0.11_2 = 011.10010101010101010_2$. Up to 9 places, $10.1011_2/0.11_2 = 11.100101010_2$. After rounding at the 9th place, $10.1011_2/0.11_2 = 11.10010101_2$.

(Check: $10.1011_2/0.11_2 = (2 + 1/2 + 1/8 + 1/16)/(1/2 + 1/4) = 3.5833333333333335$

and $11.100101010_2 = 2 + 1 + 1/2 + 1/2^4 + 1/2^6 + 1/2^8 = 3.58203125$.)

Q1.

(a) Write down the 4-bit two's complement representation of 3 and 6
(b) What is 3 + 6 using 4-bit two's complement representation?
(c) What is 3 – 6 using 4-bit two's complement representation? (Compute -6 and then 3 + (-6))
(d) Extend your answer in (c) to 8-bit two's complement representation.
(e) What does 1010 (in two's complement representation) represent? (Give it in human readable form, i.e., signed magnitude base 10.)
(f) Still using 4-bit two's complement representation, what is the minimal x, written in base 10, such that $5+x$ will give you an overflow? What is $5+x$ in the usual base 10 notation using the correct mathematical addition? What is $5+x$ in the usual base 10 notation if you use signed 4-bit addition?

(a) Answer:

(b) Answer:

(c) Answer:

(d) Answer:

(e) Answer:

(f) Answer:

Q2.

(a) Write down the 8-bit two's complement representation of 4 and 7
(b) What is 4 + 7 using 8-bit two's complement representation
(c) What is 4 – 7 using 8-bit two's complement representation? (Compute -7 and then 4 + (-7))
(d) What does 11100001 (in two's complement representation) represent? (Give it in human readable form, i.e., signed magniture in base 10.)
(e) Still using 8-bit two's complement representation, what is the minimal x, written in base 10, such that $-5-x$ will give you an overflow? What is $-5-x$ written in the usual base 10 notation using the correct mathematical subtraction. What is $-5-x$ in the usual base 10 notation if you use signed 8-bit subtraction?

(a) Answer:

(b) Answer:

(c) Answer:

(d) Answer:

(e) Answer:

Q3.

(a) What integer (written in the usual sign magnitude base 10 representation) is 0xFFFF0001 (as an 32-bit unsigned integer representation)?
(b) What integer (written in the usual sign magnitude base 10 representation) is 0xFFFF0001 (as a two's complement representation of a 32-bit signed int)?
(c) What is the minimum integer (of course using two's complement representation of signed int) you can represent on a 16-bit computer? Write it in the usual sign magnitude base 10 representation. (The PC I had in high school had a 16-bit Intel CPU.)

(a) Answer:

(b) Answer:

(c) Answer:

Q4.

(a) What is the output of this C/C++ code:

```
#include <iostream>
int main()
{
    std::cout << (1234567 ^ 7654321 ^ 1234567) << std::endl;
    return 0;
}
```

Explain!

(b) Simplify the following C++ expression:

```
x ^ (~x)
```

where `x` is a signed `int`. Explain!

(c) Simplify the following C++ expression:

```
x + ~x
```

where `x` is a signed `int`. Explain!

(a) Answer:

(b) Answer:

(c) Answer:

Q5. Implement unsigned int addition, subtraction, multiplication, and division using bit operations and also the signed int version. (The unsigned int operations were in the previous assignment.) Recall that the signed int operations should use the unsigned int versions. Also, following MIPS, you have to detect overflow and if an overflow occurs, you set `error` to `1`. See skeleton `q05s.cpp` at the bottom of this question.

For your `calculator()` function, use the following code (file: `seven_seg.h`) for the simulation of an array of seven segment displays. The accompanying `main.cpp` shows you how to use `seven_seg.h`:

```
// file: main.cpp (for testing seven_seg.h)

#include <iostream>
#include <vector>
#include <boost/range/combine.hpp>
#include "seven_seg.h"

int main()
{
    seg7_t blank = seg7({0,0,0,0,0,0,0});
    seg7_t neg = seg7({0,0,0,1,0,0,0});
    seg7_t one = seg7({0,0,1,0,0,1,0});
    seg7_t two = seg7({1,0,1,1,1,0,1});
    seg7_t six = seg7({1,1,0,1,1,1,1});
    seg7_t seven = seg7({1,0,1,0,0,1,0});
    seg7_t o = seg7({0,0,0,1,1,1,1});
    seg7_t r = seg7({0,0,0,1,1,0,0});
    seg7_t E = seg7({1,1,0,1,1,0,1});
    seg7array({blank, blank, blank, blank, blank, blank, blank, blank,
               neg, one, two, seven, blank, o, r, E}
        );
    return 0;
}
```

```
// file: seven_seg.h

#ifndef SEVEN_SEG_H
#define SEVEN_SEG_H

// See the accompanying main.cpp on how to use the console() function.
//
// The following is a 7 segment display:
//
//    a
//   b c
//    d
//   e f
```

```
//    g
//
// a,b,c,d,e,f,g are all 0 or 1. If all are 1, the display is
//
//    -
//   | |
//    -
//   | |
//    -
//
typedef std::vector< std::string > seg7_t;

void console(std::vector< std::tuple< int, int, std::vector< std::string > > > data)
{
    int rows = 5;
    int cols = 80;
    std::vector< std::string > output(rows, std::string(cols, ' '));
    if (data.size() > 16)
    {
        std::cout << "\nconsole error: data.size() too large\n" << std::endl;
        throw -1;
    }
    for (auto & e: data)
    {
        int r = std::get<0>(e);
        int c = std::get<1>(e);
        std::vector< std::string > ss = std::get<2>(e);
        for (auto & s: ss)
        {
            output[r++].replace(c, s.size(), s);
        }
    }
    for (int i = 0; i < cols + 2; ++i) std::cout << '#';
    std::cout << '\n';
    for (auto & e: output)
    {
        std::cout << '#' << e << '#' << '\n';
    }
    for (int i = 0; i < cols + 2; ++i) std::cout << '#';
    std::cout << '\n';
}

std::vector< std::string > seg7(std::vector< int > in)
{
    std::vector< std::string > s = {" - ",
                                    "| |",
                                    " - ",
                                    "| |",
                                    " - "};
    std::vector< std::pair< int, int > > indices = {{0,1},
                                                    {1,0}, {1,2},
```

```
                                                    {2,1},
                                                    {3,0}, {3,2},
                                                    {4,1}};
    for (auto && e: boost::combine(in, indices))
    {
        int bit;
        std::pair< int, int > rc;
        boost::tie(bit, rc) = e;
        if (bit == 0)
        {
            int r = rc.first, c = rc.second;
            s[r][c] = ' ';
        }
    }
    std::vector< std::string > ret;
    for (auto & e: s)
    {
        ret.push_back(e);
    }
    return ret;
}

void seg7array(std::vector< seg7_t > data)
{
    std::vector< std::tuple< int, int, seg7_t > > t;
    int c = 1;
    for (auto & e: data)
    {
        t.push_back({0, c, e});
        c += 5;
    }
    console(t);
}

#endif
```

Here's the output of `main.cpp`:

```
#############################################################################################
#                                                  -    -                   -  #
#                                              |    |    |                |   #
#                                        -         -              -    -    -  #
#                                              |  |      |       | |  |    |   #
#                                                 -             -         -  #
#############################################################################################
```

For this question, put all the code (the given skeleton code and `seven_seg.h`) into `q05s.cpp`. Make sure you can compile and run just `q05s.cpp`.

```
// file: q05s.cpp
#include <iostream>
#include <string>

typedef uint32_t bit; // only least significant bit of uint32_t should be used
typedef uint32_t word;

void readbits(word * x);
void printbit(bit x);
void printbits(word x);
void bit_half_adder(bit * s, bit * c, bit x, bit y);   // 1-bit half adder
void bit_full_adder(bit * s, bit * c1, bit x, bit y, bit c0);   // 1-bit full adder
void addu (word * ret, word x, word y);           // 32-bit unsigned int addition
void comp (word * ret, word x);                   // 32-bit two's complement
void subu (word * ret, word x, word y);           // 32-bit unsigned int subtraction
void multu(word * HI, word * LO, word x, word y); // 32-bit unsigned int mult w
word divu (word * r,  word * q, word x, word y);  // 32-bit unsigned int div


bit  error = 0;                                  // error: set to 1 if overflow,
                                                 //        otherwise set to 0
void add (word * ret, word x, word y);           // 32-bit signed int addition
void sub (word * ret, word x, word y);           // 32-bit signed int subtraction
void mult(word * HI, word * LO, word x, word y); // 32-bit signed int mult w
word div (word * r,  word * q, word x, word y);  // 32-bit signed int div

void readbits(word * x)
{
    std::string s;
    std::cin >> s;
    auto p = s.begin();
    for (int i = 31; i >= 0; --i)
    {
        int b = (*p++ == '0' ? 0 : 1);
        *x |= (b << i);
    }
}

void readbit(word * x)
{
    std::string s;
    std::cin >> s;
    int b = (s[0] == '0' ? 0 : 1);
    *x |= b;
}

void printbits(word x)
{
    for (int i = 31; i >= 0; --i)
    {
```

```
        std::cout << ((x >> i) & 1);
    }
    std::cout << ' ' << x;
    std::cout << '\n';
}

void printbit(word x)
{
    std::cout << (x & 1) << '\n';
}

void addu (word * ret, word x, word y)
{
}

void calculator()
{
    // USAGE:
    // 1. User enters "123+456" or "123-456" or "-123--456" or "-123*-456".
    //    Note: no spaces. All integers are treated as signed 32-bit integers.
    // 2. Calculator prints the result. Display is right justify in the array
    //    of 7 segment display. If error bit is 1, print "Error" in the array
    //    of 7 segment display. Print the result as signed 32-bit integer.
    //    In particular, for mult, print LO. For div, print q (the quotient).
}

int main()
{
    bit s = 0, c = 0, a = 0, b = 0;
    word w = 0, x = 0, y = 0, z = 0;

    int option;
    std::cin >> option;
    switch (option)
    {
        case 0: // test readbits and printbits
            readbits(&w);
            printbits(w);
            break;
        case 1: // test readbit and printbit
            readbit(&w);
            printbit(w);
            break;
        /*
        case 2: // test addu
            readbits(&x);
            readbits(&y);
            addu(&w, x, y);
            printbits(w);
            break;
        case 3: // test comp
```

```
            readbits(&x);
            comp(&w, x);
            printbits(w);
            break;
        case 4: // test subu
        case 5: // test multu (print HI and then LO)
        case 6: // test divu (print r and then q)

        case 7: // test add
        case 8: // test sub
        case 9: // test mult
        case 10: // test div

        case 11:
            calculator();
            break;
        */
    }

    return 0;
}
```

Q6. [This is an optional DIY just for fun]

This is a challenging and a "thinking" problem.

Planet Troisdoigts has just heard your presentation on using two's complement representation to represent 32-bit signed integers. They are impressed and want the same thing for their computer systems. But ... they prefer base 3. Since you are an expert on two's complement, they have commissioned your company to design a base-3 computer system that uses what they are calling the "three's complement" representation for representing signed integers. The commission fee was high and delicious. So your boss has accepted the project and insists that you design the system.

Suppose you are using 3 base-3 symbols. (So there are 27 patterns.)

(a) How would you represent positive and negative numbers? What negative numbers are represented? What positive numbers are represented? Draw a table that translate between Earthling signed base-10 numbers and three's complement base-3 numbers.
(b) How would you compute $5 + 2$ (in Earthling notation) in 3 base-3 symbols?
(c) How would you compute, in Earthling notation, the expression $5 - 2$ in 3 base-3 symbols? Of course from your experience with 32-bit two's complement, you would want to compute this using 3 base-3 symbols of addition and also you want to convert 2 to it's "three's complement" of 3 base-3 symbols, and then perform addition on "$5 + (-2)$", so to speak. How would you do it?!?
(d) What about multiplication?