**CISS360: Computer Systems and Assembly Language**
**Assignment a09**

Name: _____

Objectives

- Perform computations involving number representations with fractional parts in different bases without any restriction on size of the representation
- Write C/C++ programs that uses bit programming.

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:

| | | | | | |
|---|---|---|---|---|---|
| | | | 1 | 2 | 3 |
| $\times$ | | | 4 | 5 | 6 |
| | | | 7 | 6 | 2 |
| | | 6 | 3 | 7 | |
| $+$ | 5 | 1 | 4 | | |
| | 6 | 0 | 7 | 5 | 2 |

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 the LaTeX code in `how-to-write-a-multi-column-float-operation.tex`.)

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. Convert 123.567

(a) to base 2
(b) to base 8
(c) to base 16

with up to at least 8 binary/octal/hexadecimal places.

(a) Answer:

1111011.10010001

(b) Answer:

173.442

(c) Answer:

7B.94

Q2.

(a) What is $12502.307_8$ in base 10?
(b) What is $7A9B.C0F4_{16}$ in base 10?
(c) What is $0.1111\cdots_2$ in base 10?
(d) Is $0.2043_5 - 0.2043_6$ greater than 0, less than 0, or equal to zero?
(e) Is it possible that $0.0x_8 > 0.yz_2$? The left-hand-side is a base 8 number and the right-hand-side is a base 2 number. $x$ is a symbol in base 8 (i.e., $0 \leq x < 8$). $y, z$ are symbols in base 2 (i.e., $0 \leq y, z < 2$). If it's possible find all possible $x, y, z$. If not, explain why not.

(a) Answer:

680.38867

(b) Answer:

31387.75372

(c) Answer:

1

(d) Answer:

0.08263

(e) Answer:

the highest value of $x$ will be 7 which will make it 0.07 and that corresponds to 0.109

case 1: $0.00_2 = 0$ $0.109 > 0$
case 2: $0.01_2 = 0.25$ $0.109 < 0.25$

therefore $y = z = 0$ and $x = 7$

Q3. Compute the following:

(a) $101.1101_2 + 11.011_2$
(b) $101.1101_2 - 11.011101_2$
(c) $101.1101_2 \times 11.011_2$
(d) $101.1101_2 / 11.011_2$ (up to 8 places after the binary point)

(a) Answer:

$1001.0011_2$

(b) Answer:

$10.0111_2$

(c) Answer:

$10011.1001111_2$

(d) Answer:

1.101100100

Q4. The following is the famous Collatz conjecture: Let $f(n)$ (where $n$ is a positive integer) be the function

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is odd} \end{cases}$$

(Note that / is the integer division, for instance $15/4 = 3$, i.e. it is not real number division). Then given any positive integer $n$ (greater than 0), with enough applications of function $f$, you will always arrive at 1. For instance if we start with n = 10:

$$\begin{aligned} f(10) &= 10/2 = 5 \\ f(5) &= 3(5)+1 = 16 \\ f(16) &= 16/2 = 8 \\ f(8) &= 8/2 = 4 \\ f(4) &= 4/2 = 2 \\ f(2) &= 2/2 = 1 \end{aligned}$$

This 86-year-old conjecture is still an unsolved problem. You can write a simple function to compute $f$. Here's the pseudocode:

```
function f(n):
    if n is divisible by 2 then
        return n / 2
    otherwise
        return 3 * n + 1
```

and to count the number of steps to get to 1, we then do the following:

```
function steps(n):
    int num_steps = 0;
    while n is not 1:
        n = f(n)
        num_steps = num_steps + 1
    return num_steps
```

Of course you can put everything into a single function:

```
function steps(n):
    int num_steps = 0;
    while n is not 1:
        if n is divisible by 2 then
            n = n / 2
        else
            n = 3 * n + 1
        num_steps = num_steps + 1
    return num_steps
```

The goal of this exercise is to use C/C++ bit operations to improve on the speed of

the computation. (There are other optimizations besides using bit operations but we will not worry about those techniques here.)

Note that if you know that $n$ is divisible by 4 you can compute two steps in one. With this improvement you have this:

```
function steps(n):
    int num_steps = 0;
    while n is not 1:
        q = n / 4 and r = n % 4
        if r is 0, then
            n = n / 4
            num_steps = num_steps + 2
        else if r is 1, then
            ...
        else if r is 2, then
            ...
        else if r is 3, then
            ...
    return num_steps
```

This pseudocode would then allow you to reach 1 (from n) faster. The goal is to implement this strategy in a C/C++ program. In your code you cannot use `*` or `/` or `%`, i.e. you can use the following arithmetic and bitwise operators: `+`, `-`, `&`, `|`, `^`, `>>`, `<<` and their augmented version, i.e. `+=`, `-=`, `&=`, `|=`, `^=`, `>>=`, `<<=`. Make your program as efficient as possible based on the above methods. Do not include methods not described above. To test your program you should work out by hand the numbers of steps to reach 1 for n = 1, 2, 3, ..., 20 and run your program against the values computed by hand.

Test 1

```
10
6
```

(The user enters 10 and the program prints 6, i.e., after 6 steps we get 1.)

Put your C++ code in `q04s.tex`.

Note.

- This is the end of this problem for us, but the story of this conjecture goes on. The above is not the only optimization technique relevant to this program. The point is to show you that basic math and some bit programming can help in speeding up certain types of computations.
- You should write the obvious version without bitwise optimization and without the mod 4 optimization and compare the runtime of the two functions.

```
#include <iostream>

int steps(int n)
{
    int num_steps = 0;

    while (n != 1)
    {
        int mask = 3;
        int r = n & mask;
        if (r == 0)
        {
            n >>= 2;
            num_steps = num_steps + 2;
        }
        else if (r == 1)
        {
            int x = n;
            n = (n << 1) + x + 1;
            n >>= 2;
            num_steps += 3;
        }
        else if (r == 2)
        {
            n >>= 1;
            int x = n;
            n = ((n << 1)) + x + 1;
            num_steps += 2;
        }
        else
        {
            int x = n;
            n = (n << 1) + x + 1;
            n >>= 1;
            num_steps += 2;
        }
        // if (n % 2 == 0)
        // {
        //     n = n  / 2;
        // }
        // else
        // {
        //     n = 3 * n + 1;
        // }
        // num_steps = num_steps + 1;
    }
    return num_steps;
}

int main()
```

```
{
    int n;
    std::cin >> n;
    std::cout << steps(n) << std::endl;
    return 0;
}
```

Q5. Implement unsigned int addition, subtraction, multiplication, and division using bit operations.

See skeleton `q05s.cpp`.

```
#include <iostream>
#include <string>
#include <cstdint>

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

void readbits(word * x);
void printbit(word x);
void printbits(word x);
void bit_adder(word * s, word * c, word x, word y);   // 1-bit full adder
bit 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
void divu(word * r,  word * q, word x, word y);  // 32-bit unsigned int div

void half_adder(word * s, word * c, word x, word y)
{
    *s = (~x & y) | (x & ~y);

    *c = x & y;

}
void bit_adder(word * s, word * c, word x, word y, word z)
{
    word c0;
    word c1;
    word s0;
    half_adder(&s0, &c0, x, y);
    half_adder(s, &c1, s0, z);
    *c = c0 | c1;
}

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

```
    }
}

void readbit(word * x)
{
    std::string s;
    std::cin >> s;
    std::cout << s << std::endl;
    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';
}

bit addu(word * ret, word x, word y)
{
    bit c = 0;
    *ret = 0;
    for (int i = 0; i < 32; ++i)
    {

        bit c_i  = 0;
        bit s_i = 0;
        bit_adder(&s_i, &c_i, c, (x & 1), (y & 1));
        x >>= 1;
        y >>= 1;
        c = c_i;
        *ret |= (s_i << i);
    }
    return c;
}

void subu (word * ret, word x, word y)
{
    // ret = x + (2^32 - y)
    printbits(x);
    printbits(y);
```

```
    addu(ret, x, (~y + 1));
}

void multu(word * HI, word * LO, word x, word y)
{
    *HI = 0;
    *LO = 0;
    word hi, lo;
    lo = x;
    hi = 0;
    printbits(x);
    printbits(y);
    for (int i = 0; i < 64; ++i)
    {
        bit c = addu(LO, *LO, lo * (y & 1));
        hi <<= 1;
        hi |= (lo >> 31);
        lo <<= 1;
        addu(HI, *HI + c, hi * (y & 1));
        y >>= 1;
    }
}

void divu(word * r,  word * q, word x, word y)
{
    *q = 0;
    *r = x;
    word sum = y;
    while (x > sum)
    {
        sum << 1;
    }
    while (sum >= y)
    {
        word mult;
        if (*r >= x)
        {
            mult = 1;
        }
        else
            mult = 0;
        *q <<= (1|mult);
        *r -= sum * mult;
        sum >>=1;
    }
}

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

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

    return 0;
}
```

## Instructions

In `main.tex` change the email address in

```
\renewcommand\AUTHOR{jdoe5@cougars.ccis.edu}
```

yours. In the bash shell, execute "`make`" to recompile `main.pdf`. Execute "`make v`" to view `main.pdf`. Execute "`make s`" to create `submit.tar.gz` for submission.

For each question, you'll see boxes for you to fill. You write your answers in `main.tex` file. For small boxes, if you see

```
1 + 1 = \answerbox{}.
```

you do this:

```
1 + 1 = \answerbox{2}.
```

`answerbox` will also appear in "true/false" and "multiple-choice" questions.

For longer answers that needs typewriter font, if you see

```
Write a C++ statement that declares an integer variable name x.
\begin{answercode}
\end{answercode}
```

you do this:

```
Write a C++ statement that declares an integer variable name x.
\begin{answercode}
int x;
\end{answercode}
```

`answercode` will appear in questions asking for code, algorithm, and program output. In this case, indentation and spacing is significant. For program output, I do look at spaces and newlines.

For long answers (not in typewriter font) if you see

```
What is the color of the sky?
\begin{answerlong}
\end{answerlong}
```

you can write

```
What is the color of the sky?
\begin{answerlong}
The color of the sky is blue.
\end{answerlong}
```

For students beyond 245: You can put LaTeX commands in `answerlong`.

A question that begins with "T or F or M" requires you to identify whether it is true or false, or meaningless. "Meaningless" means something's wrong with the statement and it is not well-defined. Something like "$1+_2$" or "$\{2\}^{\{3\}}$" is not well-defined. Therefore a question such as "Is $42 = 1+_2$ true or false?" or "Is $42 = \{2\}^{\{3\}}$ true or false?" does not make sense. "Is $P(42) = \{42\}$ true or false?" is meaningless because $P(X)$ is only defined if $X$ is a set. For "Is $1 + 2 + 3$ true or false?", "$1 + 2 + 3$" is well–defined but as a "numerical expression", not as a "proposition", i.e., it cannot be true or false. Therefore "Is $1 + 2 + 3$ true or false?" is also not a well-defined question.

When writing results of computations, make sure it's simplified. For instance write 2 instead of $1 + 1$. When you write down sets, if the answer is $\{1\}$, I do not want to see $\{1, 1\}$.

When writing a counterexample, always write the simplest.

Here are some examples (see `instructions.tex` for details):

1. T or F or M: $1 + 1 = 2$ ..... T
2. T or F or M: $1 + 1 = 3$ ..... F
3. T or F or M: $1+^2 =$ ..... M
4. $1 + 2 =$ 3
5. Write a C++ statement to declare an integer variable named `x`.
   ```
   int x;
   ```
6. Solve $x^2 - 1 = 0$.

   Since $x^2 - 1 = (x - 1)(x + 1)$, $x^2 - 1 = 0$ implies $(x - 1)(x + 1) = 0$. Therefore $x - 1 = 0$ or $x = -1$. Hence $x = 1$ or $x = -1$.

7. Which is true? ..... C
   (A) $1 + 1 = 0$
   (B) $1 + 1 = 1$
   (C) $1 + 1 = 2$
   (D) $1 + 1 = 3$
   (E) $1 + 1 = 4$