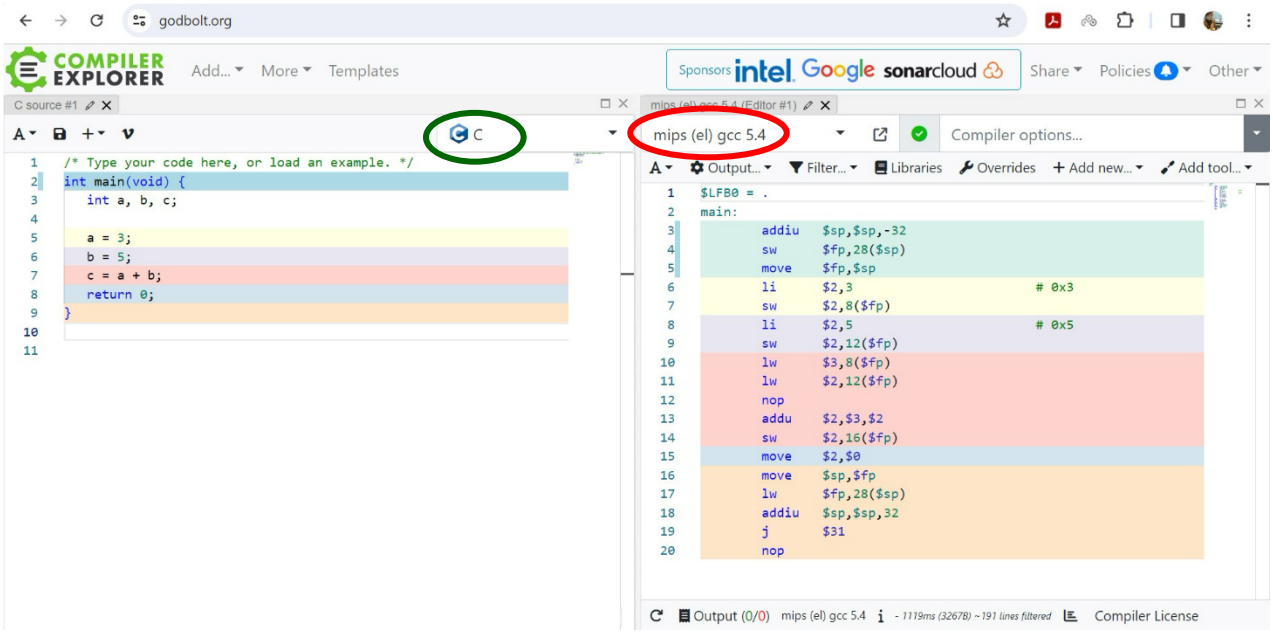
CS2100 Computer Organization
Tutorial #2: C and MIPS

Exploration: C to MIPS

Go to this website https://godbolt.org/ and copy the C code below into the left box, and ensure that you choose "C" in the dropdown list (circled green), and choose "mips (el) gcc 5.4" or "mips gcc 5.4" in the dropdown list (circled red). (Do not choose "mips64 gcc 5.4".)

```
int main(void) {
    int a, b, c;

    a = 3;
    b = 5;
    c = a + b;
    return 0;
}
```



This is just for your exploration.

We cover **sw**, **lw** and **j** in class. Some of the MIPS instructions such as **addiu** and **addu** shown above are not covered; instead, we cover **add** and **addi**. The **nop** instruction will be mentioned in the topic on Pipelining later. **move** and **li** are pseudo-instructions. In general, we do not use pseudo-instructions, except for **li** and **la** which you will learn in your labs.

You will use QTSpim, a MIPS simulator, for your labs later.

Self-Check (these will not be covered in tutorials. You may discuss this on Canvas or QnA if you have any queries):

For each of the following instructions, indicate if it is valid or not (refer to the comment for the intention). If not, explain why and suggest a correction. Note that the "|" in the comment in (d) is the bitwise OR operation.

Note: 0x indicates hexadecimal value; 0b indicates binary value. Examples: 0x12 = 18_{10}; 0b1101 = 13_{10}.

a. `add  $t1, $t2, $t3` # $t3 = $t1 + $t2
b. `addi $t1, $0, 0x25` # $t1 = 37
c. `subi $t2, $t1, 3` # $t2 = $t1 – 3
d. `ori  $t3, $t4, 0xAC120000` # $t3 = $t4 | 0xAC120000
e. `sll  $t5, $t2, 0x21` # shift left $t2 33 bits and store in $t5

Tutorial questions:

1. **C bitwise operations**

   Find out about the following bitwise operations in C.

   - | (bitwise OR)
   - & (bitwise AND)
   - ^ (bitwise XOR)
   - ~ (one's complement)
   - << (left shift)
   - >> (right shift)

   Using the following C program as a template, illustrate the above bitwise operations with your own examples. Note that variables of the data type **`char`** take up one byte (8 bits) of memory.

```
#include <stdio.h>

typedef unsigned char byte_t;

void printByte(byte_t);

int main(void) {
   byte_t a, b;

   a = 5;
   b = 22;
   printf("a    = "); printByte(a);    printf("\n");
   printf("b    = "); printByte(b);    printf("\n");
   printf("a|b  = "); printByte(a|b);  printf("\n");
   return 0;
}

void printByte(byte_t x) {
   printf("%c%c%c%c%c%c%c%c",
          (x & 0x80 ? '1': '0'),
          (x & 0x40 ? '1': '0'),
          (x & 0x20 ? '1': '0'),
```

```
            (x & 0x10 ? '1': '0'),
            (x & 0x08 ? '1': '0'),
            (x & 0x04 ? '1': '0'),
            (x & 0x02 ? '1': '0'),
            (x & 0x01 ? '1': '0'));
}
```

2. **MIPS Bitwise Operations**

Implement the following in MIPS assembly. Assume that integer variables **a**, **b** and **c** are mapped to registers $s0, $s1 and $s2 respectively. Parts (a), (b), (c) are independent of one another.

For bitwise instructions (e.g. **ori**, **andi**, etc), any immediate values you use should be written in binary for this question. This is optional for non-bitwise instructions (e.g. **addi**).

Note that bit 31 is the most significant bit (MSB) on the left, and bit 0 is the least significant bit (LSB) on the right.

(a) Set bits 2, 8, 9, 14 and 16 of **b** to 1. Leave all other bits unchanged.

(b) Copy over bits 1, 3 and 7 of **b** into **a**, without changing any other bits of **a**.

(c) Make bits 2, 4 and 8 of **c** the inverse of bits 1, 3 and 7 of **b** (i.e. if bit 1 of **b** is 0, then bit 2 of **c** should be 1; if bit 1 of **b** is 1, then bit 2 of **c** should be 0), without changing any other bits of **c**.

3. **MIPS Arithmetic**

Write the following in MIPS Assembly, using as few instructions as possible. You may rewrite the equations if necessary to minimize instructions.

In all parts you can assume that integer variables **a**, **b**, **c** and **d** are mapped to registers $s0, $s1, $s2 and $s3 respectively. Each part is independent of the others.

(a) c = a + b;

(b) d = a + b – c;

(c) c = 2b + (a – 2);

(d) d = 6a + 3(b – 2c);

4. [AY2013/14 Semester 2 Exam]
The mysterious MIPS code below assumes that **$s0 is a 31-bit binary sequence**, i.e. the MSB (most significant bit) of **$s0** is assumed to be zero at the start of the code.

```
     add   $t0, $s0, $zero  # make a copy of $s0 in $t0
     lui   $t1, 0x8000
lp:  beq   $t0, $zero, e
     andi  $t2, $t0, 1
     beq   $t2, $zero, s
     xor   $s0, $s0, $t1
s:   srl   $t0, $t0, 1
     j     lp
e:
```

(a) For each of the following initial values in register **$s0** at the beginning of the code, give the hexadecimal value of the content in register **$s0** at the end of the code.

(i) Decimal value 31.

(ii) Hexadecimal value 0x0AAAAAAA.

(b) Explain the purpose of the code in one sentence.