CS2100 Computer Organization
Tutorial 2

Discussion Questions

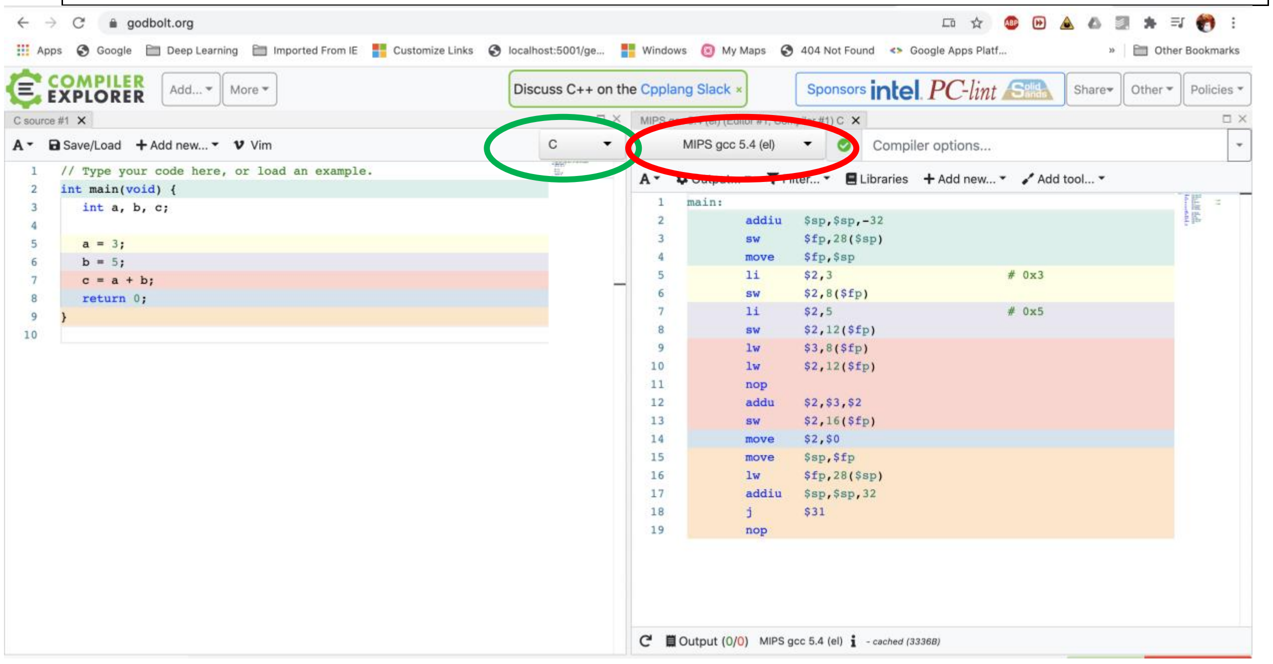
Discuss these on the Canvas Forum. Answers will not be given.

D1. **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 in green in the screenshot after the C code, and choose MIPS gcc 5.4 (el) or MIPS gcc 5.4 in the dropdown list circled red. (do not choose MIPS64 gcc 5.4 or MIPS64 gcc 5.4 (el):

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

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



C

MIPS gcc 5.4 (el)

```
// Type your code here, or load an example.
int main(void) {
  int a, b, c;

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

```
main:
        addiu   $sp,$sp,-32
        sw      $fp,28($sp)
        move    $fp,$sp
        li      $2,3                    # 0x3
        sw      $2,8($fp)
        li      $2,5                    # 0x5
        sw      $2,12($fp)
        lw      $3,8($fp)
        lw      $2,12($fp)
        nop
        addu    $2,$3,$2
        sw      $2,16($fp)
        move    $2,$0
        move    $sp,$fp
        lw      $fp,28($sp)
        addiu   $sp,$sp,32
        j       $31
        nop
```

Output (0/0) MIPS gcc 5.4 (el) - cached (3336B)

The following is extracted from the Assembly output.

```
    addiu   $sp,$sp,-32
    sw      $fp,28($sp)
    move    $fp,$sp
    li      $2,3                    # 0x3
    sw      $2,8($fp)
    movz    $31,$31,$0
    li      $2,5                    # 0x5
    sw      $2,12($fp)
    lw      $3,8($fp)
    lw      $2,12($fp)
    nop
    addu    $2,$3,$2
    sw      $2,16($fp)
    move    $2,$0
    move    $sp,$fp
    lw      $fp,28($sp)
    addiu   $sp,$sp,32
    j       $31
    nop
```

We covered **sw**, **lw** and **j** in lecture, but not the rest. Find out what they are. (Note: **li** will be used in the labs later and **nop** will be mentioned in the topic on Pipelining. **move**, like **li**, is a pseudo-instruction. **$sp**, **$fp**, **movz**, **addiu**, and **addu** are not in the syllabus.)

D2. For each of the following instructions, indicate if it is valid or not. If not, explain why and suggest a correction. Note that the | in (d) is the bitwise OR operation.

a. **`add  $t1, $t2, $t3`** # $t3 = $t1 + $t2
b. **`addi $t1, $0, 0x25`** # $t1 = 0x25
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

1. **Bitwise operations**

Find out about the following bitwise operations in C and explain and illustrate each of them with an example.

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

You may use the following code template for your illustration. Variables of the data type **`char`** take up 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. **Swapping**

We have discussed in lecture how to swap two variables in a function:

```
void swap(int *a, int *b) {
   int t = *a;
   *a = *b;
   *b = t;
}
```

In the above code, a temporary variable `t` is used.

A possible alternative is to use some bitwise operator to perform the swap, without using any temporary variable. Write a function to do this.

3. **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. Each part is independent of all the other parts. **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, etc).

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, i..e.:

MSB					LSB
Bit 31	Bit 30	Bit 29	…	Bit 1	Bit 0

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**.

4. **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)

5. [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.