

# CS2100 Computer Organization

## Tutorial 2: C and MIPS

### SUGGESTED SOLUTIONS

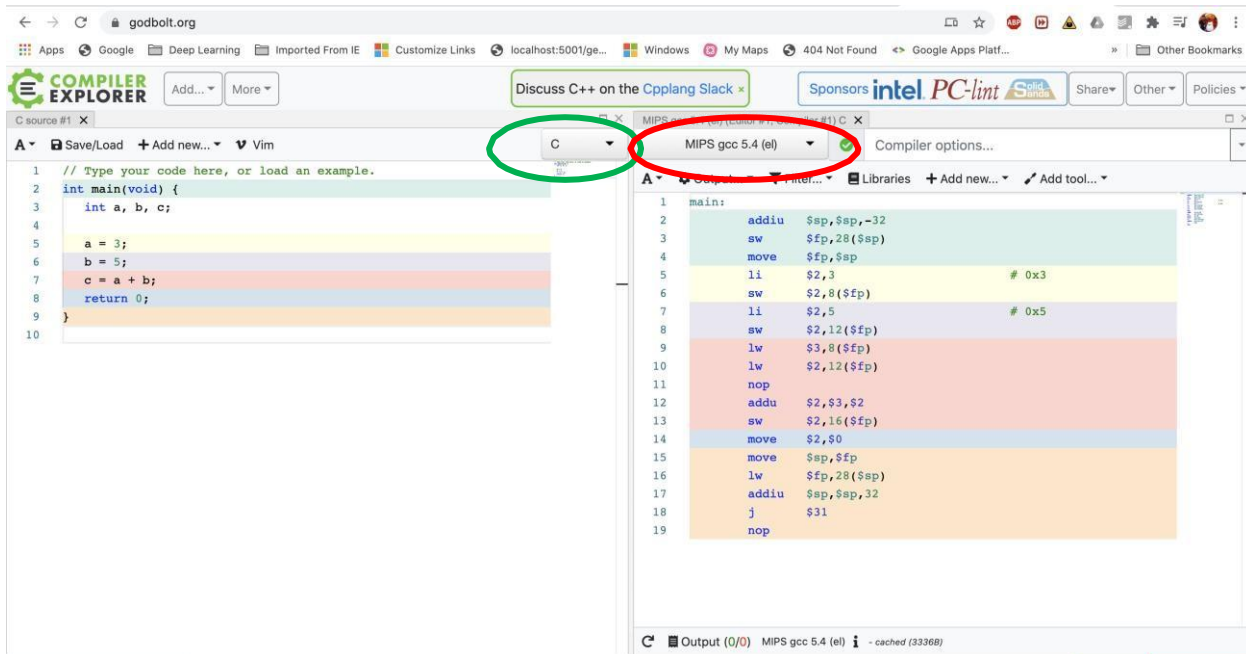
#### Discussion Questions

You may 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;  
}
```



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

---

Answers:

- a. Wrong source and destination registers: **add** \$t3, \$t1, \$t2
- b. Valid
- c. No **subi** instruction in MIPS, use **addi**: **addi** \$t2, \$t1, -3
- d. Immediate operand is 16-bit long only  
    **lui** \$t0, 0xAC12  
    **or** \$t3, \$t4, \$t0
- e. Shift distance must be in the range [0, 31]. Since shifting 33 bits to the left is equivalent to shifting 1 bit to the left, we can correct the instruction to:  
    **sll** \$t5, \$t2, 1

## Tutorial Questions

### 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'));
}
```

Note to tutors:

1. Bitwise operators not covered in lecture. Hence this tutorial question for them to explore (and relate to MIPS instructions AND, XOR, etc. later.)
2. Conditional operator **?:** not covered in lecture. Explain to students if necessary.
3. Ask students how **<<** and **>>** are related to arithmetic operations (answer: multiplication and division by powers of 2) and why they are more efficient.

## 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. What is the limitation of this new function?

Answer: Tut2Q2.c

Limitation: This new function cannot be used for general data types, eg: real numbers, strings, etc.

```
void swap(int *a, int *b) {  
    *a = *a ^ *b;  
    // Now, we can recover *a_orig by applying *a XOR *b_orig  
    *b = *a ^ *b;  
    // So we have the following situation:  
    // The value originally stored in *a, a_orig, is now in *b  
    // and *a still stores a_orig XOR b_orig  
    // This means that we can recover the value of b_orig by  
    // applying the XOR operation to *a and a_orig.  
    // Since *b stores a_orig...  
    *a = *a ^ *b;  
}
```

## 3. MIPS Bitwise Operations

Implement the following in MIPS assembly code. 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

- Set bits 2, 8, 9, 14 and 16 of `b` to 1. Leave all other bits unchanged.
- Copy over bits 1, 3 and 7 of `b` into `a`, without changing any other bits of `a`.
- 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`.

Answers:

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

To set bits, we create a “mask” with 1’s in the bit positions we want to set. Since bit 16 is in the upper 16 bits of the register, we need to use **lui** to set it.

```
lui $t0, 1           # Sets bit 16 of $t0.
ori $t0, $t0, 0b0100001100000100  # Set bits 14, 9, 8 and 2.
or  $s1, $s1, $t0
```

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

We use the property that  $x \text{ AND } 1 = x$  to copy out the values of bits 7, 3 and 1 of **b** into \$t0. Note that we zero all other bits so that they don’t change anything in \$s0 when we OR later on.

```
andi $t0, $s1, 0b0000000010001010
```

We use the property of  $x \text{ OR } 0 = x$  to copy in the bits into **a**, so we prepare **a** by zero-ing bits 7, 3 and 1. To do this we need the mask 1111111111111111 111111101110101.

```
lui  $t1, 0b1111111111111111
ori  $t1, $t1, 0b1111111101110101
and  $s0, $s0, $t1
```

Now OR together **a** and \$t0 to copy over the bits

```
or  $s0, $s0, $t0
```

- 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**.

We use the property that  $x \text{ XOR } 1 = \sim x$  to flip the values of bits 7, 3 and 1.

```
xori $t0, $s1, 0b10001010
```

Zero every bit except 7, 3 and 1.

```
andi $t0, $t0, 0b10001010
```

Shift left one position.

```
sll  $t0, $t0, 1
```

Now, to clear bits 8, 4 and 2 of **c**, we need the mask 0b1111111111111111 1111111011101011

```
lui  $t1, 0b1111111111111111
ori  $t1, $t1, 0b1111111101110101
and  $s2, $s2, $t1
```

And we OR the new **c** with \$t0

```
or  $s2, $s2, $t0
```

#### 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.  $d = 6a + 3(b - 2c)$

Answers:

- a.  $c = a + b$

```
add $s2, $s0, $s1
```

- b.  $d = a + b - c$

```
add $s3, $s0, $s1 # d = a + b
sub $s3, $s3, $s2 # d = (a + b) - c
```

- c.  $c = 2b + (a - 2)$

```
add $s2, $s1, $s1 # c = 2b (alternatively, can
                  # do a shift left 1 bit)
addi $t0, $s0, -2 # $t0 = a - 2
add $s2, $s2, $t0 # c = 2b + (a - 2)
```

- d.  $d = 6a + 3(b - 2c)$

Note to tutors: Students may find better solutions than this. Check to ensure that they achieve the equation above.

Rewrite:  $d = 6a + 3b - 6c$

Factorize out 3:  $d = 3(2a + b - 2c) = 3(2a - 2c + b) = 3(2(a - c) + b)$

```
sub $t0, $s0, $s2 # t0 = a - c
sll $t0, $t0, 1   # t0 = 2(a - c)
add $t0, $t0, $s1 # t0 = 2(a - c) + b
sll $t1, $t0, 2   # t1 = 4(2(a - c) + b)
sub $s3, $t1, $t0 # d = 3(2(a - c) + b)
```

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 **0x0AAAAAA**.
- b. Explain the purpose of the code in one sentence.

Answers:

- a.
  - i. **\$s0 = 0x8000 001F**
  - ii. **\$s0 = 0x0AAA AAAA**
- b. The code sets bit 31 of \$s0 to 1 if there are odd number of '1' in \$s0 initially, or 0 if there are even number of '1'. (This is called the even parity scheme.)