CS2100

TUTORIAL #2

C AND MIPS

(PREPARED BY: AARON TAN)

Q1. Bitwise operations

	a = 5;	00000101	
	b = 22;	00010110	
| (bitwise OR)	a|b	00010111	
& (bitwise AND)	a&b	00000100	
^ (bitwise XOR)	a^b	00010011	
~ (1's complement)	~a	11111010	
<< (left shift)	b<<2	01011000	(22×4 = 88)
>> (right shift)	b>>3	00000010	(22÷8 = 2)

Q1. What is ? :

Conditional operator

condition ? true-part : false-part

If the condition is true, it returns the true-part value;
otherwise, it returns the false-part value.

Example:

```
int a, b, c;
…
c = (a>b ? a+100 : b-10);
```

If a=12, b=34, then c becomes … 24

If a=34, b=12, then c becomes … 134

Q2. Swapping.

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

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

If this code saves a temporary variable, why not always use this method? What are the constraints of this method?

Consider what happens if a and b point to the same address

Example:

```
int x = 5;
int y = 22;
swap(&x, &y);
```

0000…00000101

0000…00010110

*a ~~0000…00000101~~

~~0000…00010011~~

0000…00010110

*b ~~0000…00010110~~

0000…00000101

Q3. (a) Set bits 2, 8, 9, 14 and 16 of *b* to 1. Leave the other bits unchanged.

Recall:
x OR 0 = *x*
x OR 1 = 1

Example: Before

b = 00110000011100000110110100100001.

$s1 (Bits 2, 8, 9, 14 and 16 are underlined.)

After

b = 00110000011100010110111100100101.

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

Q3. (b) Copy over bits 1, 3 and 7 of *b* into *a*, without changing any other bits of *a*.

Recall:
x AND 0 = 0
x AND 1 = *x*

Recall:
x OR 0 = *x*
x OR 1 = 1

Example: Before (assume that the most significant 24 bits are all zeroes)

$s0 *a* = 0 0 … 0 0 0 0 1 0 1 0 1 0.

$s1 *b* = 0 0 … 0 0 1 1 0 1 1 1 0 0.

After

a = 0 0 … 0 0 1 0 1 0 1 0 0 0.

b = 0 0 … 0 0 1 1 0 1 1 1 0 0.

```
andi  $t0, $s1, 0b0000000010001010
lui   $t1, 0b1111111111111111
ori   $t1, $t1, 0b1111111101110101
and   $s0, $s0, $t1
or    $s0, $s0, $t0
```

Q3. (c) Make bits 2, 4 and 8 of *c* the inverse of bits 1, 3 and 7 of *b*.

Recall:
x XOR 0 = *x*
x XOR 1 = *x*′

Example: Before (assume that the … part are all zeroes)

$s1 *b* = 0 0 … 0 0 1 1 0 1 1 1 0 0.

$s2 *c* = 0 0 … 0 1 0 0 1 0 1 0 1 0.

After

b = 0 0 … 0 0 1 1 0 1 1 1 0 0.

c = 0 0 … 0 0 0 0 1 0 1 1 1 0.

```
xori  $t0, $s1, 0b10001010
andi $t0, $t0, 0b10001010
sll    $t0, $t0, 1
lui    $t1, 0b1111111111111111
ori    $t1, $t1 0b1111111011101011
and   $s2, $s2, $t1
or     $s2, $s2, $t0
```

Q4. (a) c = a + b

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

a: $s0
b: $s1
c: $s2
d: $s3

Q4. (b) d = a + b - c

```
add  $s3, $s0, $s1
sub  $s3, $s3, $s2
```

a: $s0
b: $s1
c: $s2
d: $s3

Q4. (c) c = 2b + (a – 2)

```
sll  $s2, $s1, 1
addi $t0, $s0, -2
add  $s2, $s2, $t0
```

a: $s0
b: $s1
c: $s2
d: $s3

Q4. (d) $d = 6a + 3(b - 2c)$

```
sub  $t0, $s0, $s2
sll  $t0, $t0, 1
add  $t0, $t0, $s1
sll  $s3, $t0, 2
sub  $s3, $s3, $t0
```

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

a: $s0
b: $s1
c: $s2
d: $s3

Q5. $s0 is a 31-bit binary sequence with MSB=0.

```
    add  $t0, $s0, $zero
    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:
```

Loop init

"Loop cond"

"If block"

Loop update

Need to trace the MIPS code with an example to find out what it does.

Q5. $s0 is a 31-bit binary sequence with MSB=0.

```
    add  $t0, $s0, $zero
    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:
```

Example: $s0 = 31

MSB

$s0 = ~~0~~ 1 0 … 0 0 0 1 1 1 1 1

$t0 = ~~0 0 … 0 0 0 1 1 1 1 1~~

0 0 … 0 0 0 0 1 1 1 1

$t1 = 1000 0000 0000 0000 0000 0000 0000 0000

$t2 = 0 0 … 0 0 0 0 0 0 0 1

Q: What is the final value of $s0 (in hexadecimal)?

0x8000001F

What does **andi** do?

Extracts the LSB of $t0 into $t2

Q: At most how many iterations? 31

Q: What happens in each iteration?

Toggle the MSB of $s0 if $t2 = 1.

What does **xor** do?

It toggles the MSB of $s0 (why?)

All other bits in $t1 are 0s, so keep the bits from $s0

Q5. $s0 is a 31-bit binary sequence with MSB=0.

Example:
$s0 = 0x0AAAAAAA

```
    add  $t0, $s0, $zero
    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:
```

MSB

$s0 = 0 00010101010101010101010101010101

$t0 = 0 00010101010101010101010101010101

$t1 = 1000 0000 0000 0000 0000 0000 0000 0000

Q: What is the final value of $s0 (in hexadecimal)?

0x0AAAAAAA

Q5. $s0 is a 31-bit binary sequence with MSB=0.

Summary

```
     add  $t0, $s0, $zero
     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:
```

Even parity scheme. We set the parity bit so that the number of 1's is always even.

$s0 = 31

Before:

Odd number of 1's in $s0: MSB → 1

MSB

$s0 = 0 00000000000000000000000011111

After:

$s0 = 1 00000000000000000000000011111

$s0 = 0x0AAAAAAA

Before:

Even number of 1's in $s0: MSB → 0.

$s0 = 0 00010101010101010101010101010

After:

$s0 = 0 00010101010101010101010101010

END OF FILE

Additional Question 1

Suppose I was writing the code from question 3 part (c) of the question but I made a mistake and wrote the following code instead (mistake highlighted).

How would this affect what the code does?

```
xori  $t0, $s1, 0b10001010
andi $t0, $t0, 0b10001010
sll    $t0, $t0, 1
lui    $t1, $t1, 0b1111111111111110
ori    $t1, $t1 0b1111111011101011
and   $s2, $s2, $t1
or     $s2, $s2, $t0
```

Additional Answer 1

Suppose I was writing the code from question 3 part (c) of the question but I made a mistake and wrote the following code instead (mistake highlighted).

How would this affect what the code does?

Make bits 2, 4 and 8 of *c* the inverse of bits 1, 3 and 7 of *b* and make bit 16 of *c* 0.

```
xori  $t0, $s1, 0b10001010
andi $t0, $t0, 0b10001010
sll    $t0, $t0, 1
lui    $t1, $t1, 0b1111111111111110
ori    $t1, $t1 0b1111111011101011
and   $s2, $s2, $t1
or     $s2, $s2, $t0
```

Additional Question 2

Implement the following in MIPS Assembly:

d = 7a + 42b

Additional Answer

a: $s0
b: $s1
c: $s2
d: $s3

Implement the following in MIPS Assembly:

d = 7a + 42b

```
    sll  $t0, $s1, 2
    sll  $t1, $s1, 1
    add  $t0, $t0, $t1
    add  $t0, $t0, $s0
    sll  $t1, $t0, 3
    sub  $s3, $t1, $t0
```

Additional Question 3

How can the code I question 5 be modified to set MSB to 1 if there is an even number of 1s, otherwise 0?

Additional Answer 3

How can the code in question 5 be modified to set MSB to 1 if there is an even number of 1s, otherwise 0?

```
    add  $t0, $s0, $zero
    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:  xor  $s0, $s0, $t1
```

Additional Question 4

Consider the code from question 5 of the tutorial.

What are bits 0 to 15 in $t1?

```
    add  $t0, $s0, $zero
    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:
```

Additional Answer 4

Consider the code from question 5 of the tutorial.

What are bits 0 to 15 in $t1?

All 0s. This is part of the specification for lui.

```
    add  $t0, $s0, $zero
    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:
```