

MIPS Programming

Rathindra Nath Dutta & Subhojit Sarkar

Senior Research Fellow
Advanced Computing & Microelectronics Unit
Indian Statistical Institute, Kolkata



September 1, 2022

Writing Your First MIPS Code

// C code

a = 10

b = 20

c = a + b

⇒

using instructions

li a, 10

li b, 20

add c, a, b

⇒

final MIPS Code

li \$t0, 10

li \$t1, 20

add \$t3, \$t0, \$t1

A MIPS Code Template

```
# Declare main as a global function
.globl main

# All program code is placed after the
# .text assembler directive
.text

# The label 'main' represents the starting point
main:
    # YOUR CODE GOES HERE

    # Exit the program by means of a syscall.
    # by placing its code in $v0. The code for exit is "10"
    li $v0, 10 # exit syscall
    syscall

# All memory structures are placed after the
# .data assembler directive
.data

# The .word assembler directive reserves space
# in memory for one or more 4-byte words
list: .word 1, 4, 8
```

Running a MIPS Code

- Ideally one should execute on a MIPS hardware
- We will be using a free simulator tool: **SPIM**¹

¹more specifically QtSPIM: <http://spimsimulator.sourceforge.net/>

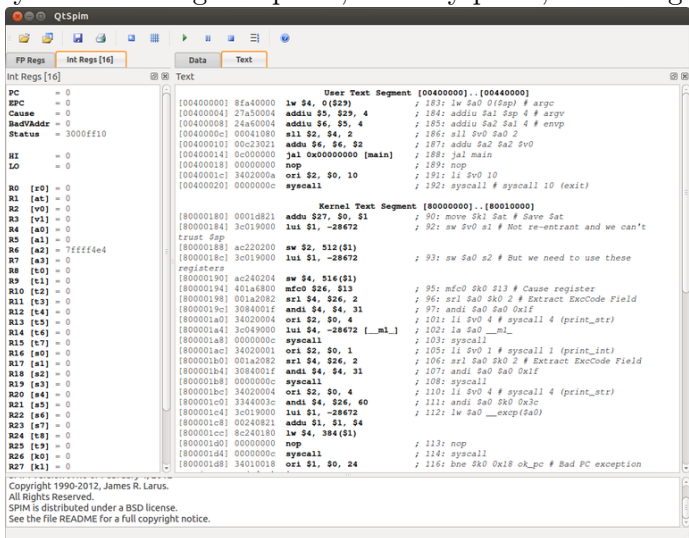
Running a MIPS Code

- Ideally one should execute on a MIPS hardware
- We will be using a free simulator tool: **SPIM**¹
- Name of the simulator is a reversal of the letters ‘MIPS’

¹more specifically QtSPIM: <http://spimsimulator.sourceforge.net/>

Getting started with QtSPIM

3 primary sections: Register panel, Memory panel, & Messages panel.



The screenshot shows the QtSPIM application window. On the left is the Register panel, and on the right is the Text panel.

Register Panel (Int Regs [16]):

```
PC = 0
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 0
R5 [a1] = 0
R6 [a2] = 7ffff4e4
R7 [a3] = 0
R8 [t0] = 0
R9 [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [a0] = 0
R17 [a1] = 0
R18 [a2] = 0
R19 [a3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

Text Panel:

```
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29)           ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4          ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4          ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2           ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2         ; 187: addu $a2 $a2 $v0
[00400014] 0c000000 jal 0x00000000 [main]      ; 188: jal main
[00400018] 00000000 nop                    ; 189: nop
[0040001c] 3402000a ori $2, $0, 10         ; 191: li $v0 10
[00400020] 0000000c syscall                ; 192: syscall # syscall 10 (exit)

Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821 addu $27, $0, $1          ; 90: move $k1 $at # Save $at
[80000184] 3c019000 lui $1, -28672         ; 92: sw $v0 $1 # Not re-entrant and we can't
                        trust $sp
[80000188] ac220200 sw $2, 512($1)
[8000018c] 3c019000 lui $1, -28672         ; 93: sw $a0 $2 # But we need to use these
                        registers
[80000190] ac240204 sw $4, 516($1)
[80000194] 401a6800 mfc0 $26, $13          ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082 srl $4, $26, 2         ; 96: srl $a0 $k0 2 # Extract ExecCode Field
[8000019c] 3084001f andi $4, $4, 31         ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004 ori $2, $0, 4           ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000 lui $4, -28672 [__ml_]  ; 102: la $a0 __ml_
[800001a8] 0000000c syscall                ; 103: syscall
[800001ac] 34020001 ori $2, $0, 1           ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082 srl $4, $26, 2         ; 106: srl $a0 $k0 2 # Extract ExecCode Field
[800001b4] 3084001f andi $4, $4, 31         ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c syscall                ; 108: syscall
[800001bc] 34020004 ori $2, $0, 4           ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c andi $4, $26, 60         ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000 lui $1, -28672         ; 112: lw $a0 __exc($a0)
[800001c8] 00240821 addu $1, $1, $4
[800001cc] 8c240180 lw $4, 384($1)
[800001d0] 00000000 nop                    ; 113: nop
[800001d4] 0000000c syscall                ; 114: syscall
[800001d8] 34010018 ori $1, $0, 24         ; 116: hne $k0 0x18 ok_pc # Bad PC exception
```

Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the File README for a full copyright notice.

Getting started with QtSPIM contd.

Text tab in Memory panel shows the Program memory contents
Data tab shows the contents of the Data memory space

```
Data
User data segment [10000000]..[10040000]
[10000000]..[1003ffff] 00000000

User Stack [7ffff4de]..[80000000]
[7ffff4dc] 00000000
[7ffff4e0] 00000000 7fffffff 7fffffff 7fffffff
[7ffff4f0] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff500] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff510] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff520] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff530] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff540] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff550] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff560] 7fffffff 7fffffff 7fffffff 7fffffff
[7ffff570] 7fffffff 7fffffff 00000000 3d5f0000
[7ffff580] 7273752f 6e69622f 7374712f 006d6970
[7ffff590] 50444c4f 2f3d4457 656d6e68 68736a2f
[7ffff5a0] 72656661 7469622f 6b637562 672f7465
[7ffff5b0] 69646172 325676e6 5f323130 60606166
[7ffff5c0] 7063655f 30373165 55415800 524f4854
[7ffff5d0] 3d595449 6d6f682f 736a2f65 65666168
[7ffff5e0] 58202e72 68747561 7469726e 4e430079
[7ffff5f0] 54524f4c 3d4d5245 6d6f6e67 65742d65
[7ffff600] 6e696d72 4e006c61 43535345 45534f4c
[7ffff610] 73752f3d 69622e72 65602f6e 69707373
[7ffff620] 25206570 73252073 47445800 5255435f
[7ffff630] 544e4552 5345445f 504f544b 696e553d
[7ffff640] 44007974 4e505349 3a3d5941 454c0030
[7ffff650] 504f5353 7e3d4e45 73752f20 69622e72
[7ffff660] 65602f6e 69707373 25206570 42440073
[7ffff670] 535f5355 49535345 425f4e4f 415f5355
[7ffff680] 45524444 753d5353 3a78696e 74736261
[7ffff690] 74636172 6d742f3d 62642e70 672d7375
[7ffff6a0] 53796255 6d364839 75672c63 373d6469
[7ffff6b0] 37393630 38666539 35323230 38303334
[7ffff6c0] 62333139 30656463 30303030 00313330
[7ffff6d0] 6e696d72 4e006c61 43535345 45534f4c
[7ffff6e0] 73752f3d 69622e72 65602f6e 69707373
[7ffff6f0] 25206570 73252073 47445800 5255435f
[7ffff700] 544e4552 5345445f 504f544b 696e553d
[7ffff710] 44007974 4e505349 3a3d5941 454c0030
[7ffff720] 504f5353 7e3d4e45 73752f20 69622e72
[7ffff730] 65602f6e 69707373 25206570 42440073
[7ffff740] 535f5355 49535345 425f4e4f 415f5355
[7ffff750] 45524444 753d5353 3a78696e 74736261
[7ffff760] 74636172 6d742f3d 62642e70 672d7375
[7ffff770] 53796255 6d364839 75672c63 373d6469
[7ffff780] 37393630 38666539 35323230 38303334
[7ffff790] 62333139 30656463 30303030 00313330
[7ffff7a0] 6e696d72 4e006c61 43535345 45534f4c
[7ffff7b0] 73752f3d 69622e72 65602f6e 69707373
[7ffff7c0] 25206570 73252073 47445800 5255435f
[7ffff7d0] 544e4552 5345445f 504f544b 696e553d
[7ffff7e0] 44007974 4e505349 3a3d5941 454c0030
[7ffff7f0] 504f5353 7e3d4e45 73752f20 69622e72
[7ffff800] 65602f6e 69707373 25206570 42440073
[7ffff810] 535f5355 49535345 425f4e4f 415f5355
[7ffff820] 45524444 753d5353 3a78696e 74736261
[7ffff830] 74636172 6d742f3d 62642e70 672d7375
[7ffff840] 53796255 6d364839 75672c63 373d6469
[7ffff850] 37393630 38666539 35323230 38303334
[7ffff860] 62333139 30656463 30303030 00313330
[7ffff870] 6e696d72 4e006c61 43535345 45534f4c
[7ffff880] 73752f3d 69622e72 65602f6e 69707373
[7ffff890] 25206570 73252073 47445800 5255435f
[7ffff8a0] 544e4552 5345445f 504f544b 696e553d
[7ffff8b0] 44007974 4e505349 3a3d5941 454c0030
[7ffff8c0] 504f5353 7e3d4e45 73752f20 69622e72
[7ffff8d0] 65602f6e 69707373 25206570 42440073
[7ffff8e0] 535f5355 49535345 425f4e4f 415f5355
[7ffff8f0] 45524444 753d5353 3a78696e 74736261
[7ffff900] 74636172 6d742f3d 62642e70 672d7375
[7ffff910] 53796255 6d364839 75672c63 373d6469
[7ffff920] 37393630 38666539 35323230 38303334
[7ffff930] 62333139 30656463 30303030 00313330
[7ffff940] 6e696d72 4e006c61 43535345 45534f4c
[7ffff950] 73752f3d 69622e72 65602f6e 69707373
[7ffff960] 25206570 73252073 47445800 5255435f
[7ffff970] 544e4552 5345445f 504f544b 696e553d
[7ffff980] 44007974 4e505349 3a3d5941 454c0030
[7ffff990] 504f5353 7e3d4e45 73752f20 69622e72
[7ffff9a0] 65602f6e 69707373 25206570 42440073
[7ffff9b0] 535f5355 49535345 425f4e4f 415f5355
[7ffff9c0] 45524444 753d5353 3a78696e 74736261
[7ffff9d0] 74636172 6d742f3d 62642e70 672d7375
[7ffff9e0] 53796255 6d364839 75672c63 373d6469
[7ffff9f0] 37393630 38666539 35323230 38303334
[7ffff000] 62333139 30656463 30303030 00313330
```

QtSPIM Demo

- Save your MIPS code with `.s` or `.asm` extension
- Load your code in QtSPIM via 'Reinitialize and Load File' option
- Click on the play button to run your code

Using QtSPIM Console

Consider the following C code fragment

```
int x = 10;  
int y = 20;  
printf("%d", x + y); // prints a integer
```

Using QtSPIM Console contd.

```
.globl main
.text

main:
    lw $t0, x
    lw $t1, y
    add $t3, $t0, $t1

    li $v0, 1 # print_int syscall
    move $a0, $t3
    syscall

    li $v0, 10 # exit syscall
    syscall

.data
x: .word 10
y: .word 20
```

¹SPIM syscalls: <https://www.doc.ic.ac.uk/lab/secondyear/spim/node8.html>

Array and Loops

Consider the following C code fragment

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // length of arr
int sum = 0;
int i = 0;
while (i != n) {
    sum = sum + arr[i];
    i = i + 1;
}
printf("%d", sum);
```

Array and Loops contd.

Terminating condition rewritten

Array indexing replaced by pointer operation

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // lenght of arr
int sum = 0;
int i = 0;
while (n != 0) {
    sum = sum + *(arr + i); // pointer arithmetic
    i = i + 1;
    n = n - 1;
}
printf("%d", sum);
```

Array and Loops contd.

The while loop is converted to do...while assuming $n > 0$

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // length of arr
int sum = 0;
int i = 0;
do {
    sum = sum + *(arr + i); // pointer arithmetic
    i = i + 1;
    n = n - 1;
} while (n != 0); // assume n > 0
printf("%d", sum);
```

Array and Loops contd.

Utility of the index variable `i` is substituted with pointer shifting

```
int arr[] = {1, 5, 8, 10, 3};
int n = 5; // length of arr
int sum = 0;
int *p = arr; // base address
do {
    sum = sum + *p;
    p = p + 1; // pointer arithmetic
    n = n - 1;
} while (n != 0); // assume n > 0
printf("%d", sum);
```

Array and Loops contd.

```
.globl main
.data
arr: .word 1, 5, 8, 10, 3
n:   .word 5
.text
main:
    la $t0, arr # p
    lw $t1, n
    li $t2, 0 # sum
loop:
    lw $t4, 0($t0) # *p
    add $t2, $t2, $t4 # sum = sum + *p
    addi $t0, $t0, 4 # incrementing p, integers are 4 byte long
    addi $t1, $t1, -1 # n = n - 1
    bne $t1, $0, loop

    li $v0, 1 # print_int syscall
    move $a0, $t2 # copy sum
    syscall
```

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
```

```
x = 1 << 20
```


Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code
```

```
x = 1 << 20
```

```
# using MIPS
```

```
li $t0, 1 # load 1
```

```
sll $t0, $t0, 20
```

```
# shift left by 20 places
```

Scaling by 2^k Efficiently

- Computing 2^{20}

// C code

`x = 1 << 20`

using MIPS

`li $t0, 1 # load 1`

`sll $t0, $t0, 20`

shift left by 20 places

- computing $n \times 2^{10}$

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code  
x = 1 << 20
```

```
# using MIPS  
li $t0, 1 # load 1  
sll $t0, $t0, 20  
# shift left by 20 places
```

- computing $n \times 2^{10}$

```
// C code  
x = n << 10
```

```
# using MIPS  
# assume $t0 contains n  
sll $t1, $t0, 10
```

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code  
x = 1 << 20
```

```
# using MIPS  
li $t0, 1 # load 1  
sll $t0, $t0, 20  
# shift left by 20 places
```

- computing $n \times 2^{10}$

```
// C code  
x = n << 10
```

```
# using MIPS  
# assume $t0 contains n  
sll $t1, $t0, 10
```

- computing $\lfloor n/2^4 \rfloor$

Scaling by 2^k Efficiently

- Computing 2^{20}

```
// C code  
x = 1 << 20
```

```
# using MIPS  
li $t0, 1 # load 1  
sll $t0, $t0, 20  
# shift left by 20 places
```

- computing $n \times 2^{10}$

```
// C code  
x = n << 10
```

```
# using MIPS  
# assume $t0 contains n  
sll $t1, $t0, 10
```

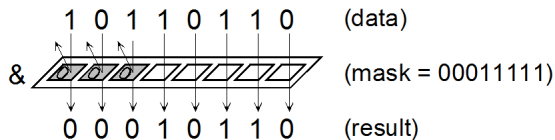
- computing $\lfloor n/2^4 \rfloor$

```
// C code  
x = n >> 4
```

```
# using MIPS  
# assume $t0 contains n  
srl $t1, $t0, 4
```

Bitwise Operation and Masking

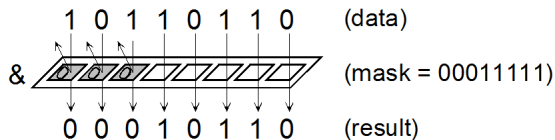
- Bitwise and operation



¹image src: <https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html>

Bitwise Operation and Masking

- Bitwise and operation

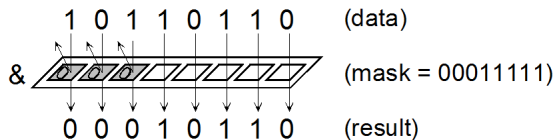


Getting i-th bit:

¹image src: <https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html>

Bitwise Operation and Masking

- Bitwise and operation

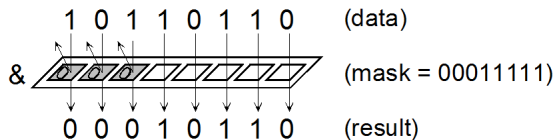


Getting i-th bit: $x \& (1 \ll i)$

¹image src: <https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html>

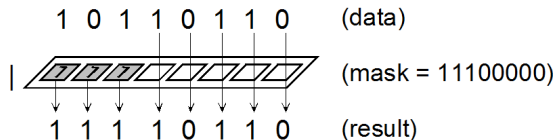
Bitwise Operation and Masking

- Bitwise and operation



Getting i-th bit: $x \& (1 \ll i)$

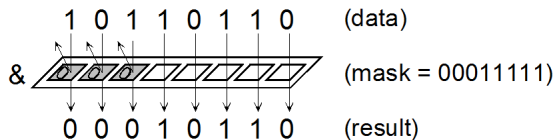
- Bitwise or operation



¹image src: <https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html>

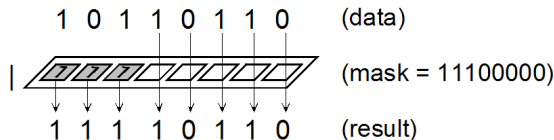
Bitwise Operation and Masking

- Bitwise and operation



Getting i -th bit: $x \& (1 \ll i)$

- Bitwise or operation

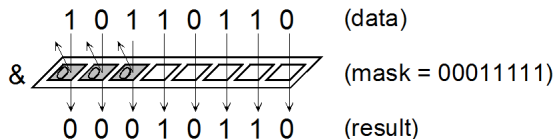


Setting i -th bit:

¹image src: <https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html>

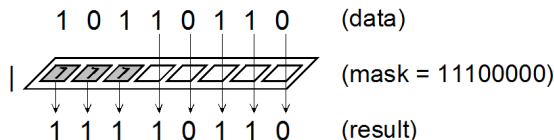
Bitwise Operation and Masking

- Bitwise and operation



Getting i -th bit: $x \& (1 \ll i)$

- Bitwise or operation



Setting i -th bit: $x | (1 \ll i)$

- Ex. What happens with $n \& (n - 1)$?

¹image src: <https://icarus.cs.weber.edu/~dab/cs1410/textbook/2.Core/bitops.html>

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise xor operation
Flipping i-th bit:

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise xor operation
Flipping i-th bit: $x \wedge (1 \ll i)$

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise xor operation
Flipping i-th bit: $x \wedge (1 \ll i)$
- Ex. What is output of: $n \wedge 0xAAAAAAAA?$

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise xor operation
Flipping i-th bit: $x \wedge (1 \ll i)$
- Ex. What is output of: $n \wedge 0xAAAAAAAAA?$
- Ex. What is output of: $n \wedge 0x55555555?$

Bitwise Operation and Masking contd.

- The xor operation

EX-OR (X-OR) Gate Truth Table

Inputs		Output $X = A \oplus B$
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Notice that: $X \oplus 0 = X$ and $X \oplus 1 = \bar{X}$

- Bitwise xor operation

Flipping i-th bit: $x \wedge (1 \ll i)$

- Ex. What is output of: $n \wedge 0xAAAAAAAA?$
- Ex. What is output of: $n \wedge 0x55555555?$
- Ex. What is output of: $n \wedge 0xFFFFFFFF?$

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything

¹image src: https://en.wikipedia.org/wiki/XOR_swap_algorithm

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything: $X \oplus X = 0$

¹image src: https://en.wikipedia.org/wiki/XOR_swap_algorithm

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything: $X \oplus X = 0$

- Swapping values of two variables

¹image src: https://en.wikipedia.org/wiki/XOR_swap_algorithm

Bitwise Operation and Masking contd.

- Getting NOTHING out of anything: $X \oplus X = 0$
- Swapping values of two variables

<i>Operation</i>	<i>Meaning</i>
$a = a \oplus b$	$a = A \oplus B$
$b = b \oplus a$	$b = B \oplus (A \oplus B) = A$
$a = a \oplus b$	$a = (A \oplus B) \oplus A = B$

x	y	
1010	0011	= 1001 → x
1001	0011	= 1010 → y
1001	1010	= 0011 → x
0011	1010	

¹image src: https://en.wikipedia.org/wiki/XOR_swap_algorithm

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Question 3

Load a constant value without specifying any constant explicitly.

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Question 3

Load a constant value without specifying any constant explicitly.

Question 4

Count the number of 1s in an integer.

Assignment 3 (informally)

Question 2

Devise an efficient way to obtain 1's complement of an integer. You are restricted from specifying any constant explicitly (cannot do $X \oplus -1$).

Question 3

Load a constant value without specifying any constant explicitly.

Question 4

Count the number of 1s in an integer.

Question 5

Suppose there are n distinct integers all in the closed interval of $[0, n]$, that is only one number is absent, and all others occur exactly once. Your task is to find the missing number efficiently.