CSCI2467: Systems Programming Concepts

Midterm review

Course Instructors:

Matthew Toups
Caitlin Boyce

Course Assistants:

Saroj Duwal
David McDonald

Spring 2020

THE UNIVERSITY of NEW ORLEANS
DEPARTMENT OF COMPUTER SCIENCE

Scores so far

- Your updated datalab scores to date are posted in AutoLab (see gradebook)
- Many folks lost points on datalab due to insufficient explanations in comments

AutoLab gradebook examples

AUTOLAB Gradebook

CSCI 2467: Systems Programming Concepts (2019-Spring)

Grades for

Please see the course syllabus for information on grading policies.

LAB ASSIGNMENTS	GDU	PLD	Final Score
Intro Lab	1	0	40.0/40.0
Data Lab	0	1	30.0/40.0
Bomb Lab	0	0	
Category Average			35.0

AutoLab gradebook examples

AUTOLAB Gradebook

CSCI 2467: Systems Programming Concepts (2019-Spring)

Grades for

Please see the course syllabus for information on grading policies.

LAB ASSIGNMENTS	GDU	PLD	Final Score
Intro Lab	0	0	33.0/40.0
Data Lab	1	1	26.0/40.0
Bomb Lab	0	0	
Category Average			**29.5**

Upcoming exam

- Midterm exam: Wednesday February 19
- Location: Math 118
  - Given during class time, 50 minutes, closed book etc
  - We will provide a sheet of notes and helpful values, you will only bring something to write with.

Exam material overview

- Chapter 2 (2.1-2.4)
  - see practice problems (in later slides)
  - see activity handouts
  - be able to explain datalab solutions
- Chapter 3 (3.1-3.6)
  - will present problems in Intel assembly format
  - see practice problems (in later slides)
  - see activity handout problems
  - be able to identify things you have seen in bomblab

Exam rough breakdown

Roughly...

- 45% bitwise operations, binary two's complement representations
- 20% datalab puzzle comment/explanation
- 35% bomblab, reverse engineering, reading assembly, filling in blanks in C code

- Class updates
1. Midterm exam info
2. Midterm format
3. Chapter 2: Data
   - operators
   - int
   - Conversion between signed & unsigned
4. Chapter 3: Machine-level programs
   - Arithmetic
   - Control
   - Activities
5. Puzzles to practice on

Midterm format

- Fill in a table with missing values
  - small examples (less than 32-bit)
  - using binary two's complement representation of ints
  - using binary operations, masking, arithmetic (w/ overflow)
  - is an expression true for all values?
- Datalab:
  - Given some C code, fill in a blank or table of values
  - Given a C function, fill in comments explaining how it is solved
- Bomblab:
  - Fill in missing C source code (based on given disassembly)
  - Given multiple C functions, circle the correct one

Boolean Algebra

Algebraic representation of logic, developed by Boole in 1850s
Encodes "True" as 1 and "False" as 0

Binary AND:
$A\&B = 1$ when
both $A = 1$ and $B = 1$

&	0	1
0	0	0
1	0	1

Binary OR:
$A|B = 1$ when
either $A = 1$ or $B = 1$

|	0	1
0	0	1
1	1	1

Binary NOT (complement):
$\sim A = 1$ when $A = 0$

~	1
0	1
1	0

Exclusive-Or (XOR):
$A \wedge B = 1$ when *either* $A = 1$ or $B = 1$ but *not* both

∧	0	1
0	0	1
1	1	0

Based on **Figure 2.7** in CS:APP3e

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
  - View 0 as "False"
  - View anything non-zero as "True"
  - Always return 0 or 1
  - Early termination!

Examples:

- $!0x41 \Rightarrow 0x00$

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
  - View 0 as "False"
  - View anything non-zero as "True"
  - Always return 0 or 1
  - Early termination!

Examples:

- $!0x41 \Rightarrow 0x00$
- $!0x00 \Rightarrow 0x01$

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
  - View 0 as "False"
  - View anything non-zero as "True"
  - Always return 0 or 1
  - Early termination!

Examples:

- !0x41 ⇒ 0x00
- !0x00 ⇒ 0x01
- !!0x41 ⇒ 0x01

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
  - View 0 as "False"
  - View anything non-zero as "True"
  - Always return 0 or 1
  - Early termination!

Examples:

- !0x41 ⇒ 0x00
- !0x00 ⇒ 0x01
- !!0x41 ⇒ 0x01
- 0x69 && 0x55 ⇒ 0x01

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
- View 0 as "False"
- View anything non-zero as "True"
- Always return 0 or 1
- Early termination!

Examples:

- !0x41 ⇒ 0x00
- !0x00 ⇒ 0x01
- !!0x41 ⇒ 0x01
- 0x69 && 0x55 ⇒ 0x01
- 0x69 || 0x55 ⇒ 0x01

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
  - View 0 as "False"
  - View anything non-zero as "True"
  - Always return 0 or 1
  - Early termination!

Examples:

- !0x41 $\Rightarrow$ 0x00
- !0x00 $\Rightarrow$ 0x01
- !!0x41 $\Rightarrow$ 0x01
- 0x69 && 0x55 $\Rightarrow$ 0x01
- 0x69 || 0x55 $\Rightarrow$ 0x01
- a && 5/a (will never divide by zero)

Logical operators

Don't confuse bitwise and logical operators! They look similar but are very different.

- &&, || , !
- View 0 as "False"
- View anything non-zero as "True"
- Always return 0 or 1
- Early termination!

Examples:

- !0x41 ⇒ 0x00
- !0x00 ⇒ 0x01
- !!0x41 ⇒ 0x01
- 0x69 && 0x55 ⇒ 0x01
- 0x69 || 0x55 ⇒ 0x01
- a && 5/a (will never divide by zero)
- p && *p (avoids null pointer access)

Shift operators

- Left Shift: $x << y$
  - Shift bitvector x left y positions (Throw away extra bits on left)
  - Fill with 0s on right
- Right Shift: $x >> y$
  - Shift bitvector x right y positions (Throw away extra bits on right)
  - ⋆ Logical shift: fill with 0s on left
  - ⋆ Arithmetic shift: Replicate most significant bit on left
- Undefined: Shift < 0 or $\geq$ word size

Examples

Argument x	01100010
$<< 3$	00010000
Log. $>> 2$	00011000
Arith. $>> 2$	00011000

Argument x	10100010
$<< 3$	00010000
Log. $>> 2$	00101000
Arith. $>> 2$	11101000

two's complement encoding example

```
short int x=   2467:  00001001 10100011
short int y=  -2467:  11110110 01011101
```

Weight	2467		-2467	
1	1	1	1	1
2	1	2	0	0
4	0	0	1	4
8	0	0	1	8
16	0	0	1	16
32	1	32	0	0
64	0	0	1	64
128	1	128	0	0
256	1	256	0	0
512	0	0	1	512
1024	0	0	1	1024
2048	1	2048	0	0
4096	0	0	1	4096
8192	0	0	1	8192
16384	0	0	1	16384
-32768	0	0	1	-32768
Sum:		2467		-2467

Numeric Ranges

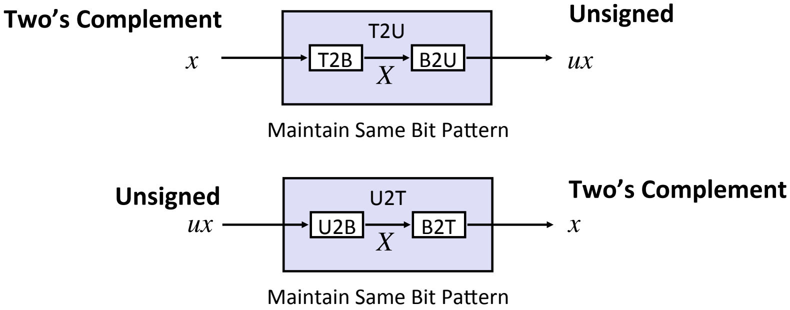
- Unsigned values
  - UMin $= 0$
    000 … 0
  - UMax $= 2^w - 1$
    111 … 1

- Two's complement values
  - TMin $= -2^{w-1}$
    100…0
  - TMax $= 2^{w-1} - 1$
    011…1

Values for $w = 16$ (`short int`)

	Decimal	Hex	Binary
UMax	65535	FF FF	11111111 11111111
TMax	32767	7F FF	01111111 11111111
TMin	-32768	80 00	10000000 00000000
-1	-1	FF FF	11111111 11111111
0	0	00 00	00000000 00000000

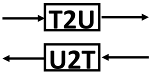
Mapping Between Signed & Unsigned



- **Mappings between unsigned and two's complement numbers:**
  Keep bit representations and reinterpret

Mapping Signed ↔ Unsigned

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

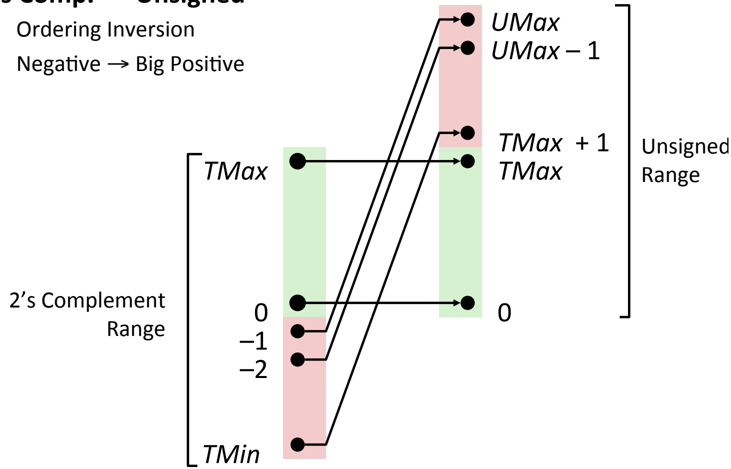


Mapping Signed ↔ Unsigned

Bits	Signed	Unsigned
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15

Conversion Visualized

- **2's Comp. → Unsigned**
  - Ordering Inversion
  - Negative → Big Positive



Practice

THE UNIVERSITY *of* NEW ORLEANS
DEPARTMENT OF COMPUTER SCIENCE

CSCI 2467, Fall 2020

Class Activity: Two's complement, bitwise and logical operators

Wednesday, January 29, 2020

2467 Instructors: M. Toups, C. Boyce
staff@cs.uno.edu

1 Introduction

In this activity you will practice working with binary numbers, bitwise operators, and logical operators. This activity is based in part on material developed by Professor Saturnino Garcia of the University of San Diego and is used with permission.

2 Review of Negative Integers

More practice

We strongly recommend you check out these practice problems from the text! Textbook problems:

- 2.1 through 2.4
- 2.6 through 2.9
- 2.12 through 2.16

Arithemetic operations

Two operand instructions

- Pay attention to order of operands
- No distinction between signed & unsigned. (why not?)

Format	Operands	Computation	
`add`	dest,src	dest = dest + src	
`sub`	dest,src	dest = dest - src	
`imul`	dest,src	dest = dest * src	
`sal`	dest,src	dest = dest << src	(also `shl`)
`sar`	dest,src	dest = dest >> src	(arithmetic)
`shr`	dest,src	dest = dest >> src	(logical)
`xor`	dest,src	dest = dest ^ src	
`and`	dest,src	dest = dest & src	
`or`	dest,src	dest = dest — src	

Arithemetic operations

One operand instructions

Format	Operand	Computation
`inc`	dest	dest = dest + 1
`dec`	dest	dest = dest - 1
`neg`	dest	dest = – dest
`not`	dest	dest = ˜dest

- See CSAPP3e for more on these operations.

Arithmetic expression example

```
long arith
(long x,long y,long z)
{
   long t1 = x+y;
   long t2 = z+t1;
   long t3 = x+4;
   long t4 = y * 48;
   long t5 = t3 + t4;
   long rval = t2 * t5;
   return rval;
}
```

```
arith:
 lea    rax, [rdi+rsi]
 add    rax, rdx
 lea    rcx, [rsi+rsi*2]
 sal    rcx, 4
 lea    rcx, [rdi+4+rcx]
 imul   rax, rcx
 ret
```

Interesting instructions:

- lea: address computation
- sal: shift left
- imul: multiplication
  (only used once!)

Arithmetic expression example

```
long arith
(long x,long y,long z)
{
  long t1 = x+y;
  long t2 = z+t1;
  long t3 = x+4;
  long t4 = y * 48;
  long t5 = t3 + t4;
  long rval = t2 * t5;
  return rval;
}
```

```
arith:
 lea   rax, [rdi+rsi]    # t1
 add   rax, rdx          # t2
 lea   rcx, [rsi+rsi*2]
 sal   rcx, 4            # t4
 lea   rcx, [rdi+4+rcx]# t5
 imul  rax, rcx         # rval
 ret
```

Register	Use
rdi	argument x
rsi	argument y
rdx	argument z
rax	t1, t2, rval
rcx	t4, t5

Conditional jumps

jX Instructions

- Jump to different part of code depending on condition codes

jX	Condition	Description
`jmp`	`1`	Unconditional
`je`	`ZF`	Equal / Zero
`jne`	`~ZF`	Not Equal / Not Zero
`js`	`SF`	Negative
`jns`	`~SF`	Nonnegative
`jg`	`~(SF^OF)&~ZF`	Greater (Signed)
`jge`	`~(SF^OF)`	Greater or Equal (Signed)
`jl`	`(SF^OF)`	Less (Signed)
`jle`	`(SF^OF)|ZF`	Less or Equal (Signed)
`ja`	`~CF&~ZF`	Above (unsigned)
`jb`	`CF`	Below (unsigned)

Conditional branch example

```c
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```asm
absdiff:
   cmp     rdi, rsi
   jle     .L2
   mov     rax, rdi
   sub     rax, rsi
   ret
.L2:    # x <= y
   mov     rax, rsi
   sub     rax, rdi
   ret
```

Compiled with:
gcc −Og −S absdiff.c −masm=intel

Register	Use
rdi	argument x
rsi	argument y
rax	return value

THE UNIVERSITY *of* NEW ORLEANS

DEPARTMENT OF COMPUTER SCIENCE

CSCI 2467, Spring 2020

Class Activity: Understanding disassembled code

Friday, February 7

CSCI 2467 Staff: staff@2467.cs.uno.edu

1 Introduction

In this activity you will get practice reading assembly language code which has been *disassembled* – taken from an existing, compiled program. This "reverse-engineering" technique is especially useful for folks in the computer security field who are studying malware and software vulnerabilities. It is also an excellent way for anyone to gain a deeper understanding of how their programs are actually compiled and executed. (The questions are from CS:APP3e by Bryant and O'Hallaron, chapter 3.)

Below is a table which should be helpful. This is the "calling convention" for `x86-64` on Linux,

3. Consider the following assembly code:

```
fun1:
        cmp     rdi, rsi
        jge     .L3
        mov     rax, rdi
        ret
.L3:
        mov     rax, rsi
        ret
```

What C function could have been compiled to generate these instructions? (There is more than one correct answer.)

Fill in the three blanks below with valid C code (using variable names `a` and `b`):

```
long fun1(long a, long b) {

    if ( ______________________________ )

            return ________________ ;

    else
```

More practice

CS:APP3e textbook practice problems:

- 3.18
- 3.21
- 3.24
- 3.26
- 3.28

Get to work!

- We have the rest of the class to work on:
  - midterm prep: ask questions
  - review textbook problems
  - review activities
  - review datalab/bomblab

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If... true for all values, or false?

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$

C `int` Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x \& 7 == 7$	$\Rightarrow (x << 30) < 0$

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x\&7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$

C `int` Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x \& 7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$

C `int` Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x\&7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x \& 7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0 \; \&\& \; y > 0$	$\Rightarrow x + y > 0$

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x \& 7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0 \ \&\& \ y > 0$	$\Rightarrow x + y > 0$
$x \geq 0$	$\Rightarrow -x \leq 0$

C `int` Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x\&7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0 \&\& y > 0$	$\Rightarrow x + y > 0$
$x \geq 0$	$\Rightarrow -x \leq 0$
$x \leq 0$	$\Rightarrow -x \geq 0$

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x\&7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0 \;\&\&\; y > 0$	$\Rightarrow x + y > 0$
$x \geq 0$	$\Rightarrow -x \leq 0$
$x \leq 0$	$\Rightarrow -x \geq 0$
	$(x \mid -x) >> 31 == -1$

C int Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x \& 7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0 \;\&\&\; y > 0$	$\Rightarrow x + y > 0$
$x \geq 0$	$\Rightarrow -x \leq 0$
$x \leq 0$	$\Rightarrow -x \geq 0$
	$(x \mid -x) >> 31 == -1$
	$ux >> 3 == ux/8$

C `int` Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x\&7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0\ \&\&\ y > 0$	$\Rightarrow x + y > 0$
$x \geq 0$	$\Rightarrow -x \leq 0$
$x \leq 0$	$\Rightarrow -x \geq 0$
	$(x \mid -x) >> 31 == -1$
	$ux >> 3 == ux/8$
	$x >> 3 == x/8$

C `int` Puzzles!

```
int x = foo();
int y = bar();
unsigned ux = x;
unsigned uy = y;
```

If...	true for all values, or false?
$x < 0$	$\Rightarrow (x * 2) < 0$
	$ux \geq 0$
$x\&7 == 7$	$\Rightarrow (x << 30) < 0$
	$ux > -1$
$x > y$	$\Rightarrow -x < -y$
	$x * x \geq 0$
$x > 0 \ \&\& \ y > 0$	$\Rightarrow x + y > 0$
$x \geq 0$	$\Rightarrow -x \leq 0$
$x \leq 0$	$\Rightarrow -x \geq 0$
	$(x \mid -x) >> 31 == -1$
	$ux >> 3 == ux/8$
	$x >> 3 == x/8$
	$x \ \& \ (x-1) \ \ != \ 0$