CMPT 295

Lecture 16 – Midterm 1 Review Session

Go over Rounding - Lecture 6 Slide 13:

Rounding

1. Round up
2. Round down
3. When half way -> When bits to right of rounding position are **100…0**$_2$
   - Round to even number: produces **0** as the least significant bit of rounded result

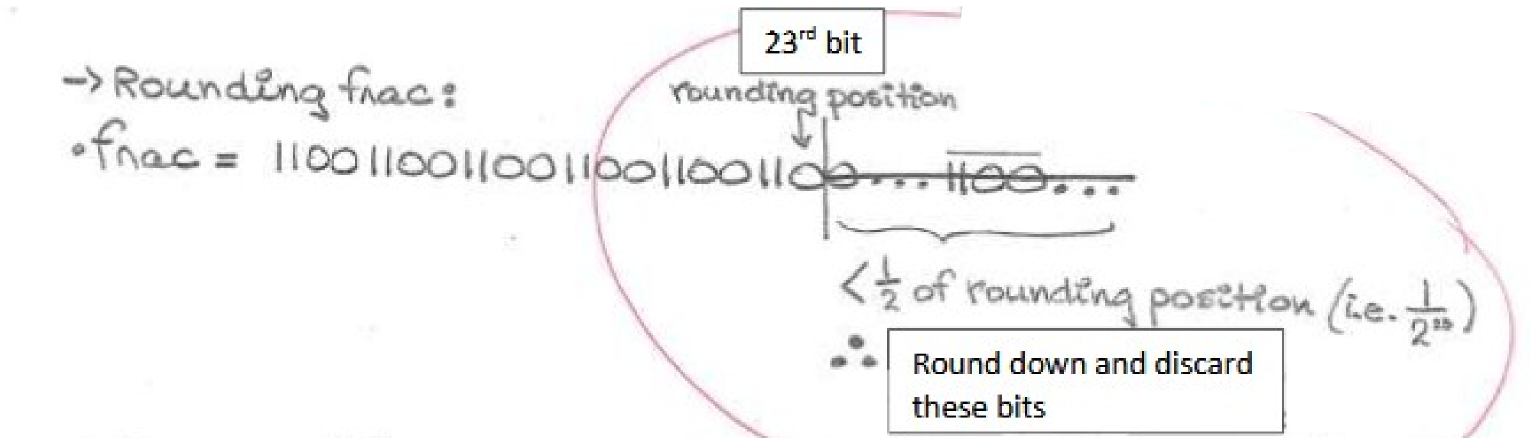
- Example: Round to nearest 1/4 (2 bits right of binary point)

Value	Binary	Rounded	Action	Rounded Value
2 3/32	10.00011_2	10.00_2	(<1/2—down)	2
2 3/16	10.00110_2	10.01_2	(>1/2—up)	2 1/4
2 7/8	10.11100_2	11.00_2	(1/2—up to even)	3
2 5/8	10.10100_2	10.10_2	(1/2—down to even)	2 1/2

24th bit

imagine this is 23th bit of frac of IEEE

Assignment 3 Question 1 a. iii



frac: 1100 1100 1100 1100 1100 110 01100 …

Assignment 3 Question 1 a. iv

- frac = 0101010101010101010101ϕ | ~~101...~~

  $> \frac{1}{2}$ of rounding position

  ∴ rounding up & discard these bits

frac: 0101 0101 0101 0101 0101 010 $\overline{101}$ …

Assignment#2 Question 2 g., h., i., k.

		Exponent			Fraction		Value		
Description	**Bit representation**	**exp**	**E**	**2^E**	**frac**	**M**	**M 2^E**	**V**	**Decimal**
zero	0 000 00	0	-2	1/4	0/4	0/4	0/16	0	0.0
Smallest positive denormalized	0 000 01	0	-2	1/4	¼	¼	1/16	1/16	0.0625
	0 000 10	0	-2	1/4	2/4 = ½	2/4 = ½	2/16	2/16	0.125
Largest positive denormalized	**0 000 11**	**0**	**-2**	**1/4**	**¾**	**¾**	**3/16**	**3/16**	**0.1875**
Smallest positive normalized	**0 001 00**	**1**	**-2**	**1/4**	**0/4**	**4/4 = 1**	**4/16**	**4/16**	**0.25**
	0 001 01	1	-2	1/4	¼	5/4	5/16	5/16	0.3125
	0 001 10	1	-2	1/4	2/4 = ½	6/4	6/16	6/16	0.375
	0 001 11	1	-2	1/4	¾	7/4	7/16	7/16	0.4375
	0 010 00	2	-1	1/2	0/4	4/4 = 1	4/8	4/8	0.5
	0 010 01	2	-1	1/2	¼	5/4	5/8	5/8	0.625
	0 010 10	2	-1	1/2	2/4 = ½	6/4	6/8	6/8	0.75
	0 010 11	2	-1	1/2	¾	7/4	7/8	7/8	0.875
One	0 011 00	3	0	1	0/4	4/4 = 1	4/4	4/4	1.0

	0 011 01	3	0	1	¼	5/4	5/4	5/4	1.25
	0 011 10	3	0	1	2/4 = ½	6/4	6/4	6/4	1.5
	0 011 11	3	0	1	¾	7/4	7/4	7/4	1.75
	0 100 00	4	1	2	0/4	4/4 = 1	8/4	8/4	2
	0 100 01	4	1	2	¼	5/4	10/4	10/4	2.5
	0 100 10	4	1	2	2/4 = ½	6/4	12/4	12/4	3
	0 100 11	4	1	2	¾	7/4	14/4	14/4	3.5
	0 101 00	5	2	4	0/4	4/4 = 1	16/4	16/4	4
	0 101 01	5	2	4	¼	5/4	20/4	20/4	5
	0 101 10	5	2	4	2/4 = ½	6/4	24/4	24/4	6
	0 101 11	5	2	4	¾	7/4	28/4	28/4	7
	0 110 00	6	3	8	0/4	4/4 = 1	32/4	32/4	8
	0 110 01	6	3	8	¼	5/4	40/4	40/4	10
	0 110 10	6	3	8	2/4 = ½	6/4	48/4	48/4	12
Largest positive normalized	0 110 11	6	3	8	¾	7/4	56/4	56/4	14
+ Infinity	0 111 00	–	–	–	–	–	–	∞	–
NaN		–	–	–	–	–	–	NaN	–

g. What is the "range" (not contiguous) of fractional decimal numbers that can be represented using this 6-bit floating-point representation?

"range" of real numbers → [-14.0 .. 14.0] not considering ±∞ and NaN
(since it is not a continuous range)

h. What is the range of the normalized exponent $\mathbf{E}$ ($\mathbf{E}$ found in the equation $v = (-1)^s M 2^E$) which can be represented by this 6-bit floating-point representation?

Range of the normalized exponent $\mathbf{E}$

range of E → [-2 .. 3]

⇓

exp -> [001 .. 110]

denormalized exponent $\mathbf{E}$

exp = 000 → E = -2
001 → E = -2
010 → E = -1
011 → E = 0
100 → E = 1
101 → E = 2
110 → E = 3
111 → ±∞, NaN

smooth transition (000 and 001)

Range of the normalized exponent $\mathbf{E}$ (001 .. 110)

i. Give an example of a fractional decimal numbers that cannot be represented using this 6-bit floating-point representation, but is within the "range" of representable values.

11.0 cannot be represented but it is within the range

What does Epsilon mean?

small positive quantity

1 : the 5th letter of the Greek alphabet — see Alphabet Table. 2 : an arbitrarily small positive quantity in mathematical analysis.

From lecture 6 Slide 15

- **exp** and **frac**: interpreted as **unsigned** values
- If **frac** = 000…0 -> M = 1.0
- If **frac** = 111…1 -> M = 2.0 – ε (where ε means a very small value)

k. How close is the value of the **frac** of the largest normalized number to 1? In other words, how close is **M** to 2, i.e., what is ε (epsilon) in this equation: 1 <= **M** < 2 ~~– ε~~? Express ε as a fractional decimal number.

First, let's fix the above equation "1 <= **M** < 2 – ε". It should be 1 <= **M** < 2.

Answer:

The value of the "frac" of the largest normalized number is .11 -> ¾ = 0.75_{10}

How close is the value of the "frac" of the largest normalized number to 1 -> ¼ = 0.25_{10}

So, ε (epsilon) is ¼ = 0.25_{10}

1.0 <= M < 2.0
1.0 <= M <= 2.0 – ε
1.0 <= (1 + frac) <= 2.0 – ε
0.0 <= frac <= 1.0 – ε

Assignment#3 Question 1

1. [10 points] Memory addressing modes – Marked by Aditi

   Assume the following values are stored at the indicated memory addresses and registers:

Memory Address	Value
0x230	0x23
0x234	0x00
0x235	0x01
0x23A	0xed
0x240	0xff

Register	Value
%rdi	0x230
%rsi	0x234
%rcx	0x4
%rax	0x1

   Imagine that the operands in the table below are the **Src** (source) operands for some unspecified assembly instructions (any instruction except `lea*`), fill in the following table with the appropriate answers.

   Note: We do not need to know what these assembly instructions are in order to fill the table.

Operand	Operand Value (expressed in hexadecimal)	Operand Form (Choices are: Immediate, Register or one of the 9 Memory Addressing Modes)
%rsi	0x234	Register
(%rdi)	0x23	Indirect memory addressing mode
$0x23A	0x23A	Immediate value
0x240	0xff	Absolute memory addressing mode (this answer is preferable to “Imm” as it is more specific than “Imm” and highlights the fact that it does not require a “$” – see first row of table below)
10(%rdi)	0xed	“Base + displacement” memory addressing mode
560(%rcx,%rax)	0x01	Indexed memory addressing mode
-550(, %rdi, 2)	0xed	Scaled indexed memory addressing mode
0x6(%rdi, %rax, 4)	0xed	Scaled indexed memory addressing mode

Still using the first table listed above displaying the values stored at various memory addresses and registers, fill in the following table with three different **Src** (source) operands for some unspecified assembly instructions (any instruction except `lea*`). For each row, this operand must result in the operand **Value** listed and must satisfy the **Operand Form** listed.

Operand	Value	Operand Form (Choices are: Immediate, Register or one of the 9 Memory Addressing Modes)
`0x234`	0x00	Absolute memory addressing mode
`(%rdi, %rax, 4)`	0x00	Scaled indexed memory addressing mode
`(%rdi, %rcx)`	0x00	Indexed memory addressing mode

Other answers are possible!

Assignment#3 Question 2

2. [2 marks] Machine level instructions and their memory location Marked by Aditi

   Consider a function called **`arith`**, defined in a file called **`arith.c`** and called from the main function found in the file called **`main.c`**.

   This function **`arith`** performs some arithmetic manipulation on its **three parameters**.

   Compiling **`main.c`** and **`arith.c`** files, we created an executable called **`ar`**, then we executed the command:

```
objdump -d ar > arith.objdump
```

We display the partial content of **arith.objdump** below. The file **arith.objdump** is the disassembled version of the executable file **ar**.

Your task is to fill in its missing parts, which have been underlined:

```
0000000000400527 <arith>:
  400527:   48 8d 04 37        lea    (%rdi,%rsi,1),%rax
  40052b:   48 01 d0           add    %rdx,%rax
  40052e:   48 8d 0c 76        lea    (%rsi,%rsi,2),%rcx
  400532:   48 c1 e1 04        shl    $0x4,%rcx
  400536:   48 8d 54 0f 04     lea    0x4(%rdi,%rcx,1),%rdx
  40053b:   48 0f af c2        imul   %rdx,%rax
  40053f:   c3                 retq
```

Hand tracing code!

Assignment#4 Question 2

In the assembly code, there are a lot more steps than in the C code, so how to match them and create the C code.

Consider the following assembly code:

```
# long func(long x, int n)
# x in %rdi, n in %esi, result in %rax
func:
      movl   %esi, %ecx
      movl   $1,   %edx
      movl   $0,   %eax
      jmp    cond
loop:
      movq   %rdi, %r8
      andq   %rdx, %r8
      orq    %r8,  %rax
      salq   %cl,  %rdx   # shift left %rdx by content of %cl*
cond:
      testq %rdx, %rdx    # %rdx <- %rdx & %rdx
      jne    loop         # jump if not zero (when %rdx & %rdx != 0)
                          # fall thru to ret (when %rdx & %rdx == 0
      ret
```

The preceding code was generated by compiling C code that had the following overall form:

```
long func(long x, int n) {
    long result = _________;
    long mask;

    for (mask = _________ ;mask _________ ;mask = _________ )
        result |= ______________________________ ;
    return result;
}
```

From our Lectures 14 and 15

Example

caller rdi rsi rdx

```
void multstore(long x, long y, long *dest) {
    long t = mult2(x, y);
    *dest = t;
    return;
}
```

callee rdi rsi

```
long mult2(long a, long b) {
  long s = a * b;
  return s;
}
```

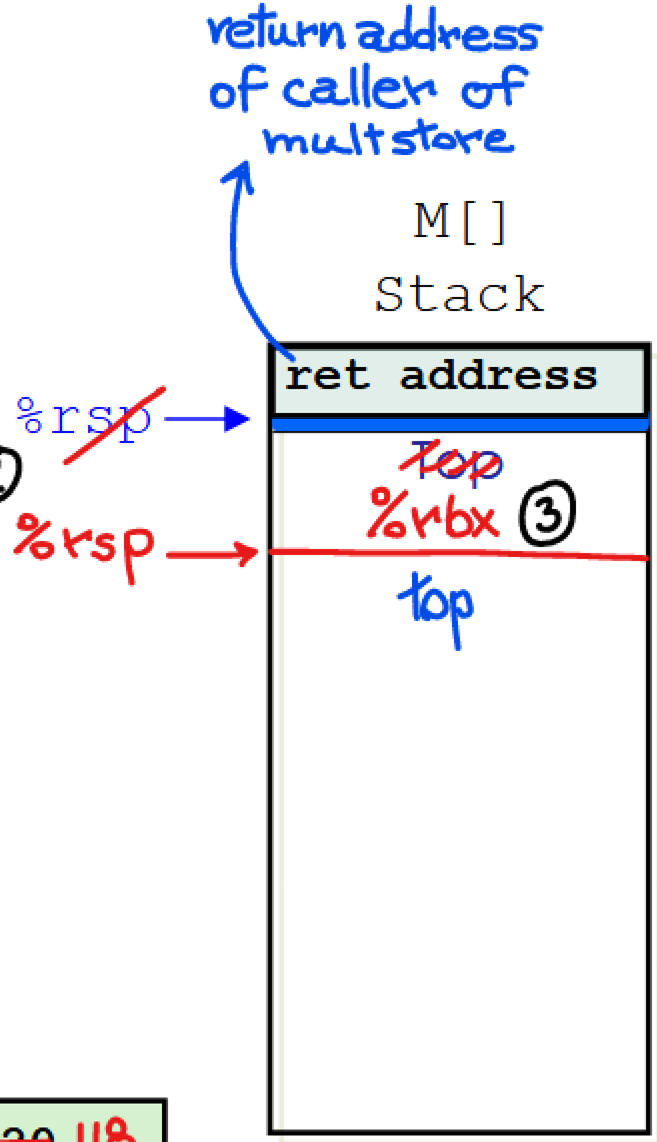
```
0000000000400540 <multstore>:
  400540: push   %rbx            # Save %rbx
  400541: mov    %rdx,%rbx       # Save dest
  400544: callq  400550 <mult2>  # mult2(x,y)
  400549: mov    %rax,(%rbx)     # Save at dest
  40054c: pop    %rbx            # Restore %rbx
  40054d: retq                   # Return
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax   # a
  400553:  imul   %rsi,%rax   # a * b
  400557:  retq               # Return
```

Example – Steps 1 and 2

```
0000000000400540 <multstore>:
1. 400540: push   %rbx              # Save %rbx
2. 400541: mov    %rdx,%rbx         # Save dest
   400544: callq  400550 <mult2>    # mult2(x,y)
   400549: mov    %rax,(%rbx)       # Save at dest
   40054c: pop    %rbx              # Restore %rbx
   40054d: retq                     # Return
```

```
0000000000400550 <mult2>:
  400550:  mov    %rdi,%rax    # a
  400553:  imul   %rsi,%rax    # a * b
  400557:  retq                # Return
```



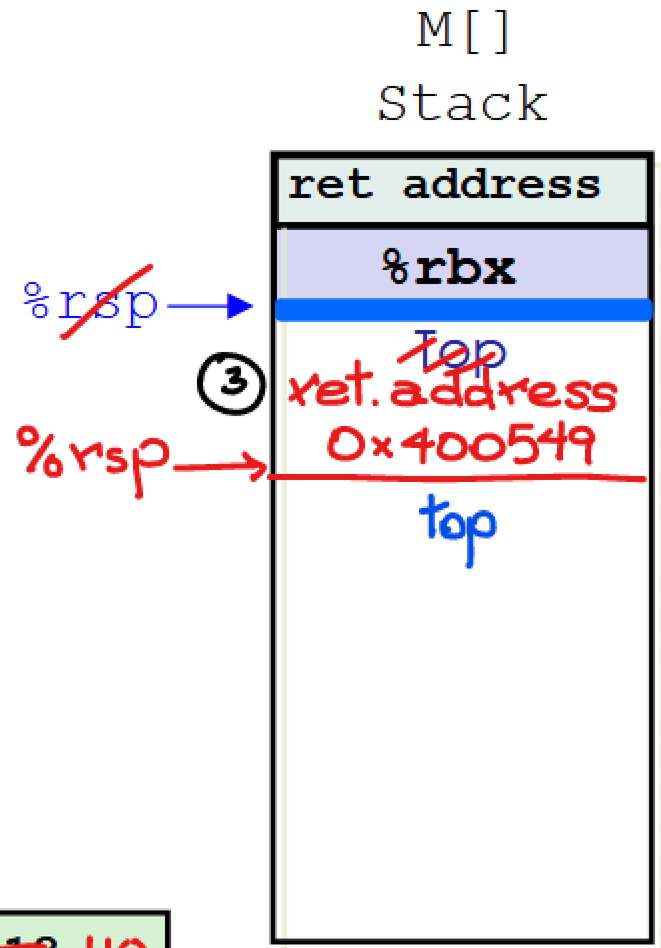
%rdi	x	%rbx	dest ⑤	%rsp	0x120 118
%rsi	y	%rax		%rip (PC)	0x400540 544 ① 544 ④
%rdx	dest → mem. address				

Example – Steps 3 and 4

```
0000000000400540 <multstore>:
  400540: push    %rbx              # Save %rbx
  400541: mov     %rdx,%rbx         # Save dest
3.400544: callq   400550 <mult2>    # mult2(x,y)
  400549: mov     %rax,(%rbx)       # Save at dest
  40054c: pop     %rbx              # Restore %rbx
  40054d: retq                      # Return
```

⑤

```
0000000000400550 <mult2>:
4.400550:   mov    %rdi,%rax    # a
  400553:   imul   %rsi,%rax    # a * b
  400557:   retq                # Return
```



%rdi	X → a	%rbx	dest	%rsp	0x118 110
%rsi	y → b	%rax	a	%rip (PC)	0x400544 549 ① 550 ④ 553
%rdx	dest				

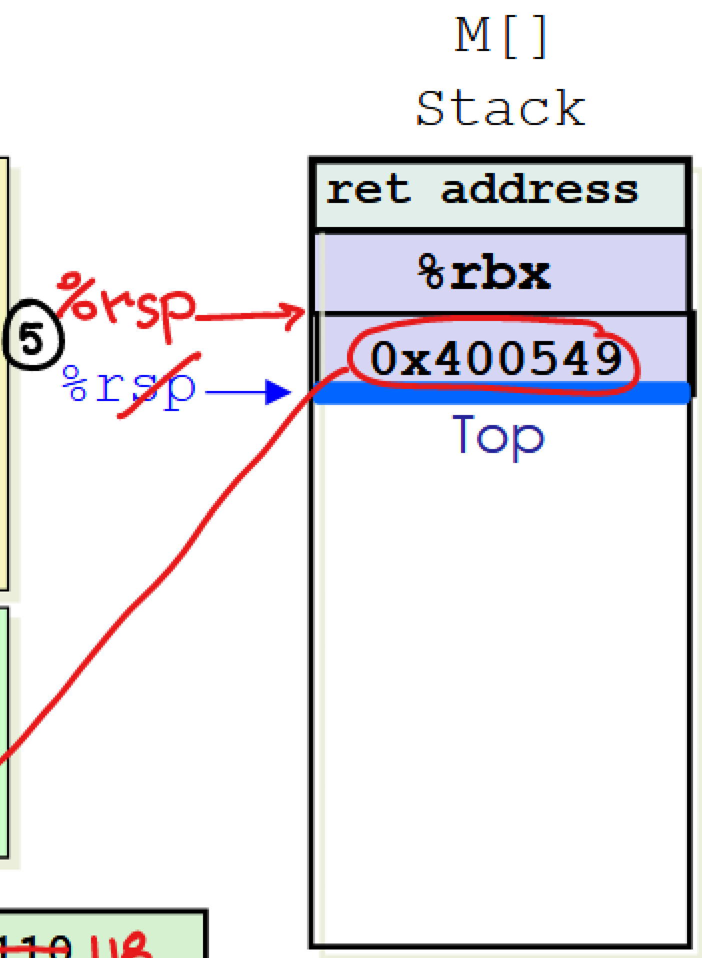
Example – Steps 5 and 6

```
0000000000400540 <multstore>:
  400540: push   %rbx              # Save %rbx
  400541: mov    %rdx,%rbx         # Save dest
  400544: callq  400550 <mult2>    # mult2(x,y)
  400549: mov    %rax,(%rbx)       # Save at dest
  40054c: pop    %rbx              # Restore %rbx
  40054d: retq                     # Return
```

```
0000000000400550 <mult2>:
   400550:  mov    %rdi,%rax    # a
5. 400553:  imul   %rsi,%rax    # a * b
6. 400557:  retq                # Return
```

%rdi	x→a	%rbx	dest	%rsp	0x~~110~~ 118
%rsi	y→b	%rax	a * b	%rip (PC)	0x400~~553~~ ~~557~~ 549
%rdx	dest				

~~address of next instr~~



Homework

Example – Steps 7, 8 and 9

```
0000000000400540 <multstore>:
   400540: push   %rbx              # Save %rbx
   400541: mov    %rdx,%rbx         # Save dest
   400544: callq  400550 <mult2>    # mult2(x,y)
7. 400549: mov    %rax,(%rbx)       # Save at dest
8. 40054c: pop    %rbx              # Restore %rbx
9. 40054d: retq                     # Return
```

```
0000000000400550 <mult2>:
   400550:  mov    %rdi,%rax     # a
   400553:  imul   %rsi,%rax     # a * b
   400557:  retq                 # Return
```

M[]
Stack

ret address
%rbx
Top

%rsp → (points below %rbx, at Top)

Register	Value	Register	Value	Register	Value
%rdi	x (a)	%rbx	dest	%rsp	0x118
%rsi	y (b)	%rax	a*b	%rip	0x400549
%rdx	dest				

Next next Lecture

- Introduction
  - C program -> assembly code -> machine level code
- Assembly language basics: data, `move` operation
  - Memory addressing modes
- Operation `leaq` and Arithmetic & logical operations
- Conditional Statement – Condition Code + `cmovX`
- Loops
- Function call – Stack – Recursion
  - Overview of Function Call
  - Memory Layout and Stack - x86-64 instructions and registers
  - Passing control
  - Passing data – Calling Conventions
  - Managing local data
  - Recursion
- Array
- Buffer Overflow
- Floating-point operations