18-600 Recitation #9

Mid-Term I Review

October 24th, 2017

Announcements

- Shell Lab is due on 10/26
- Midterm Rooms PGH:
  - DH A302: Sections A, B, C
  - HH 1107: Section D
- Midterm Rooms SV:
  - B23 118: Section SA
  - B23 211: Section SB
- Time: 10/30, 3:30 - 6pm (Pacific)
- Cheatsheet: One hand-written or printed double sided 8 ½ x 11-inch paper with no worked out problems.

Bits & Bytes

Please fill in the following table, assuming the following ---

- An 8-bit machine using two's complement arithmetic for signed integers.
- Right shifts on signed integers are arithmetic.
- Right shifts on unsigned integers are logical.
- x and y are signed integers, unless otherwise specified.

Expression	Decimal	Binary
$-T_{min}$		
`x`		10011000
`y`	63	
`(unsigned)x`		
`x || 0xdeadbeef`		
`-x`		
`x >> 2`		
`0x18 & y`		
`x > y`		
`((unsigned) x) >> 2`		

Bits & Bytes

Expression	Decimal	Binary
$-T_{min}$	-128	10000000
`x`	-104	10011000
`y`	63	00111111
`(unsigned)x`	152	10011000
`x || 0xdeadbeef`	1	00000001
`-x`	104	01101000
`x >> 2`	-26	11100110
`0x18 & y`	24	00011000
`x > y`	0	00000000
`((unsigned) x) >> 2`	38	00100110

Dynamic Range (Positive Only)

$v = (-1)^s M\, 2^E$

n: E = Exp – Bias

d: E = 1 – Bias

	s exp frac	E	Value	
Denormalized numbers	0 0000 000	-6	0	
	0 0000 001	-6	1/8*1/64 = 1/512	closest to zero
	0 0000 010	-6	2/8*1/64 = 2/512	
	…			
	0 0000 110	-6	6/8*1/64 = 6/512	
	0 0000 111	-6	7/8*1/64 = 7/512	largest denorm
Normalized numbers	0 0001 000	-6	8/8*1/64 = 8/512	smallest norm
	0 0001 001	-6	9/8*1/64 = 9/512	
	…			
	0 0110 110	-1	14/8*1/2 = 14/16	
	0 0110 111	-1	15/8*1/2 = 15/16	closest to 1 below
	0 0111 000	0	8/8*1 = 1	
	0 0111 001	0	9/8*1 = 9/8	closest to 1 above
	0 0111 010	0	10/8*1 = 10/8	
	…			
	0 1110 110	7	14/8*128 = 224	
	0 1110 111	7	15/8*128 = 240	largest norm
	0 1111 000	n/a	inf	

Assembly

Structures & Unions

```
struct bar {
int a; // can use both a and b simultaneously
char b;
 } bar;
```

```
union foo {
 int a; // can't use both a and b at once
 char b;
 }  foo;
```

```
struct bar y;
y.a = 3; // OK
y.b = 'c'; // OK
```

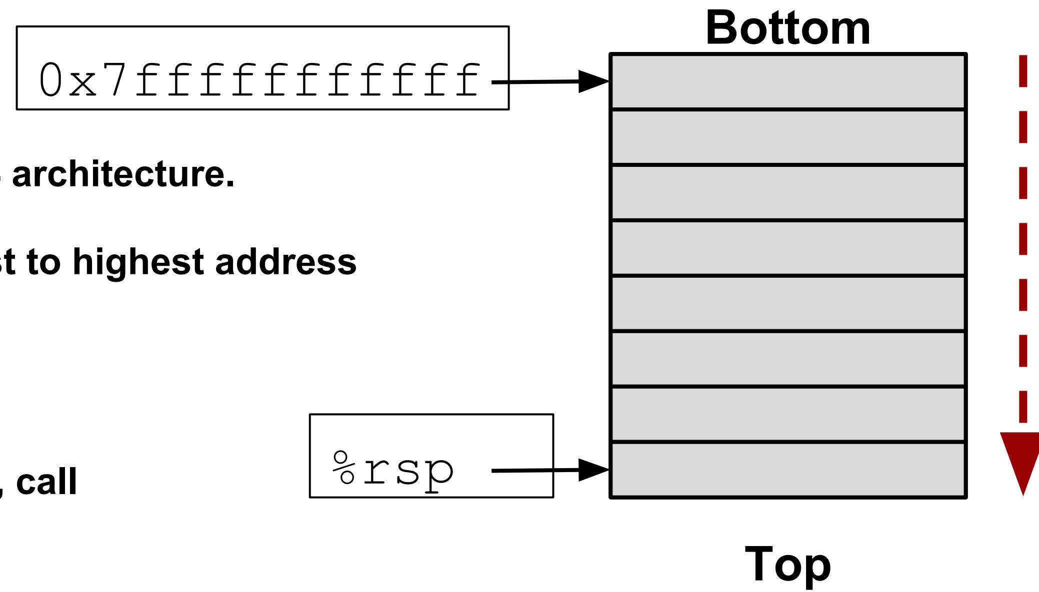
```
union foo x;
x.a = 3; // OK
x.b = 'c'; // NO! this affects the value of x.a
since compiler allocates memory for the
largest of all the members
```

Be comfortable with pointers and dereferencing
The use of parenthesis in mov commands.
%rax vs. (%rax)

Stack

Key points to remember:

- **Encodings are little endian on x86-64 architecture.**
- **Strings are always stored from lowest to highest address (Endianness does not apply)**
- **Register conventions in x86-64**
- **How rsp changes with push, pop, ret, call**



Superscalar Architecture - Refresher

Key Concepts in Superscalar architecture:

- **REFER TO SUPERSCALAR QUESTION IN MOCK EXAM AND SOLVE IT INDEPENDENTLY!!**
- Understand the behavior of execution-units with and without a superscalar design and their impact on execution time of instructions through execution unit.

- Understand the dependencies like:
  - **Read-After-Write**
  - **Write-After-Read**
  - **Write-After-Write**

- Which of the above 3 dependencies cannot be avoided by architectural modifications or register renaming ? i.e which is/are true dependency among them ? Why ?
  Answer: Read-After-Write, as that dependency requires the use of a new value in the subsequent instruction, else it would break program order.

Superscalar

Key Concepts in Superscalar architecture:

- Concepts regarding encoding of an instruction.
  Example: Say given 16 instructions, and 32 architectural registers to manage in an instruction set, with formatting like MOV R1, R2, R3. How many minimum bits are needed to encode this instruction.
  Answer: Given 16 instructions → 4 bits to distinguish between them
  Given 32 registers → 5 bits to distinguish among them.
  Given the instruction formatting of MOV R1, R2, R3 this would need a total of
  4 + 5 + 5 + 5 bits = 19 bits

- A complete understanding of the examples in Slides 19-26 will be beneficial.
  http://www.ece.cmu.edu/~ece600/recitations/recitation06.pdf
- Walk through the example in Arch Lab recitation where registers are mapped from architectural registers to physical registers for a refresher on register renaming.
- Understand which dependencies are eliminated in the table for the example.

Pipelining

- Understand how hazards create stalls and bubbles within the pipeline
  - Data hazards (e.g. RAW dependencies, load-to-use)
    - How does data forwarding help?
  - Structural hazards (e.g. full RS/ROB)
  - Control hazards (e.g. branch misprediction)
  - Refer to *Lecture #8: pg. 27 - 69*
- Being able to **draw** the pipeline stages with Assembly code given.
  - Refer to *Mock Midterm*.