University of Washington – Computer Science & Engineering

Autumn 2016 Instructor: Justin Hsia 2016-11-02

CSE351 MIDTERM B

Last Name:	
First Name:	
Student ID Number:	
Section you attend (circle):	Chris Yufang · John · Kevin · Sachin · Suraj Waylon · Thomas · Xi
Name of person to your Left | Right	|
All work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CSE351 who haven't taken it yet. **(please sign)**	

Do not turn the page until 2:30.

Instructions

- This exam contains 10 pages, including this cover page. Show scratch work for partial credit, but put your final answers in the boxes and blanks provided.
- The last page is a reference sheet. Feel free to detach it from the rest of the exam.
- The exam is closed book (no laptops, tablets, wearable devices, or calculators). You are allowed one page (US letter, double-sided) of *handwritten* notes.
- Please silence and put away all cell phones and other mobile or noise-making devices. Remove all hats, headphones, and watches.
- You have 50 minutes to complete this exam.

Advice

- Read questions carefully before starting. Skip questions that are taking a long time.
- Read *all* questions first and start where you feel the most confident.
- Relax. You are here to learn.

Question	1	2	3	4	5	Total
Possible Points	8	12	11	12	12	**55**

Question 1: Computer Architecture Design [8 pts]

Answer the following questions in the boxes provided with a **single sentence fragment**. Please try to write as legibly as possible.

(A) Why can't we upgrade to more registers like we can with memory? [2 pt]

(B) Why don't we see new assembly instruction sets as frequently as we see new programming languages? [2 pt]

(C) Name one reason why a program written in a CISC language might run slower than the same program written in a RISC language and one reason why the reverse might be true: [4 pt]

CISC slower:	RISC slower:

Question 2: Number Representation [12 pts]

(A) What is the value of the `char` 0b 1101 1010 in decimal? [1 pt]

(B) What is the value of **`char`** `z = (0xB << 6)` in decimal? [1 pt]

(C) Let `char x = 0xC0`. Give one value (in hex) for `char y` that results in *both* signed and unsigned overflow for `x+y`. [2 pt]

For the rest of this problem we are working with a floating point representation that follows the same conventions as IEEE 754 except using 8 bits split into the following vector widths:

Sign (1)	Exponent (4)	Mantissa (3)

(D) What is the *magnitude* of the **bias** of this new representation? [2 pt]

(E) Translate the floating point number 0b 1100 1110 into decimal. [3 pt]

(F) What is the smallest positive integer that can't be represented in this floating point encoding scheme? Hint: For what integer will the "one's digit" get rounded off? [3 pt]

Question 3: C & Assembly [11 pts]

We are writing the function `toLower`, which takes a `char` pointer and converts a string of letters (assume only letters and spaces) to lowercase, leaving spaces as spaces. Example: If the pointer `p` points to `"TeST oNe"`, then after `toLower(p)`, `p` now points to `"test one"`.

ASCII	**'A'**	**'Z'**	**Space**
Binary	`0b 0100 0001`	`0b 0101 1010`	`0b 0010 0000`
Binary	`0b 0110 0001`	`0b 0111 1010`	`0b 0010 0000`
ASCII	**'a'**	**'z'**	**Space**

(A) Using the table of ASCII values (in binary) above, complete the function using a bitwise operator: [2 pt]

```
void toLower (char * p) {
    while(*p != 0) {

        *p = ___________________;

        p++;
    }
}
```

(B) Fill in the blanks in the x86-64 code below with the correct instructions and operands. *Remember to use the proper size suffixes and correctly-sized register names!* You may assume that Lines 4, 7, and 8 are correctly filled in. [9 pt]

```
   toLower(char*):

 1     movzbq  ______, %rax        # get *p

 2     ______  ______, ______      # conditional

 3     ______  ______              # conditional jump
   .Loop:
 4     <<answer to part (A)>>      # to lowercase

 5     movb    %al, _________      # update char in memory

 6     ______  ______, %rdi        # increment p
 7     <<same as Line 1>>          # get new *p
 8     <<same as Line 2>>          # conditional

 9     ______  .Loop               # conditional jump
   .Exit:

10     ______                      # return
```

Question 4: Pointers & Memory [12 pts]

For this problem we are using a 64-bit x86-64 machine (**little endian**). The initial state of memory (values in hex) is shown below:

Word Addr	+0	+1	+2	+3	+4	+5	+6	+7
0x00	AC	AB	03	01	BA	5E	BA	11
0x08	5E	00	AB	0C	BE	A7	CE	FA
0x10	1D	B0	99	DE	AD	60	BB	40
0x18	14	CD	FA	1D	D0	41	ED	77
0x20	BA	B0	FF	20	80	AA	BE	EF

```
char* cp = 0x10
short* sp = 0x08
unsigned* up = 0x24
```

(A) What are the values (in hex) stored in each register shown after the following x86 instructions are executed? Remember to use the appropriate bit widths. [6 pt]

```
leaw (%rsi, %rdi), %ax
movb 8(%rdi), %bl
movswl (,%rdi,4), %ecx
```

Register	Value (hex)
%rdi	0x0000 0000 0000 0003
%rsi	0x0000 0000 0000 0005
%ax	
%bl	
%rcx	

(B) It's a memory scavenger hunt! Complete the C code below to fulfill the behaviors described in the comments using pointer arithmetic. [6 pt]

```
long v1 = (long) *(cp + _____);  // set v1 = 0x60
unsigned* v2 = up + _____;        // set v2 = 64
long v3 = *(long *)(sp + _____); // set v3 = 0xB01DFACE
```

Question 5: The Stack [12 pts]

The recursive factorial function `fact()` and its x86-64 disassembly is shown below:

```
int fact(int n) {
     if(n==0 || n==1)
          return 1;
     return n*fact(n-1);
```

```
000000000040052d <fact>:
  40052d:  83 ff 00         cmpl   $0, %edi
  400530:  74 05            je     400537 <fact+0xa>
  400532:  83 ff 01         cmpl   $1, %edi
  400535:  75 07            jne    40053e <fact+0x11>
  400537:  b8 01 00 00 00   movl   $1, %eax
  40053c:  eb 0d            jmp    40054b <fact+0x1e>
  40053e:  57               pushq  %rdi
  40053f:  83 ef 01         subl   $1, %edi
  400542:  e8 e6 ff ff ff   call   40052d <fact>
  400547:  5f               popq   %rdi
  400548:  0f af c7         imull  %edi, %eax
  40054b:  f3 c3            rep ret
```

(A) Circle one: [1 pt] `fact()` is saving `%rdi` to the Stack as a **Caller** // **Callee**

(B) How much space (in bytes) does this function take up in our final executable? [2 pt]

(C) **Stack overflow** is when the stack exceeds its limits (i.e. runs into the Heap). Provide an argument to `fact(n)` here that will cause stack overflow. [2 pt]

«Problem continued on next page»

(D) If we use the `main` function shown below, answer the following for the execution of the entire program: [4 pt]

```
void main() {
     printf("result = %d\n",fact(4));
}
```

Total frames created:	Maximum stack frame depth:

(E) In the situation described above where `main()` calls `fact(4)`, we find that the word `0x2` is stored on the Stack at address `0x7fffdc7ba888`. At what address on the Stack can we find the return address to `main`()? [3 pt]

This page purposely left blank

CSE 351 Reference Sheet

Binary	Decimal	Hex
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	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
1	2	4	8	16	32	64	128	256	512	1024

IEEE 754 FLOATING-POINT STANDARD

Value: $\pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}}$

Bit Fields: $(-1)^S \times 1.M \times 2^{(E+\text{bias})}$

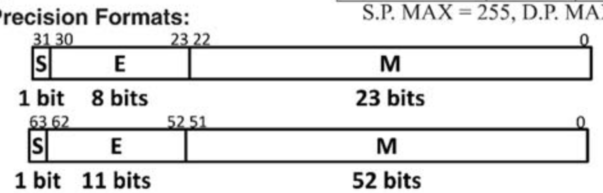
where Single Precision Bias = -127,
Double Precision Bias = -1023.

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

IEEE Single Precision and Double Precision Formats:



Assembly Instructions

`mov a, b`	Copy from `a` to `b`.
`movs a, b`	Copy from `a` to `b` with sign extension.
`movz a, b`	Copy from `a` to `b` with zero extension.
`lea a, b`	Compute address and store in `b`. *Note:* the scaling parameter of memory operands can only be 1, 2, 4, or 8.
`push src`	Push `src` onto the stack and decrement stack pointer.
`pop dst`	Pop from the stack into `dst` and increment stack pointer.
`call <func>`	Push return address onto stack and jump to a procedure.
`ret`	Pop return address and jump there.
`add a, b`	Add from `a` to `b` and store in `b` (and sets flags).
`imul a, b`	Multiply `a` and `b` and store in `b` (and sets flags).
`and a, b`	Bitwise AND of `a` and `b`, store in `b` (and sets flags).
`sar a, b`	Shift value of `b` *right* (*arithmetic*) by `a` bits, store in `b` (and sets flags).
`shr a, b`	Shift value of `b` *right* (*logical*) by `a` bits, store in `b` (and sets flags).
`shl a, b`	Shift value of `b` *left* by `a` bits, store in `b` (and sets flags).
`cmp a, b`	Compare `b` with `a` (compute `b-a` and set condition codes based on result).
`test a, b`	Bitwise AND of `a` and `b` and set condition codes based on result.
`jmp <label>`	Unconditional jump to address.
`j* <label>`	Conditional jump based on condition codes (*more on next page*).
`set* a`	Set byte based on condition codes.

Conditionals

Instruction		`cmp b, a`	`test a, b`
je	"Equal"	a == b	a & b == 0
jne	"Not equal"	a != b	a & b != 0
js	"Sign" (negative)		a & b < 0
jns	(non-negative)		a & b >= 0
jg	"Greater"	a > b	a & b > 0
jge	"Greater or equal"	a >= b	a & b >= 0
jl	"Less"	a < b	a & b < 0
jle	"Less or equal"	a <= b	a & b <= 0
ja	"Above" (unsigned >)	a > b	
jb	"Below" (unsigned >)	a < b	

Sizes

C type	x86-64 suffix	Size (bytes)
char	b	1
short	w	2
int	l	4
long	q	8

Registers

Name	Convention	Name of "virtual" register		
		Lowest 4 bytes	Lowest 2 bytes	Lowest byte
%rax	Return value – Caller saved	%eax	%ax	%al
%rbx	Callee saved	%ebx	%bx	%bl
%rcx	Argument #4 – Caller saved	%ecx	%cx	%cl
%rdx	Argument #3 – Caller saved	%edx	%dx	%dl
%rsi	Argument #2 – Caller saved	%esi	%si	%sil
%rdi	Argument #1 – Caller saved	%edi	%di	%dil
%rsp	Stack Pointer	%esp	%sp	%spl
%rbp	Callee saved	%ebp	%bp	%bpl
%r8	Argument #5 – Caller saved	%r8d	%r8w	%r8b
%r9	Argument #6 – Caller saved	%r9d	%r9w	%r9b
%r10	Caller saved	%r10d	%r10w	%r10b
%r11	Caller saved	%r11d	%r11w	%r11b
%r12	Callee saved	%r12d	%r12w	%r12b
%r13	Callee saved	%r13d	%r13w	%r13b
%r14	Callee saved	%r14d	%r14w	%r14b
%r15	Callee saved	%r15d	%r15w	%r15b