Name: Jeff Guo UIN: 670009684

CS 261 Machine Organisation (Fall 2019)-Homework 2

Due Date: 5:00 p.m., September 16, 2019
Late Due Date: 5:00 p.m., September 17, 2019

This assignment is due via Gradescope.

For Problems A and B, you need to fill in the forms in this pdf file and submit the entire pdf file to "Hwk 2AB"on Gradescope.

For Problem C, you need to submit a hwkTwoC.c file. This file should only contain your solution to the problem, in a function named "hwkTwoC". Submit your .c file to "Hwk 2C" on Gradescope. Auto-grading of 20/35 points will be immediately available. The remaining 15/35 points will be auto-graded as well, however this will not be visible until after the due date.

Problem A (25 points):

Consider the following assembly language code:

```
.LC0:
        .string  "answer %d\n"
        .text
        .globl   main
        .type    main, @function
main:
        pushq   %rbp
        movq    %rsp, %rbp
        subq    $16, %rsp
        movl    $12, -8(%rbp)
        movl    $3, -4(%rbp)
        movl    $0, -16(%rbp)
        movl    -8(%rbp), %eax
        xorl    -4(%rbp), %eax
        movl    %eax, -12(%rbp)
        jmp     .L2
.L3:
        movl    -12(%rbp), %eax
        andl    $1, %eax
        addl    %eax, -16(%rbp)
        sarl    -12(%rbp)
.L2:
        cmpl    $0, -12(%rbp)
        jne     .L3
        movl    -16(%rbp), %eax
        movl    %eax, %esi
        leaq    .LC0(%rip), %rdi
        movl    $0, %eax
        call    printf@PLT
        movl    $0, %eax
        leave
        ret
```

This code came from skeleton C file below after optimizing with O0. This means “gcc -O0 -S -fno-asynchronous-unwind-tables” command was used to convert C File into assembly file. (The last option was used to disable cfi directives)

Complete the C code given below using the provided assembly code (5 blanks * 5 points per blank = 25 points).

One approach may be to ignore the skeleton file and create an equivalent C code from the assembly code, and then rewrite the code to fit the skeleton file.

```
int main(){
   int a = ____12____, b = 3;
   int count = ____0____;
   for(int c = a^____b; c != 0; c = c-____1){
      count += ( c&____1 );
   }
   printf("answer %d\n", count);
   return 0;
}
```

Problem B (40 points):

Consider the following initial values for the registers and memory. Values are reset back to the values shown in the table before each operation. Determine the result of the following operations and state where the result is stored.

(40 points, 2 points for destination, 3 points for result)

Address	Value	Register	Value
0x80	0xF1	%r8	0x50
0x58	0xA9	%r9	0xAD0AD
0x76	0xC4	%r10	0xCE001
0x84	0xB7	%rax	0x80

Instruction	Destination	Result
incl %r8d	0x50	0x81
movb %r10b, %r9b	%r9b	0x58
cwtl	%ax	%eax
orq 8(%r8), %r10	%r10	0xCE059
shrb $1, %r10b	%r10b	0x00
sarw $3, %r9w	%r9w	0x5A15
movzwl %ax, %r8d	%r8d	0x80
negq 4(%rax)	%rax	0x84

Problem C (35 points):

A bitwise rotation, or circular shift, is an operation that moves the bits in the operand by wrapping around the end. For example, in a left shift, newly vacant bit positions (in the least significant end) are filled in with zeros. But in a rotating left shift, the bit positions are instead filled with the bits (from the most significant bit positions) that were shifted or were "pushed out".

Circular shift operations find applications in fields such as cryptography, and in a slightly more morbid scenario, as a solution to the Josephus problem.

Let us consider a similar operation as described above, but applicable to only the upper 32 bits of an unsigned integer x. That is, the lower 32 bits of the result are identical to the lower 32 bits of the original integer x. The upper 32 bits, however, are left circular shifted 'n' number of times (0 <= n <= 32).

To demonstrate what that implies, let us consider a scaled down example with a 16-bit original integer and n=3. If the original number is 0xA1CE, the function returns 0x0DCE. Notice the upper 8 bits A1(10100001) have been circular left rotated 3 times to become 0D(00001101), while the lower 8 bits remained the same.

Assumptions:

- Left shifts of unsigned data are logical shift operations (which is the same as an arithmetic left-shift).
- Right shifts of unsigned data are logical shift operations.

Forbidden:

- Casting, either explicit or implicit.
- Relative comparison operators (<, >, <=, and >=).
- Division, modulus, and multiplication.
- Conditionals (if or ? :), loops, switch statements, function calls, and macro invocations.

Operations allowed:

- All bit-level and logic operations
- Equality (==) and inequality (!=) only for test cases
- Addition, Subtraction
- Integer constants INT_MAX and INT_MIN

Keeping in mind the above, write a C function hwkTwoC with the following prototype, and save it in a file named hwkTwoC.c for submission on Gradescope.

```
/*

*hwkTwoC  – returns result after performing circular left shift n times on the 32 higher order bits of
x

[ 0 <= n <= 32 ]

*/

unsigned long hwkTwoC(unsigned long x, int n);
```