Assembly: loops

COSC 208, Introduction to Computer Systems, 2022-03-29

Announcements

- Project 2 due this Thursday

Outline

- Warm-up
- while loops
- Loops duality

Warm-up

Q1: *Assume the registers currently hold the following values:*

```
sp = 0xA980
w/x0 = 0
w/x1 = 1
w/x2 = 2
w/x3 = 3
w/x4 = 4
w/x5 = 5
```

Draw the contents of the stack after the following instructions have been executed:

```
sub sp, sp, #0x30
str w0, [sp, #16]
str x1, [sp]
str w2, [sp, #20]
str x3, [sp, #32]
str w4, [sp, #28]
str w5, [sp, #8]
```

Q2: *The following C code was compiled into assembly (using `gcc`). Label each line of assembly code with the line number of the line of C code from which the assembly instruction was derived.*

```
1   int abs(int value) {
2       if (value < 0) {
3            value = value * -1;
4       }
5       return value;
6   }
```

```
000000000000074c <abs>:
    74c:   d10043ff        sub     sp, sp, #0x10    //
    750:   b9000fe0        str     w0, [sp, #12]    //
    754:   b9400fe0        ldr     w0, [sp, #12]    //
    758:   7100001f        cmp     w0, #0x0         //
    75c:   5400008a        b.ge    76c <abs+0x20>   //
    760:   b9400fe0        ldr     w0, [sp, #12]    //
    764:   4b0003e0        neg     w0, w0           //
    768:   b9000fe0        str     w0, [sp, #12]    //
    76c:   b9400fe0        ldr     w0, [sp, #12]    //
    770:   910043ff        add     sp, sp, #0x10    //
    774:   d65f03c0        ret                      //
```

while loops

- Mapping C while loops to assembly code

```
1   int pow2(int n) {
2       int result = 1;
3       while (n > 0) {
4           result = result * 2;
5           n = n - 1;
6       }
7       return result;
8   }
```

```
0000000000400584 <pow2>:
    400584:       d10043ff        sub     sp, sp, #0x10          //
    400588:       b9000fe0        str     w0, [sp, #12]          //
    40058c:       52800028        mov     w8, #0x1               //
    400590:       b9000be8        str     w8, [sp, #8]           //
    400594:       b9400fe8        ldr     w8, [sp, #12]          //
    400598:       7100011f        cmp     w8, #0x0               //
    40059c:       37000128        b.le    4005c0 <pow2+0x3c>     //
    4005a0:       b9400be8        ldr     w8, [sp, #8]           //
    4005a4:       52800049        mov     w9, #0x2               //
    4005a8:       1b097d08        mul     w8, w8, w9             //
    4005ac:       b9000be8        str     w8, [sp, #8]           //
    4005b0:       b9400fe8        ldr     w8, [sp, #12]          //
    4005b4:       71000508        subs    w8, w8, #0x1           //
    4005b8:       b9000fe8        str     w8, [sp, #12]          //
    4005bc:       17fffff5        b       400594 <pow2+0x10>     //
    4005c0:       b9400be0        ldr     w0, [sp, #8]           //
    4005c4:       910043ff        add     sp, sp, #0x10          //
    4005c8:       d65f03c0        ret                            //
```

- Goto form

```
int pow2_goto(int n) {
    int result = 1;
loop_top:
    if (n <= 0)
        goto _________________;
    result = result * 2;
    n = n - 1;
    goto _________________;
after_while:
    return result;
}
```

Loop duality

Q3: *Write a function called `tally_while` that is semantically equivalent to the function below, but uses a `while` loop instead of a `for` loop.*

```
int tally_for(int x) {
    int result = 0;
    for (int i = 1; i <= x; i++) {
        result = result + i;
    }
    return result;
}
```

```

```

Q4: *The following C code was compiled into assembly (using `clang`). Label each line of assembly code with the line number of the line of C code from which the assembly instruction was derived.*

```c
int powi(int m, int n) {
    int result = 1;
    for (int i = 0; i < n; i++) {
        result *= m;
    }
    return result;
}
```

```
0000000000400544 <powi>:
    400544:    d10043ff     sub    sp, sp, #0x10        //
    400548:    b9000fe0     str    w0, [sp, #12]        //
    40054c:    b9000be1     str    w1, [sp, #8]         //
    400550:    52800028     mov    w8, #0x1             //
    400554:    b90007e8     str    w8, [sp, #4]         //
    400558:    b90003ff     str    wzr, [sp]            //
    40055c:    b94003e8     ldr    w8, [sp]             //
    400560:    b9400be9     ldr    w9, [sp, #8]         //
    400564:    6b09011f     cmp    w8, w9               //
    400568:    5400012a     b.ge    40058c <powi+0x48>  //
    40056c:    b9400fe8     ldr    w8, [sp, #12]        //
    400570:    b94007e9     ldr    w9, [sp, #4]         //
    400574:    1b087d28     mul    w8, w9, w8           //
    400578:    b90007e8     str    w8, [sp, #4]         //
    40057c:    b94003e8     ldr    w8, [sp]             //
    400580:    11000508     add    w8, w8, #0x1         //
    400584:    b90003e8     str    w8, [sp]             //
    400588:    17fffff5     b    40055c <powi+0x18>     //
    40058c:    b94007e0     ldr    w0, [sp, #4]         //
    400590:    910043ff     add    sp, sp, #0x10        //
    400594:    d65f03c0     ret                         //
```

Conditionals and loops

- Q5: *The following C code was compiled into assembly (using `clang`). For each line of assembly, indicate which original line of C code the assembly instruction was derived from.*

```c
int onebits(unsigned int num) {
    int ones = 0;
    while (num != 0) {
        if (num & 0b1) {
            ones++;
        }
        num = num >> 1;
    }
    return ones;
}
```

```
0000000000400584 <onebits>:
    400584:    d10043ff     sub    sp, sp, #0x10            //
    400588:    b9000fe0     str    w0, [sp, #12]            //
    40058c:    b9000bff     str    wzr, [sp, #8]            //
    400590:    b9400fe8     ldr    w8, [sp, #12]            //
    400594:    34000168     cbz    w8, 4005c0 <onebits+0x3c>//
    400598:    b9400fe8     ldr    w8, [sp, #12]            //
    40059c:    12000108     and    w8, w8, #0x1             //
    4005a0:    34000088     cbz    w8, 4005b0 <onebits+0x2c>//
    4005a4:    b9400be8     ldr    w8, [sp, #8]             //
    4005a8:    11000508     add    w8, w8, #0x1             //
    4005ac:    b9000be8     str    w8, [sp, #8]             //
    4005b0:    b9400fe8     ldr    w8, [sp, #12]            //
    4005b4:    53017d08     lsr    w8, w8, #1               //
    4005b8:    b9000fe8     str    w8, [sp, #12]            //
    4005bc:    17fffff5     b    400590 <onebits+0xc>       //
    4005c0:    b9400be0     ldr    w0, [sp, #8]             //
    4005c4:    910043ff     add    sp, sp, #0x10            //
    4005c8:    d65f03c0     ret                             //
```

- Q6: *Write a function called `onebits_goto` that behaves the same as `onebits` but matches the structure of the assembly code that will be generated for `onebits`.*