Assembly: loops

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

Announcements

- Project 2 due Thursday, March 31

Outline

- Warm-up
- while loops
- Loops duality
- Conditionals and loops

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]
```

```
        0  1  2  3  4  5  6  7  8
 sp ->  +--+--+--+--+--+--+--+--+
0xA950  |           1           |
        +--+--+--+--+--+--+--+--+
0xA958  |     5     |           |
        +--+--+--+--+--+--+--+--+
0xA960  |     0     |     2     |
        +--+--+--+--+--+--+--+--+
0xA968  |           |     4     |
        +--+--+--+--+--+--+--+--+
0xA970  |           3           |
        +--+--+--+--+--+--+--+--+
0xA978  |                       |
        +--+--+--+--+--+--+--+--+
0xA980  |                       |
        +--+--+--+--+--+--+--+--+
```

- 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.*

```c
int abs(int value) {
    if (value < 0) {
        value = value * -1;
    }
    return value;
}
```

```
000000000000074c <abs>:
    74c:   d10043ff        sub     sp, sp, #0x10    // 1
    750:   b9000fe0        str     w0, [sp, #12]    // 1
    754:   b9400fe0        ldr     w0, [sp, #12]    // 2
    758:   7100001f        cmp     w0, #0x0         // 2
    75c:   5400008a        b.ge    76c <abs+0x20>   // 2
    760:   b9400fe0        ldr     w0, [sp, #12]    // 3
    764:   4b0003e0        neg     w0, w0           // 3
    768:   b9000fe0        str     w0, [sp, #12]    // 3
    76c:   b9400fe0        ldr     w0, [sp, #12]    // 5
    770:   910043ff        add     sp, sp, #0x10    // 5
    774:   d65f03c0        ret                      // 5
```

while loops

- Mapping C while loops to assembly code

```c
int pow2(int n) {
    int result = 1;
    while (n > 0) {
        result = result * 2;
        n = n - 1;
    }
    return result;
}
```

```
0000000000400584 <pow2>:
    400584:       d10043ff        sub     sp, sp, #0x10          // 1
    400588:       b9000fe0        str     w0, [sp, #12]          // 1
    40058c:       52800028        mov     w8, #0x1               // 2
    400590:       b9000be8        str     w8, [sp, #8]           // 2
    400594:       b9400fe8        ldr     w8, [sp, #12]          // 3
    400598:       7100011f        cmp     w8, #0x0               // 3
    40059c:       37000128        b.le    4005c0 <pow2+0x3c>     // 3
    4005a0:       b9400be8        ldr     w8, [sp, #8]           // 4
    4005a4:       52800049        mov     w9, #0x2               // 4
    4005a8:       1b097d08        mul     w8, w8, w9             // 4
    4005ac:       b9000be8        str     w8, [sp, #8]           // 4
    4005b0:       b9400fe8        ldr     w8, [sp, #12]          // 5
    4005b4:       71000508        subs    w8, w8, #0x1           // 5
    4005b8:       b9000fe8        str     w8, [sp, #12]          // 5
    4005bc:       17fffff5        b       400594 <pow2+0x10>     // 6
    4005c0:       b9400be0        ldr     w0, [sp, #8]           // 7
    4005c4:       910043ff        add     sp, sp, #0x10          // 7
    4005c8:       d65f03c0        ret                            // 7
```

- Goto form

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

Loop duality

- Q2: *Write a function called `tally_while` that is semantically equivalent to the function below, but uses a `while` loop instead of a `for` loop.*

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

```c
int tally_while(int y) {
    int result = 0;
    int i = 1;
    while (i <= y) {
        result = result + i;
        i++;
    }
    return result;
}
```

- Assembly code for both functions is equivalent

```
00000000004005c0 <tally_for>:
    4005c0: d10043ff    sub     sp, sp, #0x10
    4005c4: b9000fe0    str     w0, [sp,#12]
    4005c8: b9000bff    str     wzr, [sp,#8]
    4005cc: 320003e8    mov     w8, #0x1
    4005d0: b90007e8    str     w8, [sp,#4]
    4005d4: b94007e8    ldr     w8, [sp,#4]
    4005d8: b9400fe9    ldr     w9, [sp,#12]
    4005dc: 6b09011f    cmp     w8, w9
    4005e0: 5400012c    b.gt    400604
<tally_for+0x44>
    4005e4: b9400be8    ldr     w8, [sp,#8]
    4005e8: b94007e9    ldr     w9, [sp,#4]
    4005ec: 0b090108    add     w8, w8, w9
    4005f0: b9000be8    str     w8, [sp,#8]
    4005f4: b94007e8    ldr     w8, [sp,#4]
    4005f8: 11000508    add     w8, w8, #0x1
    4005fc: b90007e8    str     w8, [sp,#4]
    400600: 17fffff5    b       4005d4
<tally_for+0x14>
    400604: b9400be0    ldr     w0, [sp,#8]
    400608: 910043ff    add     sp, sp, #0x10
    40060c: d65f03c0    ret
```

```
0000000000400610 <tally_while>:
    400610: d10043ff    sub     sp, sp, #0x10
    400614: b9000fe0    str     w0, [sp,#12]
    400618: b9000bff    str     wzr, [sp,#8]
    40061c: 320003e8    mov     w8, #0x1
    400620: b90007e8    str     w8, [sp,#4]
    400624: b94007e8    ldr     w8, [sp,#4]
    400628: b9400fe9    ldr     w9, [sp,#12]
    40062c: 6b09011f    cmp     w8, w9
    400630: 5400012c    b.gt    400654
<tally_while+0x44>
    400634: b9400be8    ldr     w8, [sp,#8]
    400638: b94007e9    ldr     w9, [sp,#4]
    40063c: 0b090108    add     w8, w8, w9
    400640: b9000be8    str     w8, [sp,#8]
    400644: b94007e8    ldr     w8, [sp,#4]
    400648: 11000508    add     w8, w8, #0x1
    40064c: b90007e8    str     w8, [sp,#4]
    400650: 17fffff5    b       400624
<tally_while+0x14>
    400654: b9400be0    ldr     w0, [sp,#8]
    400658: 910043ff    add     sp, sp, #0x10
    40065c: d65f03c0    ret
```

- Q3: *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
1  int powi(int m, int n) {
2      int result = 1;
3      for (int i = 0; i < n; i++) {
4          result *= m;
5      }
6      return result;
7  }
```

```
0000000000400544 <powi>:
    400544:    d10043ff     sub    sp, sp, #0x10        // 1
    400548:    b9000fe0     str    w0, [sp, #12]        // 1
    40054c:    b9000be1     str    w1, [sp, #8]         // 1
    400550:    52800028     mov    w8, #0x1             // 2
    400554:    b90007e8     str    w8, [sp, #4]         // 2
    400558:    b90003ff     str    wzr, [sp]            // 3
    40055c:    b94003e8     ldr    w8, [sp]             // 3
    400560:    b9400be9     ldr    w9, [sp, #8]         // 3
    400564:    6b09011f     cmp    w8, w9               // 3
    400568:    5400012a     b.ge    40058c <powi+0x48>  // 3
    40056c:    b9400fe8     ldr    w8, [sp, #12]        // 4
    400570:    b94007e9     ldr    w9, [sp, #4]         // 4
    400574:    1b087d28     mul    w8, w9, w8           // 4
    400578:    b90007e8     str    w8, [sp, #4]         // 4
    40057c:    b94003e8     ldr    w8, [sp]             // 3
    400580:    11000508     add    w8, w8, #0x1         // 3
    400584:    b90003e8     str    w8, [sp]             // 3
    400588:    17fffff5     b    40055c <powi+0x18>     // 5
    40058c:    b94007e0     ldr    w0, [sp, #4]         // 6
    400590:    910043ff     add    sp, sp, #0x10        // 6
    400594:    d65f03c0     ret                         // 6
```

- *How would the correspondence between assembly code and C code be different if `powi` used a while loop?*

```
1  int powi(int m, int n) {
2      int result = 1;
3A      int i = 0;
4      while (i < n) {
5          result *= m;
6          i++;
7      }
8      return result;
9  }
```

```
0000000000400544 <powi>:
    400544:    d10043ff     sub    sp, sp, #0x10        // 1
    40054c:    b9000fe0     str    w0, [sp, #12]        // 1
    400550:    b9000be1     str    w1, [sp, #8]         // 1
    400548:    52800028     mov    w8, #0x1             // 2
    400554:    b90007e8     str    w8, [sp, #4]         // 2
    400558:    b90003ff     str    wzr, [sp]            // 3
    40055c:    b94003e8     ldr    w8, [sp]             // 4
    400560:    b9400be9     ldr    w9, [sp, #8]         // 4
    400564:    6b09011f     cmp    w8, w9               // 4
    400568:    5400012a     b.ge    40058c <powi+0x48>  // 4
    40056c:    b9400fe8     ldr    w8, [sp, #12]        // 5
    400570:    b94007e9     ldr    w9, [sp, #4]         // 5
    400574:    1b087d28     mul    w8, w9, w8           // 5
    400578:    b90007e8     str    w8, [sp, #4]         // 5
    40057c:    b94003e8     ldr    w8, [sp]             // 6
    400580:    11000508     add    w8, w8, #0x1         // 6
    400584:    b90003e8     str    w8, [sp]             // 6
    400588:    17fffff5     b    40055c <powi+0x18>     // 7
    40058c:    b94007e0     ldr    w0, [sp, #4]         // 8
    400590:    910043ff     add    sp, sp, #0x10        // 8
    400594:    d65f03c0     ret                         // 8
```

Conditionals and loops

- Q4: *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            // 1
    400588:    b9000fe0     str    w0, [sp, #12]            // 1
    40058c:    b9000bff     str    wzr, [sp, #8]            // 2
    400590:    b9400fe8     ldr    w8, [sp, #12]            // 3
    400594:    34000168     cbz    w8, 4005c0 <onebits+0x3c>// 3
    400598:    b9400fe8     ldr    w8, [sp, #12]            // 4
    40059c:    12000108     and    w8, w8, #0x1             // 4
    4005a0:    34000088     cbz    w8, 4005b0 <onebits+0x2c>// 4
    4005a4:    b9400be8     ldr    w8, [sp, #8]             // 5
    4005a8:    11000508     add    w8, w8, #0x1             // 5
    4005ac:    b9000be8     str    w8, [sp, #8]             // 5
    4005b0:    b9400fe8     ldr    w8, [sp, #12]            // 7
    4005b4:    53017d08     lsr    w8, w8, #1               // 7
    4005b8:    b9000fe8     str    w8, [sp, #12]            // 7
    4005bc:    17fffff5     b    400590 <onebits+0xc>       // 8
    4005c0:    b9400be0     ldr    w0, [sp, #8]             // 9
    4005c4:    910043ff     add    sp, sp, #0x10            // 9
    4005c8:    d65f03c0     ret                             // 9
```

- Q5: *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`.*

```c
int onebits_goto(unsigned int num) {
    int ones = 0;
LOOP_TOP:
    if (num == 0)
        goto LOOP_BOTTOM;
    if (num & 0b1 == 0)
        goto IF_END;
    ones++;
IF_END:
    num = num >> 1;
    goto LOOP_TOP;
LOOP_BOTTOM:
    return ones;
}
```