Assembly: functions

COSC 208, Introduction to Computer Systems, 2022-03-31

Announcements

- Project 2 due Thursday at 11pm

Outline

- Warm-up
- Functions

Warm-up

- Q1: *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 log10i(int num) {
    int result = -1;
    if (num > 0) {
        result = 0;
        while (num >= 10) {
            result = result + 1;
            num = num / 10;
        }
    }
    return result;
}
```

```
0000000000400584 <log10i>:
    400584:    d10043ff     sub    sp, sp, #0x10    // 1
    400588:    b9000fe0     str    w0, [sp, #12]    // 1
    40058c:    12800008     mov    w8, #0xffffffff  // 2
    400590:    b9000be8     str    w8, [sp, #8]     // 2
    400594:    b9400fe8     ldr    w8, [sp, #12]    // 3
    400598:    7100011f     cmp    w8, #0x0         // 3
    40059c:    370001a8     b.le   4005d0 <log10i+0x4c> // 3
    4005a0:    b9000bff     str    wzr, [sp, #8]    // 4
    4005a4:    b9400fe8     ldr    w8, [sp, #12]    // 5
    4005a8:    7100291f     cmp    w8, #0xa         // 5
    4005ac:    5400012b     b.lt   4005d0 <log10i+0x4c> // 5
    4005b0:    b9400be8     ldr    w8, [sp, #8]     // 6
    4005b4:    11000508     add    w8, w8, #0x1     // 6
    4005b8:    b9000be8     str    w8, [sp, #8]     // 6
    4005bc:    b9400fe8     ldr    w8, [sp, #12]    // 7
    4005c0:    52800149     mov    w9, #0xa         // 7
    4005c4:    1ac90d08     sdiv    w8, w8, w9      // 7
    4005c8:    b9000fe8     str    w8, [sp, #12]    // 7
    4005cc:    17fffff6     b    4005a4 <log10i+0x20> // 8
    4005d0:    b9400be0     ldr    w0, [sp, #8]     // 10
    4005d4:    910043ff     add    sp, sp, #0x10    // 10
    4005d8:    d65f03c0     ret                     // 10
```

- Q2: *Write a function called `log10i_goto` that behaves the same as `log10i` but matches the structure of the assembly code that will be generated for `log10i`.*

```c
int log10i_goto(int num) {
    int result = -1;
    if (num <= 0)
        goto IF_END;
    result = 0;
LOOP_TOP:
    if (num < 10)
        goto LOOP_END:
    result = result + 1;
    num = num / 10;
    goto LOOP_TOP;
LOOP_END:
IF_END:
    return result;
}
```

Functions

- Calling conventions
  - *In which registers are parameters stored?* — `x0`/`w0`, `x1`/`w1`, `x2`/`w2`, ...
  - *In which register is the return value stored?* — `x0`/`w0`
  - Return address stored in `x30`
  - Caller's return address stored in callee's stack frame
- Use `bl` (branch & link) to make a function call
  - Store `pc+4` in `x30`
  - Update `pc` to specified code address
- Mapping C functions to assembly code

```c
int multiply(int a, int b) {
    int c = a * b;
    return c;
}
int volume(int x, int y, int z) {
    int w = multiply(x, y);
    w = multiply(w, z);
    return w;
}
```

```
0000000000400544 <multiply>:
    400544:    d10043ff     sub    sp, sp, #0x10    // 1
    400548:    b9000fe0     str    w0, [sp, #12]    // 1
    40054c:    b9000be1     str    w1, [sp, #8]     // 1
    400550:    b9400fe8     ldr    w8, [sp, #12]    // 2
    400554:    b9400be9     ldr    w9, [sp, #8]     // 2
    400558:    1b097d08     mul    w8, w8, w9       // 2
    40055c:    b90007e8     str    w8, [sp, #4]     // 2
    400560:    b94007e0     ldr    w0, [sp, #4]     // 3
    400564:    910043ff     add    sp, sp, #0x10    // 3
    400568:    d65f03c0     ret                     // 3
```

```
000000000040056c <volume>:
    40056c:    d10083ff     sub    sp, sp, #0x20    // 5
    400570:    f9000bfe     str    x30, [sp, #16]   // 5
    400574:    b9000fe0     str    w0, [sp, #12]    // 5
    400578:    b9000be1     str    w1, [sp, #8]     // 5
    40057c:    b90007e2     str    w2, [sp, #4]     // 5
    400580:    b9400fe0     ldr    w0, [sp, #12]    // 6
    400584:    b9400be1     ldr    w1, [sp, #8]     // 6
    400588:    97ffffef     bl    400544 <multiply> // 6
    40058c:    b90003e0     str    w0, [sp]         // 6
    400590:    b94003e0     ldr    w0, [sp]         // 7
    400594:    b94007e1     ldr    w1, [sp, #4]     // 7
    400598:    97ffffeb     bl    400544 <multiply> // 7
    40059c:    b90003e0     str    w0, [sp]         // 7
    4005a0:    b94003e0     ldr    w0, [sp]         // 8
    4005a4:    f9400bfe     ldr    x30, [sp, #16]   // 8
    4005a8:    910083ff     add    sp, sp, #0x20    // 8
    4005ac:    d65f03c0     ret                     // 8
```

- Q3: *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.*

```
10  int main() {
11      int n = volume(1, 3, 5);
12      return n;
13  }
```

```
00000000004005b0 <main>:
    4005b0:    d10083ff     sub    sp, sp, #0x20    // 10
    4005b4:    f9000bfe     str    x30, [sp, #16]   // 10
    4005b8:    52800020     mov    w0, #0x1         // 11
    4005bc:    52800061     mov    w1, #0x3         // 11
    4005c0:    528000a2     mov    w2, #0x5         // 11
    4005c4:    97ffffe9     bl    40056c <volume>   // 11
    4005c8:    b9000be0     str    w0, [sp, #8]     // 11
    4005cc:    b9400be0     ldr    w0, [sp, #8]     // 12
    4005d0:    f9400bfe     ldr    x30, [sp, #16]   // 12
    4005d4:    910083ff     add    sp, sp, #0x20    // 12
    4005d8:    d65f03c0     ret                     // 12
```

- Q4: Tracing assembly code with functions: Assume the registers have the following initial values:
  - `pc` = `0x4005b0`
  - `sp` = `0xFF0`
  - `x30` = `0x4005dc`

Practice with functions

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 sum(int a, int b) {
    int c = a + b;
    return c;
}
int triple(int u, int v) {
    int r = 3;
    int s = sum(u, v);
    int t = s * r;
    return t;
 }
```

```
0000000000400544 <sum>:
    400544:    d10043ff     sub    sp, sp, #0x10   //
    400548:    b9000fe0     str    w0, [sp, #12]   //
    40054c:    b9000be1     str    w1, [sp, #8]    //
    400550:    b9400fe8     ldr    w8, [sp, #12]   //
    400554:    b9400be9     ldr    w9, [sp, #8]    //
    400558:    0b090108     add    w8, w8, w9      //
    40055c:    b90007e8     str    w8, [sp, #4]    //
    400560:    b94007e0     ldr    w0, [sp, #4]    //
    400564:    910043ff     add    sp, sp, #0x10   //
    400568:    d65f03c0     ret                    //
000000000040056c <triple>:
    40056c:    d100c3ff     sub    sp, sp, #0x30   //
    400570:    f90013fe     str    x30, [sp, #32]  //
    400574:    b9001fe0     str    w0, [sp, #28]   //
    400578:    b9001be1     str    w1, [sp, #24]   //
    40057c:    52800068     mov    w8, #0x3        //
    400580:    b90017e8     str    w8, [sp, #20]   //
    400584:    b9401fe0     ldr    w0, [sp, #28]   //
    400588:    b9401be1     ldr    w1, [sp, #24]   //
    40058c:    97fffee     bl     400544 <sum>    //
    400590:    b90013e0     str    w0, [sp, #16]   //
    400594:    b94013e8     ldr    w8, [sp, #16]   //
    400598:    b94017e9     ldr    w9, [sp, #20]   //
    40059c:    1b097d08     mul    w8, w8, w9      //
    4005a0:    b9000fe8     str    w8, [sp, #12]   //
    4005a4:    b9400fe0     ldr    w0, [sp, #12]   //
    4005a8:    f94013fe     ldr    x30, [sp, #32]  //
    4005ac:    9100c3ff     add    sp, sp, #0x30   //
    4005b0:    d65f03c0     ret                    //
```

```
00000000004005b4 <main>:
    4005b4:    d10083ff     sub    sp, sp, #0x20   //
    4005b8:    f9000bfe     str    x30, [sp, #16]  //
    4005bc:    52800048     mov    w8, #0x2        //
    4005c0:    b9000be8     str    w8, [sp, #8]    //
    4005c4:    52800069     mov    w9, #0x3        //
    4005c8:    b90007e9     str    w9, [sp, #4]    //
    4005cc:    b9400be0     ldr    w0, [sp, #8]    //
    4005d0:    b94007e1     ldr    w1, [sp, #4]    //
    4005d4:    97ffffe5     bl     40056c <triple> //
    4005d8:    b90003e0     str    w0, [sp]        //
    4005dc:    b94003e0     ldr    w0, [sp]        //
    4005e0:    f9400bfe     ldr    x30, [sp, #16]  //
    4005e4:    910083ff     add    sp, sp, #0x20   //
    4005e8:    d65f03c0     ret                    //
```

```
11  int main() {
12      int x = 2;
13      int y = 3;
14      int z = triple(x, y);
15      return z;
16  }
```

Q6: *Draw the contents of the stack and registers just prior to the execution of the second-to-last assembly instruction in `sum`. Assume the initial value in the `sp` register is `0xF80`.*

- `pc` = `0x4005b4`
- `sp` = `0xF80`
- `x30` = `0x400488`

Assembly: functions

COSC 208, Introduction to Computer Systems, 2022-03-31

Announcements

- Project 2 due tonight

Warm-up

Q1: *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 log10i(int num) {
    int result = -1;
    if (num > 0) {
        result = 0;
        while (num >= 10) {
            result = result + 1;
            num = num / 10;
        }
    }
    return result;
}
```

```
0000000000400584 <log10i>:
    400584:    d10043ff     sub    sp, sp, #0x10
    400588:    b9000fe0     str    w0, [sp, #12]
    40058c:    12800008     mov    w8, #0xffffffff
    400590:    b9000be8     str    w8, [sp, #8]
    400594:    b9400fe8     ldr    w8, [sp, #12]
    400598:    7100011f     cmp    w8, #0x0
    40059c:    370001a8     b.le   4005d0 <log10i+0x4c>
    4005a0:    b9000bff     str    wzr, [sp, #8]
    4005a4:    b9400fe8     ldr    w8, [sp, #12]
    4005a8:    7100291f     cmp    w8, #0xa
    4005ac:    5400012b     b.lt   4005d0 <log10i+0x4c>
    4005b0:    b9400be8     ldr    w8, [sp, #8]
    4005b4:    11000508     add    w8, w8, #0x1
    4005b8:    b9000be8     str    w8, [sp, #8]
    4005bc:    b9400fe8     ldr    w8, [sp, #12]
    4005c0:    52800149     mov    w9, #0xa
    4005c4:    1ac90d08     sdiv    w8, w8, w9
    4005c8:    b9000fe8     str    w8, [sp, #12]
    4005cc:    17fffff6     b    4005a4 <log10i+0x20>
    4005d0:    b9400be0     ldr    w0, [sp, #8]
    4005d4:    910043ff     add    sp, sp, #0x10
    4005d8:    d65f03c0     ret
```

Q2: *Write a function called `log10i_goto` that behaves the same as `log10i` but matches the structure of the assembly code that will be generated for `log10i`.* On your own paper...

Functions

Mapping C functions to assembly code

```c
int multiply(int a, int b) {
    int c = a * b;
    return c;
}
int volume(int x, int y, int z) {
    int w = multiply(x, y);
    w = multiply(w, z);
    return w;
}
```

```
0000000000400544 <multiply>:
    400544:    d10043ff     sub    sp, sp, #0x10       //
    400548:    b9000fe0     str    w0, [sp, #12]       //
    40054c:    b9000be1     str    w1, [sp, #8]        //
    400550:    b9400fe8     ldr    w8, [sp, #12]       //
    400554:    b9400be9     ldr    w9, [sp, #8]        //
    400558:    1b097d08     mul    w8, w8, w9          //
    40055c:    b90007e8     str    w8, [sp, #4]        //
    400560:    b94007e0     ldr    w0, [sp, #4]        //
    400564:    910043ff     add    sp, sp, #0x10       //
    400568:    d65f03c0     ret                        //
000000000040056c <volume>:
    40056c:    d10083ff     sub    sp, sp, #0x20       //
    400570:    f9000bfe     str    x30, [sp, #16]      //
    400574:    b9000fe0     str    w0, [sp, #12]       //
    400578:    b9000be1     str    w1, [sp, #8]        //
    40057c:    b90007e2     str    w2, [sp, #4]        //
    400580:    b9400fe0     ldr    w0, [sp, #12]       //
    400584:    b9400be1     ldr    w1, [sp, #8]        //
    400588:    97fffef     bl     400544 <multiply>   //
    40058c:    b90003e0     str    w0, [sp]            //
    400590:    b94003e0     ldr    w0, [sp]            //
    400594:    b94007e1     ldr    w1, [sp, #4]        //
    400598:    97fffeb     bl     400544 <multiply>   //
    40059c:    b90003e0     str    w0, [sp]            //
    4005a0:    b94003e0     ldr    w0, [sp]            //
    4005a4:    f9400bfe     ldr    x30, [sp, #16]      //
    4005a8:    910083ff     add    sp, sp, #0x20       //
    4005ac:    d65f03c0     ret                        //
```

Q3: *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.*

```
10  int main() {
11      int n = volume(1, 3, 5);
12      return n;
13  }
```

```
00000000004005b0 <main>:
    4005b0:    d10083ff     sub    sp, sp, #0x20       //
    4005b4:    f9000bfe     str    x30, [sp, #16]      //
    4005b8:    52800020     mov    w0, #0x1            //
    4005bc:    52800061     mov    w1, #0x3            //
    4005c0:    528000a2     mov    w2, #0x5            //
    4005c4:    97ffffe9     bl     40056c <volume>     //
    4005c8:    b9000be0     str    w0, [sp, #8]        //
    4005cc:    b9400be0     ldr    w0, [sp, #8]        //
    4005d0:    f9400bfe     ldr    x30, [sp, #16]      //
    4005d4:    910083ff     add    sp, sp, #0x20       //
    4005d8:    d65f03c0     ret                        //
```

Q4: *Tracing assembly code with functions. Assume the registers have the following initial values:*

- `pc` = `0x4005b0`
- `sp` = `0xFF0`
- `x30` = `0x4005dc`

Practice with functions

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 sum(int a, int b) {
    int c = a + b;
    return c;
}
int triple(int u, int v) {
    int r = 3;
    int s = sum(u, v);
    int t = s * r;
    return t;
 }
```

```
0000000000400544 <sum>:
    400544:    d10043ff     sub    sp, sp, #0x10   //
    400548:    b9000fe0     str    w0, [sp, #12]   //
    40054c:    b9000be1     str    w1, [sp, #8]    //
    400550:    b9400fe8     ldr    w8, [sp, #12]   //
    400554:    b9400be9     ldr    w9, [sp, #8]    //
    400558:    0b090108     add    w8, w8, w9      //
    40055c:    b90007e8     str    w8, [sp, #4]    //
    400560:    b94007e0     ldr    w0, [sp, #4]    //
    400564:    910043ff     add    sp, sp, #0x10   //
    400568:    d65f03c0     ret                    //
000000000040056c <triple>:
    40056c:    d100c3ff     sub    sp, sp, #0x30   //
    400570:    f90013fe     str    x30, [sp, #32]  //
    400574:    b9001fe0     str    w0, [sp, #28]   //
    400578:    b9001be1     str    w1, [sp, #24]   //
    40057c:    52800068     mov    w8, #0x3        //
    400580:    b90017e8     str    w8, [sp, #20]   //
    400584:    b9401fe0     ldr    w0, [sp, #28]   //
    400588:    b9401be1     ldr    w1, [sp, #24]   //
    40058c:    97fffee     bl     400544 <sum>    //
    400590:    b90013e0     str    w0, [sp, #16]   //
    400594:    b94013e8     ldr    w8, [sp, #16]   //
    400598:    b94017e9     ldr    w9, [sp, #20]   //
    40059c:    1b097d08     mul    w8, w8, w9      //
    4005a0:    b9000fe8     str    w8, [sp, #12]   //
    4005a4:    b9400fe0     ldr    w0, [sp, #12]   //
    4005a8:    f94013fe     ldr    x30, [sp, #32]  //
    4005ac:    9100c3ff     add    sp, sp, #0x30   //
    4005b0:    d65f03c0     ret                    //
```

```
00000000004005b4 <main>:
    4005b4:    d10083ff     sub    sp, sp, #0x20   //
    4005b8:    f9000bfe     str    x30, [sp, #16]  //
    4005bc:    52800048     mov    w8, #0x2        //
    4005c0:    b9000be8     str    w8, [sp, #8]    //
    4005c4:    52800069     mov    w9, #0x3        //
    4005c8:    b90007e9     str    w9, [sp, #4]    //
    4005cc:    b9400be0     ldr    w0, [sp, #8]    //
    4005d0:    b94007e1     ldr    w1, [sp, #4]    //
    4005d4:    97ffffe5     bl     40056c <triple> //
    4005d8:    b90003e0     str    w0, [sp]        //
    4005dc:    b94003e0     ldr    w0, [sp]        //
    4005e0:    f9400bfe     ldr    x30, [sp, #16]  //
    4005e4:    910083ff     add    sp, sp, #0x20   //
    4005e8:    d65f03c0     ret                    //
```

```
11  int main() {
12      int x = 2;
13      int y = 3;
14      int z = triple(x, y);
15      return z;
16  }
```

Q6: *Draw the contents of the stack and registers just prior to the execution of the second-to-last assembly instruction in* `sum`*. Assume the initial value in the* `sp` *register is* `0xF80`*.*