Assembly: functions

COSC 208, Introduction to Computer Systems, 2021-10-18

Announcements

- Project 2 Part 1 due this Thursday at 11pm

Warm-up

- Q1: *The following C code was compiled into assembly. 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 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;
}
```

```
000000000000071c <multiply>:
    71c:    d10083ff    sub sp, sp, #0x20           // Line 1
    720:    b9000fe0    str w0, [sp, #12]           //   |
    724:    b9000be1    str w1, [sp, #8]            //   V
    728:    b9400fe1    ldr w1, [sp, #12]           // Line 2
    72c:    b9400be0    ldr w0, [sp, #8]            //   |
    730:    1b007c20    mul w0, w1, w0              //   |
    734:    b9001fe0    str w0, [sp, #28]           //   V
    738:    b9401fe0    ldr w0, [sp, #28]           // Line 3
    73c:    910083ff    add sp, sp, #0x20           //   |
    740:    d65f03c0    ret                         //   V
0000000000000744 <volume>:
    744:    a9bd7bfd    stp x29, x30, [sp, #-48]!   // Line 5
    748:    910003fd    mov x29, sp                 //   |
    74c:    b9001fe0    str w0, [sp, #28]           //   |
    750:    b9001be1    str w1, [sp, #24]           //   |
    754:    b90017e2    str w2, [sp, #20]           //   V
    758:    b9401be1    ldr w1, [sp, #24]           // Line 6
    75c:    b9401fe0    ldr w0, [sp, #28]           //   |
    760:    97ffffef    bl  71c <multiply>          //   |
    764:    b9002fe0    str w0, [sp, #44]           //   V
    768:    b94017e1    ldr w1, [sp, #20]           // Line 7
    76c:    b9402fe0    ldr w0, [sp, #44]           //   |
    770:    97ffffeb    bl  71c <multiply>          //   |
    774:    b9002fe0    str w0, [sp, #44]           //   V
    778:    b9402fe0    ldr w0, [sp, #44]           // Line 8
    77c:    a8c37bfd    ldp x29, x30, [sp], #48     //   |
    780:    d65f03c0    ret                         //   V
```

Functions

- Noteworthy instructions
  - `stp` --- update `sp`, then store values at `sp` and `sp+8`
  - `ldp` --- load values at `sp` and `sp+8`, then update `sp`
  - `bl` --- "branch with link"; store `pc+4` in `x30`, then update `pc` to specified code address
- Q2: *Translate each assembly instruction into semantically equivalent C code. For example `stp x29, x30, [sp, #-48]!` translates to:*

```
sp = sp - 48;
*sp = x29;
*(sp + 8) = x30;
```

```
0000000000000744 <volume>:                          // sp = sp - 48;
    744:    a9bd7bfd    stp x29, x30, [sp, #-48]!   // *sp = x29; *(sp + 8) = x30
    748:    910003fd    mov x29, sp                 // x29 = sp
    74c:    b9001fe0    str w0, [sp, #28]           // *(sp + 28) = w0
    750:    b9001be1    str w1, [sp, #24]           // *(sp + 24) = w1
    754:    b90017e2    str w2, [sp, #20]           // *(sp + 20) = w2
    758:    b9401be1    ldr w1, [sp, #24]           // w1 = *(sp + 24)
    75c:    b9401fe0    ldr w0, [sp, #28]           // w0 = *(sp + 28)
    760:    97ffffef    bl  71c <multiply>          // x30 = pc + 4; pc = 71c
    764:    b9002fe0    str w0, [sp, #44]           // *(sp + 44) = w0
    768:    b94017e1    ldr w1, [sp, #20]           // w1 = *(sp + 20)
    76c:    b9402fe0    ldr w0, [sp, #44]           // w0 = *(sp + 44)
    770:    97ffffeb    bl  71c <multiply>          // x30 = pc + 4; pc = 71c
    774:    b9002fe0    str w0, [sp, #44]           // *(sp + 44) = w0
    778:    b9402fe0    ldr w0, [sp, #44]           // w0 = *(sp + 44)
    77c:    a8c37bfd    ldp x29, x30, [sp], #48     // x29 = *sp; x30 = *(sp + 8);
                                                    // sp = sp + 48
    780:    d65f03c0    ret                         // pc = x30
```

- 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 stack pointer stored in `x29`
  - Caller's stack pointer stored at the top of callee's stack frame
  - Caller's return address stored 8 bytes below the top of callee's stack frame
- Q3: *Tracing the assembly code.*

Extra practice

- Q4: *The following C code was compiled into assembly. 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 three(int x) {
    return x + 3;
}
int two(int y) {
    return three(y) + 2;
}
int one(int z) {
    return two(z) + 1;
}
```

```
000000000000071c <three>:
    71c:    d10043ff    sub sp, sp, #0x10           // Line 1
    720:    b9000fe0    str w0, [sp, #12]           //   V
    724:    b9400fe0    ldr w0, [sp, #12]           // Line 2
    728:    11000c00    add w0, w0, #0x3            //   |
    72c:    910043ff    add sp, sp, #0x10           //   |
    730:    d65f03c0    ret                         //   V
0000000000000734 <two>:
    734:    a9be7bfd    stp x29, x30, [sp, #-32]!   // Line 4
    738:    910003fd    mov x29, sp                 //   |
    73c:    b9001fe0    str w0, [sp, #28]           //   V
    740:    b9401fe0    ldr w0, [sp, #28]           // Line 5
    744:    97fffff6    bl  71c <three>             //   |
    748:    11000800    add w0, w0, #0x2            //   |
    74c:    a8c27bfd    ldp x29, x30, [sp], #32     //   |
    750:    d65f03c0    ret                         //   V
0000000000000754 <one>:
    754:    a9be7bfd    stp x29, x30, [sp, #-32]!   // Line 7
    758:    910003fd    mov x29, sp                 //   |
    75c:    b9001fe0    str w0, [sp, #28]           //   V
    760:    b9401fe0    ldr w0, [sp, #28]           // Line 8
    764:    97fffff4    bl  734 <two>               //   |
    768:    11000400    add w0, w0, #0x1            //   |
    76c:    a8c27bfd    ldp x29, x30, [sp], #32     //   |
    770:    d65f03c0    ret                         //   V
```

- Q5: *Translate each assembly instruction into semantically equivalent C code.*

```
000000000000071c <three>:
    71c:    d10043ff    sub sp, sp, #0x10           // sp = sp - 0x10
    720:    b9000fe0    str w0, [sp, #12]           // *(sp + 12) = w0
    724:    b9400fe0    ldr w0, [sp, #12]           // w0 = *(sp + 12)
    728:    11000c00    add w0, w0, #0x3            // w0 = w0 + 0x3
    72c:    910043ff    add sp, sp, #0x10           // sp = sp + 0x10
    730:    d65f03c0    ret                         // pc = x30
0000000000000734 <two>:
    734:    a9be7bfd    stp x29, x30, [sp, #-32]!   // sp = sp - 32;
                                                    // *sp = x29; *(sp + 8) = x30
    738:    910003fd    mov x29, sp                 // x29 = sp
    73c:    b9001fe0    str w0, [sp, #28]           // *(sp + 28) = w0
    740:    b9401fe0    ldr w0, [sp, #28]           // w0 = *(sp + 28)
    744:    97fffff6    bl  71c <three>             // x30 = pc + 4; pc = 71c
    748:    11000800    add w0, w0, #0x2            // w0 = w0 + 0x2
    74c:    a8c27bfd    ldp x29, x30, [sp], #32     // x29 = *sp; x30 = *(sp + 8);
                                                    // sp = sp + 32
    750:    d65f03c0    ret                         // pc = x30
0000000000000754 <one>:
    754:    a9be7bfd    stp x29, x30, [sp, #-32]!   // sp = sp - 32;
                                                    // *sp = x29; *(sp + 8) = x30
    758:    910003fd    mov x29, sp                 // x29 = sp
    75c:    b9001fe0    str w0, [sp, #28]           // *(sp + 28) = w0
    760:    b9401fe0    ldr w0, [sp, #28]           // w0 = *(sp + 28)
    764:    97fffff4    bl  734 <two>               // x30 = pc + 4; pc = 734
    768:    11000400    add w0, w0, #0x1            // w0 = w0 + 0x1
    76c:    a8c27bfd    ldp x29, x30, [sp], #32     // x29 = *sp; x30 = *(sp + 8);
                                                    // sp = sp + 32
    770:    d65f03c0    ret                         // pc = x30
```

- Q6: *Draw the contents of the registers and stack immediately before executing the instruction at address `72c`. Assume registers have the following initial values: `sp=0xFD0`, `pc=0x754`, `w0=0`, `x29=0xFF0`, `x30=0x784`*