Assembly: Tracing; conditionals intro

COSC 208, Introduction to Computer Systems, 2021-10-13

Outline

- Warm-up
- Tracing assembly code
- Project 2 overview
- Arithmetic operations
- Conditionals

Warm-up

- Q1: *Write the C code equivalent for each line of assembly, treating registers as if they were variable names. For example, the C code equivalent for* `sub sp, sp, #0x20` *is* `sp = sp - 0x20`

```
0000000000400544 <sum>:
    400544: d10083ff    sub sp, sp, #0x20   sp = sp - 0x20
    400548: b9001fe0    str w0, [sp, #28]   *(sp + 28) = w0
    40054c: f9000be1    str x1, [sp, #16]   *(sp + 16) = x1
    400550: f9400be8    ldr x8, [sp, #16]   x8 = *(sp + 16)
    400554: b9400109    ldr w9, [x8]        w9 = *x8
    400558: b9000fe9    str w9, [sp, #12]   *(sp + 12) = w9
    40055c: b9401fe9    ldr w9, [sp, #28]   w9 = *(sp + 28)
    400560: b9400fea    ldr w10, [sp, #12]  w10 = *(sp + 12)
    400564: 0b0a0129    add w9, w9, w10     w9 = w9 + w10
    400568: b9000be9    str w9, [sp, #8]    *(sp + 8) = w9
    40056c: b9400be0    ldr w0, [sp, #8]    w0 = *(sp + 8)
    400570: 910083ff    add sp, sp, #0x20   sp = sp + 0x20
```

Tracing assembly code

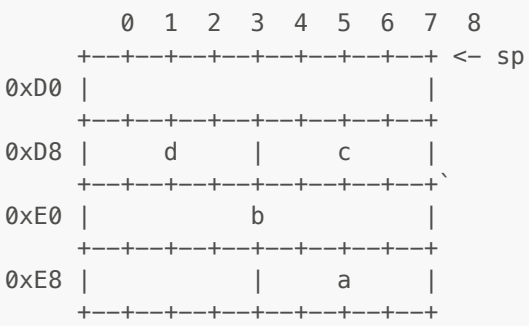
- C code

```
1  int sum(int a, int *b) {
2     int c = *b;
3     int d = a + c;
4     return d;
5  }
```

- Assembly code

```
0000000000400544 <sum>:
    400544: d10083ff    sub sp, sp, #0x20   // Line 1
    400548: b9001fe0    str w0, [sp, #28]   //   |
    40054c: f9000be1    str x1, [sp, #16]   //   V
    400550: f9400be8    ldr x8, [sp, #16]   // Line 2
    400554: b9400109    ldr w9, [x8]        //   |
    400558: b9000fe9    str w9, [sp, #12]   //   V
    40055c: b9401fe9    ldr w9, [sp, #28]   // Line 3
    400560: b9400fea    ldr w10, [sp, #12]  //   |
    400564: 0b0a0129    add w9, w9, w10     //   |
    400568: b9000be9    str w9, [sp, #8]    //   V
    40056c: b9400be0    ldr w0, [sp, #8]    // Line 4
    400570: 910083ff    add sp, sp, #0x20   //   V
```

- Stack (before executing last assembly instruction; assume `sp = 0xF0` initially)



Conditionals

- Q6: *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.*

```
1  int divide(int numerator, int denominator) {
2      int result = -1;
3      result = numerator / denominator;
4      return result;
5  }
```

```
0000000000400544 <divide>:
    400544: d10043ff    sub sp, sp, #0x10       // Line 1
    400548: 12800008    mov w8, #0xffffffff     // Line 2
    40054c: b9000fe0    str w0, [sp, #12]       // Line 1
    400550: b9000be1    str w1, [sp, #8]        //   V
    400554: b90007e8    str w8, [sp, #4]        // Line 2
    400558: b9400fe8    ldr w8, [sp, #12]       // Line 3
    40055c: b9400be9    ldr w9, [sp, #8]        //   |
    400560: 1ac90d08    sdiv    w8, w8, w9      //   |
    400564: b90007e8    str w8, [sp, #4]        //   V
    400568: b94007e0    ldr w0, [sp, #4]        // Line 4
    40056c: 910043ff    add sp, sp, #0x10       //   |
    400570: d65f03c0    ret                     //   V
```

    - *Why is `#0xffffffff` being stored in `w8`?* — this is the two's complement representation of -1
- *When might this function cause an error?* — when denominator is 0

- *How would you modify the C code to avoid an error?*

```c
int divide_safe(int numerator, int denominator) {
    int result = -1;
    if (denominator != 0) {
        result = numerator / denominator;
    }
    return result;
}
```

Conditional assembly code

```
0000000000400544 <divide_safe>:
    400544: d10043ff    sub sp, sp, #0x10                  // Line 1
    400548: 12800008    mov w8, #0xffffffff                // Line 2
    40054c: b9000fe0    str w0, [sp, #12]                  // Line 1
    400550: b9000be1    str w1, [sp, #8]                   //   V
    400554: b90007e8    str w8, [sp, #4]                   // Line 2
    400558: b9400be8    ldr w8, [sp, #8]                   // Line 3
    40055c: 340000a8    cbz w8, 400570 <divide_safe+0x2c>  //   V
    400560: b9400fe8    ldr w8, [sp, #12]                  // Line 4
    400564: b9400be9    ldr w9, [sp, #8]                   //   |
    400568: 1ac90d08    sdiv    w8, w8, w9                 //   |
    40056c: b90007e8    str w8, [sp, #4]                   //   V
    400570: b94007e0    ldr w0, [sp, #4]                   // Line 6
    400574: 910043ff    add sp, sp, #0x10                  //   |
    400578: d65f03c0    ret                                //   V
```

- *What does the `cbz` instruction do?* — "jumps" (i.e., branches) to a different instruction when the specified register's value is zero
- *Why does the assembly use `cbz` when the C code contains `!= 0`?* — the C code checks for the condition that must be true to execute the if body, whereas the assembly code checks for the condition that must be true to **skip over** the if body