Assembly: conditionals; `goto`

COSC 208, Introduction to Computer Systems, 2021-10-15

Announcements

- Project 2 Part A due Thursday, Oct 21

Outline

- Warm-up
- From conditionals to `goto`

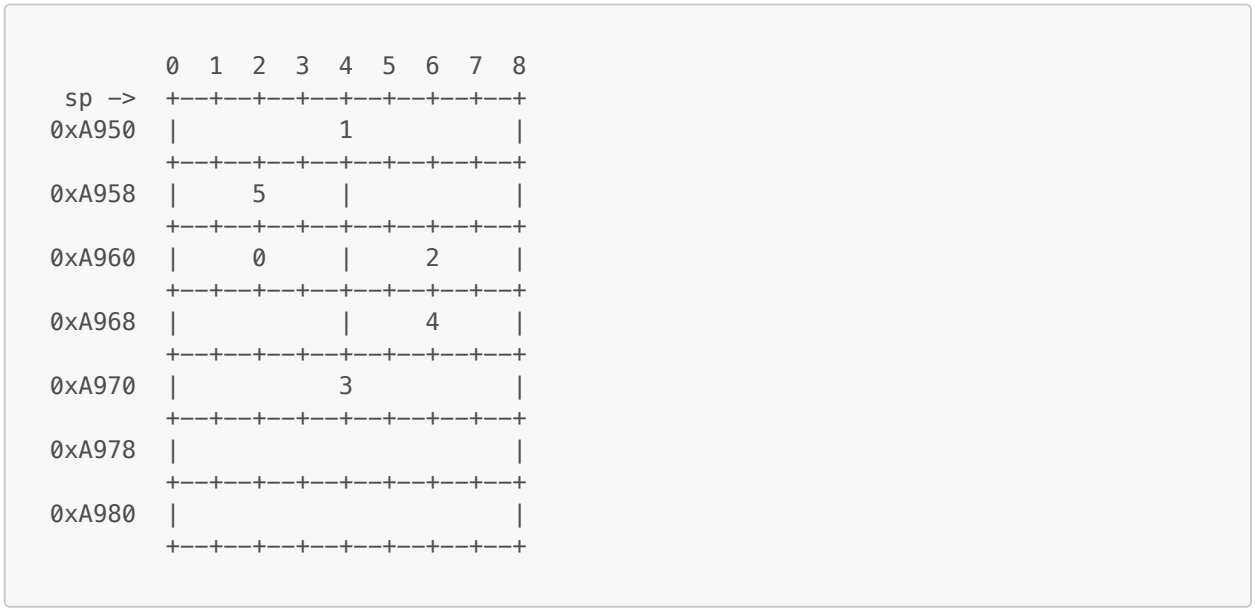
Warm-up

- 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]
```



Mapping C code to assembly code

- Mathematical operation pattern
  - load (ldr) first operand from the stack into a register
  - load (ldr) second operand from the stack into a register
  - compute
  - store (str) resulting value from a register onto the stack
- Local variable initialization pattern
  - move (mov) constant value into register
  - store (str) register value onto the stack
- Function begin pattern
  - create stack frame by decreasing (sub) stack pointer
  - store (str) parameter values in registers w/x0, w/x1, etc. onto the stack
- Function return/end pattern
  - load (ldr) return value from the stack into the w/x0 register
  - destroy stack frame by increasing (add) stack pointer
  - return (ret) to caller

`goto`

- Mapping C conditionals to assembly code. Last lecture example

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

```
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                  // Line 7
    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
- *How would we express `cbz` in C code?* — using an if statement and a `goto` statement

```c
int divide_safe_goto(int numerator, int denominator) {
    int result = -1;
    if (denominator == 0)
        goto after;
    result = numerator / denominator;
after:
    return result;
}
```

Practice

- *What happens if the code includes an else statement?* — if condition is true, execute the if body and skip over the else body; if condition is false, skip over the if body and execute the else body

```c
int flip(int bit) {
   int result = -1;
   if (bit == 0) {
        result = 1;
    }
    else {
        result = 0;
    }
    return result;
}
```

- Q2: *The above 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.*

```
0000000000400544 <flip>:
    400544: d10043ff    sub sp, sp, #0x10               // Line 1
    400548: 12800008    mov w8, #0xffffffff             // Line 2
    40054c: b9000fe0    str w0, [sp, #12]               // Line 1
    400550: b9000be8    str w8, [sp, #8]                // Line 2
    400554: b9400fe8    ldr w8, [sp, #12]               // Line 3
    400558: 35000088    cbnz    w8, 400568 <flip+0x24>  //   V
    40055c: 52800028    mov w8, #0x1                    // Line 4
    400560: b9000be8    str w8, [sp, #8]                //   V
    400564: 14000002    b   40056c <flip+0x28>          // Line 5
    400568: b9000bff    str wzr, [sp, #8]               // Line 7
    40056c: b9400be0    ldr w0, [sp, #8]                // Line 9
    400570: 910043ff    add sp, sp, #0x10               //   |
    400574: d65f03c0    ret                             //   V
```

- Q3: *How does the C code and its assembly differ in terms of the conditional execution? i.e. compare and contrast the `else` and the two branches.*

- Q4: *Write a function called `flip_goto` that behaves the same as `flip` but matches the structure of the assembly code that will be generated for `flip`. (Hint: you'll need two `goto` statements.)*

```
int flip_goto(int bit) {
    int result = -1;
    if (bit != 0)
        goto else_body;
    result = 1;
    goto after_else;
else_body:
    result = 0;
after_else:
    return result;
}
```

Translate the following assembly code snippets into low-level C code, treating registers as if they were variable names. (Hint: each snippet translates into an if-statement)

- Q5:

```
cmp w0, w1
b.eq 0xAB40 <foo+0x40>
```

```
if (w0 == w1)
    pc = 0xAB40
```

- Q6:

```
cmp w0, #0x20
b.lt 0xAB80 <foo+0x80>
```

```
if (w0 < 0x20)
    pc = 0xAB8D
```

- Q7:

```
cmp w1, #0x1
b.ne 0xABC0 <foo+0xC0>
```

```
if (w1 != 1)
    pc = 0xABC0
```

- Q8:

```
cmp w0, w1
b.le 0xABF0 <foo+0xF0>
```

```
if (w0 <= w1)
    pc = 0xABF0
```