Assembly: operations; load/store cont.

COSC 208, Introduction to Computer Systems, 2021-10-08

Write the C code equivalent for each line of assembly, treating registers as if they were variable names.

- Q1: `lsl w9, w9, w10`

  ```
  w9 = w9 << w10
  ```

- Q2: `and w9, w9, w10`

  ```
  w9 = w9 & w10
  ```

- Q3: `mul w9, w9, w10`

  ```
  w9 = w9 * w10
  ```

- Q4: `sdiv w9, w9, w10`

  ```
  w9 = w9 / w10
  ```

  The `udiv` and `sdiv` instructions operate on 32-bit and 64-bit data respectively. Note that you cannot multiply 32-bit registers with 64 bit registers.

Practice

The following C program (`operands.c`) has been compiled into assembly:

```c
int operandsA(int a) {
    return a;
}

long operandsB(long b) {
    return b;
}

int operandsC(int *c) {
    return *c;
}

long operandsD(long *d) {
    return *d;
}

int main() {
    operandsA(5);
    operandsB(5);
    int x = 5;
    operandsC(&x);
    long y = 5;
    operandsD(&y);
}
```

- Q5: *Write the C code equivalent for each line of assembly, treating registers as if they were variable names. The assembly code for the `operandsA` function has already been translated into C code.*

```
    00000000000007ec <operandsA>:
        7ec:    d10043ff    sub sp, sp, #0x10   // sp = sp - 0x10
        7f0:    b9000fe0    str w0, [sp, #12]   // *(sp + 12) = w0
        7f4:    b9400fe0    ldr w0, [sp, #12]   // w0 = *(sp + 12)
        7f8:    910043ff    add sp, sp, #0x10   // sp = sp + 0x10
        7fc:    d65f03c0    ret                 // return

    0000000000000800 <operandsB>:
        800:    d10043ff    sub sp, sp, #0x10   // sp = sp - 0x10
        804:    f90007e0    str x0, [sp, #8]    // *(sp + 8) = w0
        808:    f94007e0    ldr x0, [sp, #8]    // x0 = *(sp + 8)
        80c:    910043ff    add sp, sp, #0x10   // sp = sp + 0x10
        810:    d65f03c0    ret                 // return
```

```
0000000000000814 <operandsC>:
    814:    d10043ff    sub sp, sp, #0x10   // sp = sp - 0x10
    818:    f90007e0    str x0, [sp, #8]    // *(sp + 8) = x0
    81c:    f94007e0    ldr x0, [sp, #8]    // x0 = *(sp + 8)
    820:    b9400000    ldr w0, [x0]        // w0 = *x0
    824:    910043ff    add sp, sp, #0x10   // sp = sp + 0x10
    828:    d65f03c0    ret                 // return

000000000000082c <operandsD>:
    82c:    d10043ff    sub sp, sp, #0x10   // sp = sp - 0x10
    830:    f90007e0    str x0, [sp, #8]    // *(sp + 8) = x0
    834:    f94007e0    ldr x0, [sp, #8]    // x0 = *(sp + 8)
    838:    f9400000    ldr x0, [x0]        // x0 = *x0
    83c:    910043ff    add sp, sp, #0x10   // sp = sp + 0x10
    840:    d65f03c0    ret                 // return
```

- Q6: *How does the assembly code for `operandsA` and `operandsB` differ? Why?*
    - `operandsA` takes and returns an int, which is 32-bits, whereas `operandsB` takes and returns a long, which is 64-bits, so:
        - `operandsA` uses `w0` while `operandsB` uses `x0`
        - `operandsA` stores the parameter at `sp + 12` while `operandsB` stores the parameter at `sp + 8`
- Q7: *How does the assembly code for `operandsB` and `operandsD` differ? Why?*
    - `operandsB` takes and returns a long, whereas `operandsD` takes a pointer to a long and returns a long, so:
    - `operandsD` must deference the pointer (`ldr x0, [x0]`) before returning
- Q8: *How does the assembly code for `operandsC` and `operandsD` differ? Why?*
    - `operandsC` takes a pointer to an int and returns and int, whereas `operandsD` takes a pointer to a long and returns a long
    - both take a memory address (a 64-bit value), which is initially in `x0` and stored at `sp + 8`
    - the derefence of the pointer is a 32-bit value in `operandsC` and a 64-bit value in `operandsD`, so the value is loaded into `w0` in `operandsC` and `x0` in `operandsD`

- Q6: *Write the C code equivalent for each line of assembly, treating registers as if they were variable names.*

```
000000000000083c <deref>:
        83c:   d10083ff        sub     sp, sp, #0x20    // sp = sp - 0x20
        840:   f90007e0        str     x0, [sp, #8]     // *(sp + 8) = x0
        844:   f94007e0        ldr     x0, [sp, #8]     // x0 = *(sp + 8)
        848:   f9400000        ldr     x0, [x0]         // w0 = *x0
        84c:   f9000fe0        str     x0, [sp, #24]    // *(sp + 24) = w0
        850:   f9400fe0        ldr     x0, [sp, #24]    // w0 = *(sp + 24)
        854:   910083ff        add     sp, sp, #0x20    // sp = sp + 0x20
        858:   d65f03c0        ret                      // return
```