Assembly: load/store operations; arithmetic operations; translating assembly code to low-level C code

COSC 208, Introduction to Computer Systems, 2022-03-22

Announcements

- Project 2 due Thursday, Mar 31

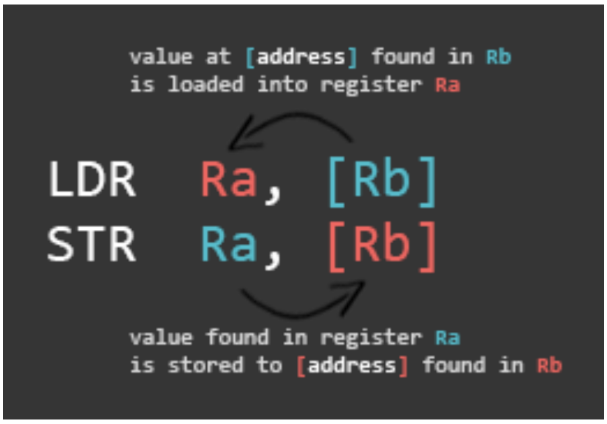
Outline

- Warm-up
- Assembly (recap)
- Load/store operations
- Arithmetic and bitwise operations
- Translating assembly code to low-level C code

Operands

- Registers
    - General purpose: `w0` through `w30` (32-bit) and `x0` through `x30` (64-bit)
    - Stack pointer (top of current stack frame): `sp`
- Constant -- e.g., `#0x20` vs `#12` (hexa vs decimal)
- Memory
    - Dereference --- e.g., `[x1]`
    - Add to (offset from) memory address, then dereference --- e.g., `[sp,#16]`

Load/store operations



value at [address] found in Rb
is loaded into register Ra

```
LDR   Ra, [Rb]
STR   Ra, [Rb]
```

value found in register Ra
is stored to [address] found in Rb

Warm-up: load/store operations

Q1: *Write the C code equivalent for each line of assembly, treating registers as if they were variable names.*

- `ldr x0, [sp]`

- `str w0, [sp]`

- `ldr x1, [sp, #12]`

- `str x2, [x3, #0x10]`

Arithmetic and logical operations

Q2: *Write the C code equivalent for each line of assembly, treating registers as if they were variable names.*

- `lsl w0, w1, w2`

- `and w3, w4, #20`

- `mul w5, w6, #0x11`

- `sdiv x7, x8, x9`

Mapping assembly code to C code

Q3: *The following C code was compiled into assembly.*

```
1   #include <stdio.h>
2   int years_to_double(int rate) {
3       int ruleof72 = 72;
4       int years = ruleof72 / rate;
5       return years;
6   }
7   int main() {
8       int r = 10;
9       int y = years_to_double(r);
10      printf("With an interest rate of %d%% it will take ~%d
            years to double your money\n", r, y);
11  }
```

For each line of assembly, indicate which original line of C code (above) the assembly instruction was derived from.

```
000000000000076c <years_to_double>:
    76c:    d10083ff    sub sp, sp, #0x20
    770:    b9000fe0    str w0, [sp, #12]
    774:    52800900    mov w0, #0x48
    778:    b9001be0    str w0, [sp, #24]
    77c:    b9401be1    ldr w1, [sp, #24]
    780:    b9400fe0    ldr w0, [sp, #12]
    784:    1ac00c20    sdiv    w0, w1, w0
    788:    b9001fe0    str w0, [sp, #28]
    78c:    b9401fe0    ldr w0, [sp, #28]
    790:    910083ff    add sp, sp, #0x20
    794:    d65f03c0    ret
```

Translating assembly code to low-level C code

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);
}
```

Q4: *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 low-level 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   //
    804:    f90007e0    str x0, [sp, #8]    //
    808:    f94007e0    ldr x0, [sp, #8]    //
    80c:    910043ff    add sp, sp, #0x10   //
    810:    d65f03c0    ret                 //

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

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

Q5: *How does the assembly code for `operandsA` and `operandsB` differ? Why?*

Q6: *How does the assembly code for `operandsB` and `operandsD` differ? Why?*

Q7: *How does the assembly code for `operandsC` and `operandsD` differ? Why?*