Computer Systems, Spring 2019

Week 4 Lab

Assembly Language Instructions

Solutions

1. Suppose we have some variables in the registers. Just use the register name as a variable name: for example, think of the register R4 as a variable name. Avoid using R0 and R15; thus the variables should be restricted to R1, R2, ..., R14. Write a sequence of instructions that carry out this calculation: `R3 := R1 + R2 * R3`. Just use arithmetic instructions, and just write a code fragment, not a complete program.

   Solution.

   ```
   mul    R4,R2,R3     ; R4 := R2*R3
   add    R3,R1,R4     ; R3 := R1 + (R2*R3)
   ```

2. Now suppose that we have variables x, y, z in memory. Write a sequence of instructions that carry out this calculation: `z := x - 13 * y`. Write a comment on each instruction that describes what it does. The comments should be as informative as possible: for example, if R1 and R2 contain variables x and y respectively, then a good comment for add R5,R1,R2 would be `R5 := x+y` and a comment like `R5 := R1+R2` gives little "added value".

   Solution.

   ```
   load    R1,x[R0]    ; R1 := x
   lea     R2,13[R0]   ; R2 := 13
   load    R3,y[R0]    ; R3 := y
   mul     R2,R2,R3    ; R2 := 13*y
   sub     R1,R1,R2    ; R1 := x - (13*y)
   store   R1,z[R0]    ; z := x - (13*y)
   ```

3. Hand-execute the following program fragment. After each instruction, show what register has changed, and its new value. Add a comment to each instruction that describes what it does.

   ```
   lea   R2,23[R0]
   lea   R3,4[R0]
   lea   R5,30[R0]
   add   R3,R5,R3
   add   R4,R2,R0
   sub   R6,R4,R5
   ```

   Solution. The effect of each instruction is shown in a comment.

```
lea   R2,23[R0]   ; R2 := 23
lea   R3,4[R0]    ; R3 := 4
lea   R5,30[R0]   ; R5 := 30
add   R3,R5,R3    ; R3 := 34
add   R4,R2,R0    ; R4 := 23
sub   R6,R4,R5    ; R6 := -7
```

4. Explain the difference between the `load` and `store` instructions. Write the instructions needed to perform

```
R4 := x
total := R5
```

Solution. The load instruction copies a word from memory to a register; the store instruction copies a word from a register to memory.

```
load   R4,x[R0]         ; R4 := x
store  R5,total[R0]     ; total := R5
```

5. Explain the difference between the `load` and `lea` instructions. Write the instructions needed to perform

```
R1 := 27
R2 := x
```

Solution. The load instruction copies a variable (specified by its address) from memory into the destination register. The lea instruction puts a constant into the destination register. (Later these descriptions will be expanded; the instructions actually do a little more.)

```
lea    R1,27[R0]  ; R1 := 27
load   R2,x[R0]   ; R2 := x
```

6. Sometimes in programming you need to swap the values of two variables. For example, suppose that x = 29 and y = 67. After swapping the variables, the values are x = 67 and y = 29.

   (a) Explain why you can't just write x := y; y := x.

   (b) Write assignment statements that swap the two variables in a high level language (you may use our Computer Systems 1 language, or Python).

   (c) Write assembly language instructions that swap the two variables. Hint: you'll need to use registers and some load and store instructions; there is no way to do this without passing the data through registers.

   (d) Discuss the two versions of swap that you wrote (the high level language version and the assembly language version).

(e) Write a complete assembly language program that defines two variables, x (with initial value 3) and y (with initial value 19). The program should swap the variables and then halt.

Solution.

(a) This is incorrect because the old value of x is destroyed by the first assignment. The final result would be x = 29 and y = 29.

(b) Here is a correct version, which uses a third variable temp: temp := x; x := y; y := temp

(c) The best solution is to use two registers (it doesn't matter which registers; R4 and R5 are chosen arbitrarily here):

```
    load   R4,x[R0]       ; R4 := x
    load   R5,y[R0]       ; R5 := y
    store  R5,x[R0]       ; x := old value of y
    store  R4,y[R0]       ; y := old value of x
```

Another way to solve it is to translate each of the three assignment statements above into assembly language.

```
; temp := x
    load   R4,x[R0]       ; R4 := x
    store  R4,temp[R0]    ; temp := old value of x
; x := y
    load   R4,y[R0]       ; R4 := old value of y
    store  R4,x[R0]       ; x := old value of y
; y := temp
    load   R4,temp[R0]    ; R4 := temp = old value of x
    store  R4,y[R0]       ; y := old value of x
```

(d) The high level language version needs an extra variable temp. The assembly language version cannot copy from one variable in memory directly to another; the only way to copy a variable is to load it into a register and then to store it into the other variable. Since we have to load the variables into registers anyway, it's possible to use the registers to hold the old values of x and y; thus the registers play the role of temp. The most important point is that both of the assembly language solutions given above are correct. Correctness is always the most important criterion. However, we're also interested in efficiency which is a secondary consideration but still important. The assembly language version using two registers requires only four instructions; the literal translation of the high level statements (using temp) requires six instructions and will run more slowly. Normally in assembly language programming we use lots of registers in order to simplify the code and make it more efficient.
(Aside: You may have come across the "xor swap trick" for swapping two variables using exclusive or. If not, that's fine! You don't need to learn it. If you do know about the xor swap trick, consider that it is considerably less efficient than the method shown above (it uses

longer code, more memory space, and more execution time), and it makes the program harder to read. There is one situation where the xor swap trick is appropriate (the variables are already in registers, and there is no third register available to use as temp). However, this situation is rare, so the xor swap trick is usually inappropriate.)

(e) The program begins with the instructions that do the work, followed by an instruction to terminate the program. At the end come the data statements, which define the variables and their initial values.

```
    load   R4,x[R0]      ; R4 := x
    load   R5,y[R0]      ; R5 := y
    store  R5,x[R0]      ; x := old value of y
    store  R4,y[R0]      ; y := old value of x
    trap   R0,R0,R0      ; terminate
x   data   3
y   data   19
```

Problems using the computer

See Moodle for instructions on downloading Sigma16 and running it.