MIPS Assembly: Loops, Memory, and Instructions

CS 64: Computer Organization and Design Logic
Lecture #6

Ziad Matni
Dept. of Computer Science, UCSB

Looking for Mandarin bilinguals!

If you are a native Mandarin speaker, please consider participating in our study. This study looks at perceptions of Mandarin speakers. You will be entered into a lottery where you can earn $100 VISA Card.

Please contact **ahansia**@umail.ucsb.edu if interested.

This Week on "Didja Know Dat?!"

The very first High-Level programming languages were developed in the 1940s and 1950s. Like **Plankalkül**, **Short Code**, **Autocode**, **FORTRAN** and **COBOL**. Then, over the next 20 years, came a slew of programming languages, including **Pascal**, **A**, **APL**, **CPL**, **BCPL**, which led to **B**.

When UNIX was being developed at Bell Labs in the 1970s, the go-to language was **B**, which was not up-to-speed with current H/W. So, it was developed to be "better" and then called… **C** (who said engineers aren't creative?). We also got **BASIC** which came with every Apple computer.

In 1985, Bjarne Stroustrup, took C and gave it "classes". Since it was incrementally better than C, he called it **C++** (get it?).
We also got **Eiffel**, **Ada**, and **Perl** around this time.

In the 1990s, we got OOPLs like **Visual Basic, Java**, **Python**, **Ruby**, **Objective-C**. Microsoft also mucked with C and Java and wanted to call the new language "Cool" ("C-like Object Oriented Language"), but went with the more boring **C#**. :/
The WWW ushered in **HTML**, **CSS**, **JS**, **PHP** among others...

And in the 21st Century: **D**, **Go**, **Swift**, **Scala**, and many others.

<u>*Fun-Fact*</u>: We can cover most of the alphabet with programming languages: **A**, **B**, **C**, **D**, **E**, **F**, **G**, **J**, **K**, **L**, **M**, **Q**, **R**, **S**, and **T** and don't forget **P#**, **J#**, **F#**, **X++**, **C–**, and **A++**…

Thank you Wikipedia!

Lecture Outline

- Loops
- Reading/Writing MIPS Memory
- MIPS Memory Conventions
- MIPS Instruction Representations

Any Questions From Last Lecture?

Loops

- How might we translate the following to assembly?

```
sum = 0;
while (n != 0)
{
    sum = sum + n;
    n--;
}
printf(sum);
```

n = 3; sum = 0;
while (n != 0) { sum += n; n--; }

```
.text
main:
    li $t0, 3    # n
    li $t1, 0    # running sum
loop:
    beq $t0, $zero, loop_exit
    addu $t1, $t1, $t0
    addi $t0, $t0, -1
    j loop

loop_exit:
    li $v0, 1
    la $a0, $t1
    syscall

    li $v0, 10
    syscall
```

Set up the variables in $t0, $t1

If **$t0 == 0** go to “loop_exit”

(otherwise) make $t1 the (unsigned) sum of $t1 and $t0 (i.e. **sum += n**)

decrement $t0 (i.e. **n--**)

jump to the code labeled “loop” (i.e. **repeat loop**)

prepare to print out an integer, which is inside the $t1 reg. (i.e. **print sum**)

end the program

Branching/Loop Exercise

Consider this C/C++ code:

```
int x(SomeNumber), y;
if (x == 5) y = 8;
else if (x < 7) y = x + x;
else y = -1;
print(y);
```

Let's write it in MIPS assembly!

Branching/Loop Exercise

```
int x(SomeNumber), y;
if (x == 5) y = 8;
else if (x < 7) y = x + x;
else y = -1;
print(y);
```

Plan it out:

```
main:
    # setup vars x and y in regs
    # do the 1st if and branch to "equal_five"
    # otherwise do the 2nd if and branch to "less_than_seven"
    # otherwise do the else statement (y = -1)
           (is there something else that should go here???)

equal_five:
    # make y = 8
less_than_seven:
    # make y = x + x

print_out_and_exit:
    # print out the answer
    # exit the program
```

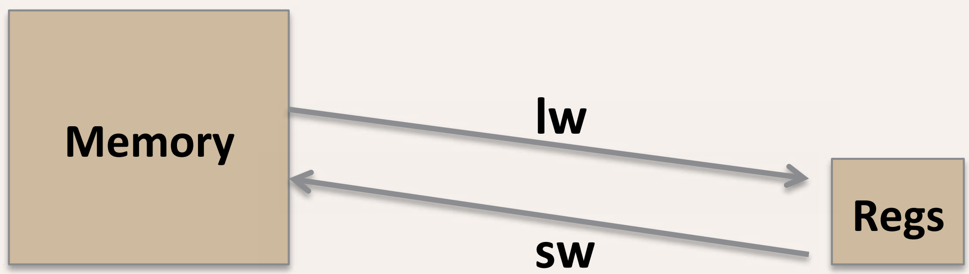
Larger Data Structures

- Recall: registers vs. memory
  - Where do data structures, arrays, etc. go?
  - Which is faster to access? Why?

- Some data structures have to be stored in memory
  - So we need instructions that "shuttle" data to/from the CPU and computer memory (RAM)

Accessing Memory

- Two base instructions:
  - load-word (**lw**) from memory to registers
  - store-word (**sw**) from registers to memory



- MIPS lacks instructions that do more with memory than access it (e.g., retrieve something from memory and then add)
  - Operations are done step-by-step
  - Mark of RISC architecture

Example 4

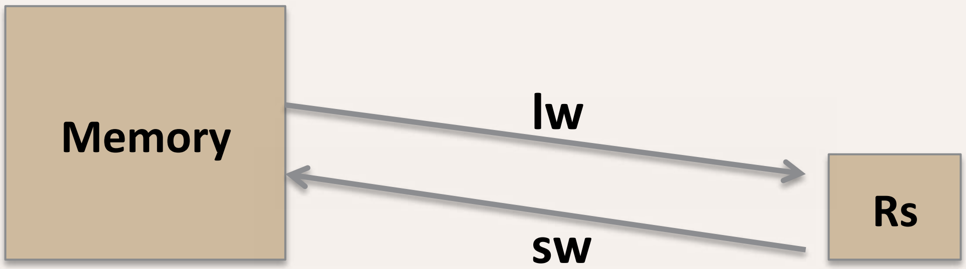
What does this do?

```
.data
num1: .word 42   # What is 42?
num2: .word 7    # What is 7?
num3: .space 1   # What is 1?

.text
main:
    lw $t0, num1
    lw $t1, num2
    add $t2, $t0, $t1
    sw $t2, num3

    li $v0, 1
    lw $a0, num3
    syscall

    li $v0, 10
    syscall
```



YOUR TO-DOs

- Assignment/Lab #3
  - ~~Will post online on WEDNESDAY~~
  - ~~Your lab is on THURSDAY~~
  - Assignment will be due on ~~FRIDAY~~ MONDAY

</LECTURE>