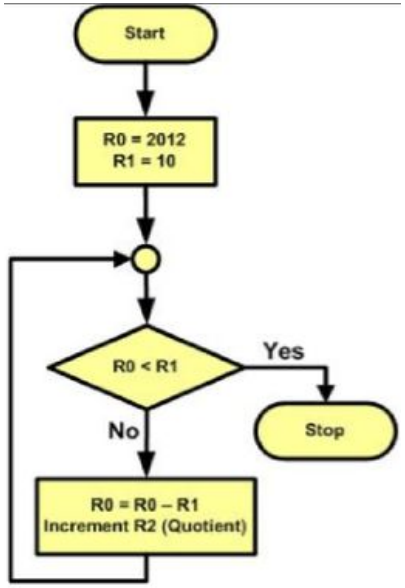
Loops in MIPS
Memory Topics in MIPS

CS 64: Computer Organization and Design Logic
Lecture #7
Fall 2019

Ziad Matni, Ph.D.
Dept. of Computer Science, UCSB



This Week on "Didja Know Dat?!"

Steve Wozniak and Steve Job's first commercial venture was the **Apple 1** in **1976** using an **8-bit MOS 6502 CPU**. It was built for $500 and initially **sold for $666.66** because Wozniak "*liked repeating digits*" (about $2900 in today's dollars). Keyboard and TV not included. They sold about 200 of them in 10 months, thus assuring the continuation of their company.

Previously, the only other popular "personal" computer was the Altair 8800, which you had to operate with switches!

Lecture Outline

- Loop Instructions
- Addressing MIPS Memory
- Global Variables
- Arrays

Any Questions From Last Lecture?

Pop Quiz!

- **You have 5 minutes to fill in the missing code. You can use your MIPS Reference Card.**
- **Fill in the 4 blank spaces :**

```
main: # assume $t0 has been declared earlier (not here)
            li $t1, 0
            li _____________
            blt ___________________________
            li $t1, 1
exit:       _____________________________
            _____________________________
```

In C++, the code would be:

```
if (t0 >= -42)
   t1 = 1;
else
   t1 = 0;
```

Pop Quiz!

- **You have 5 minutes to fill in the missing code. You can use your MIPS Reference Card.**
- **Fill in the 4 blank spaces :**

```
main: # assume $t0 has been declared earlier (not here)
            li $t1, 0
            li  $t2, -42          # something to compare!
            blt $t0, $t2, exit
            li $t1, 1
exit:       li $v0, 10
            syscall
```

In C++, the code would be:

```
if (t0 >= -42)
   t1 = 1;
else
   t1 = 0;
```

Loops

- How might we translate the following C++ to assembly?

```
n = 3;
sum = 0;
while (n != 0)
{
    sum += n;
    n--;
}
cout << 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
   move $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

Let's Run More Programs!!
Using SPIM

- More!!
- This time exploring conditional logic and loops

These assembly code programs are made available to you via the class webpage

More Branching Examples

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

```
.text
main:   # t0: x and t1: y
     li $t0, 5      # example
     li $t2, 5      # what's
this?
     beq $t0, $t2, equal_5

     # check if less than 7
     li $t2, 7
     slt $t3, $t0, $t2
     bne $t3, $zero, less_than_7

    # fall through to final else
     li $t1, -1
     j after_branches

equal_5:
     li $t1, 8
     j after_branches
```

```
less_than_7:
     add $t1, $t0, $t0
 # could jump to after_branches,
 # but this is what we will fall
 # through to anyways

after_branches:
# print out the value in y ($t1)
     li $v0, 1
     move $a0, $t1
     syscall

     # exit the program
     li $v0, 10
     syscall
```

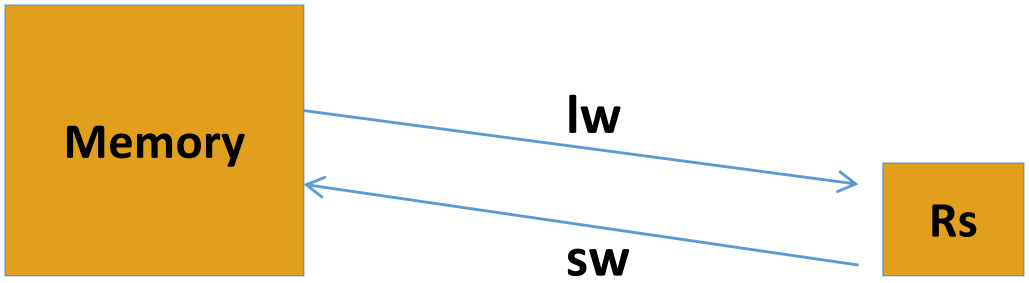
Larger Data Structures

- Recall: registers vs. memory
  - Where would 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

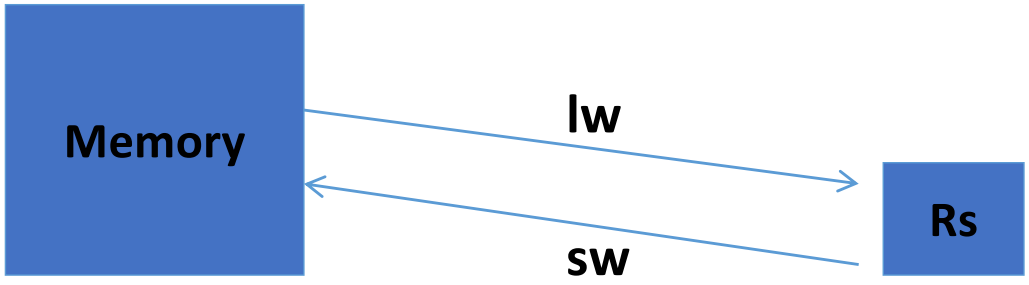
Example 4
What does this do?

```
.data
num1: .word 42
num2: .word 7
num3: .space 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
```



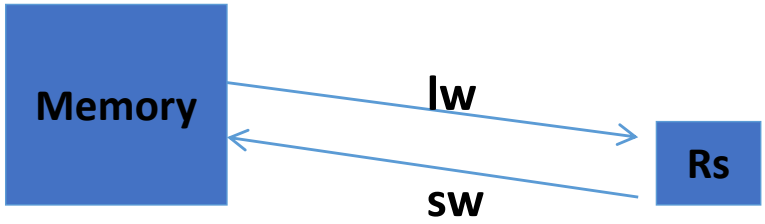
Example 4

```
.data
num1: .word 42   # define 32b w/ value = 42
num2: .word 7          # define 32b w/ value = 7
num3: .space 1   # define one (1) 32b space

.text
main:
      lw $t0, num1           # load what's in num1 (42) into $t0
      lw $t1, num2           # load what's in num2 (7) into $t1
      add $t2, $t0, $t1      # ($t0 + $t1) → $t2
      sw $t2, num3       # load what's in $t2 (49) into num3 space

      li $v0, 1
      lw $a0, num3      # put the number you want to print in $a0
      syscall           # print integer

      li $v0, 10        # exit
      syscall
```



Addressing Memory

- If you're not using the **.data** declarations, then you need *starting addresses* of the data in memory with ***lw*** and ***sw*** instructions

Example: `lw $t0, 0x0000400A` ← not a real address, just looks like one…

Example: `lw $t0, 16($s0)`

- 1 word = 32 bits (in MIPS)
  - So, in a 32-bit unit of memory, that's 4 bytes
  - Represented with 8 hexadecimals

8 x 4 bits = 32 bits... checks out…

- MIPS addresses sequential memory addresses, but not in "words"
  - Addresses are in Bytes instead
  - MIPS words *must* start at addresses that are multiples of 4
  - Called an ***alignment restriction***

YOUR TO-DOs

- Do readings!
  - Check syllabus for details!
- Review ALL the demo codes
  - Available via the class website
- Work on Assignment #3
  - Due on **Wednesday, 10/23, by 11:59:59 PM**

</LECTURE>