Flow Control & Memory Use in Assembly

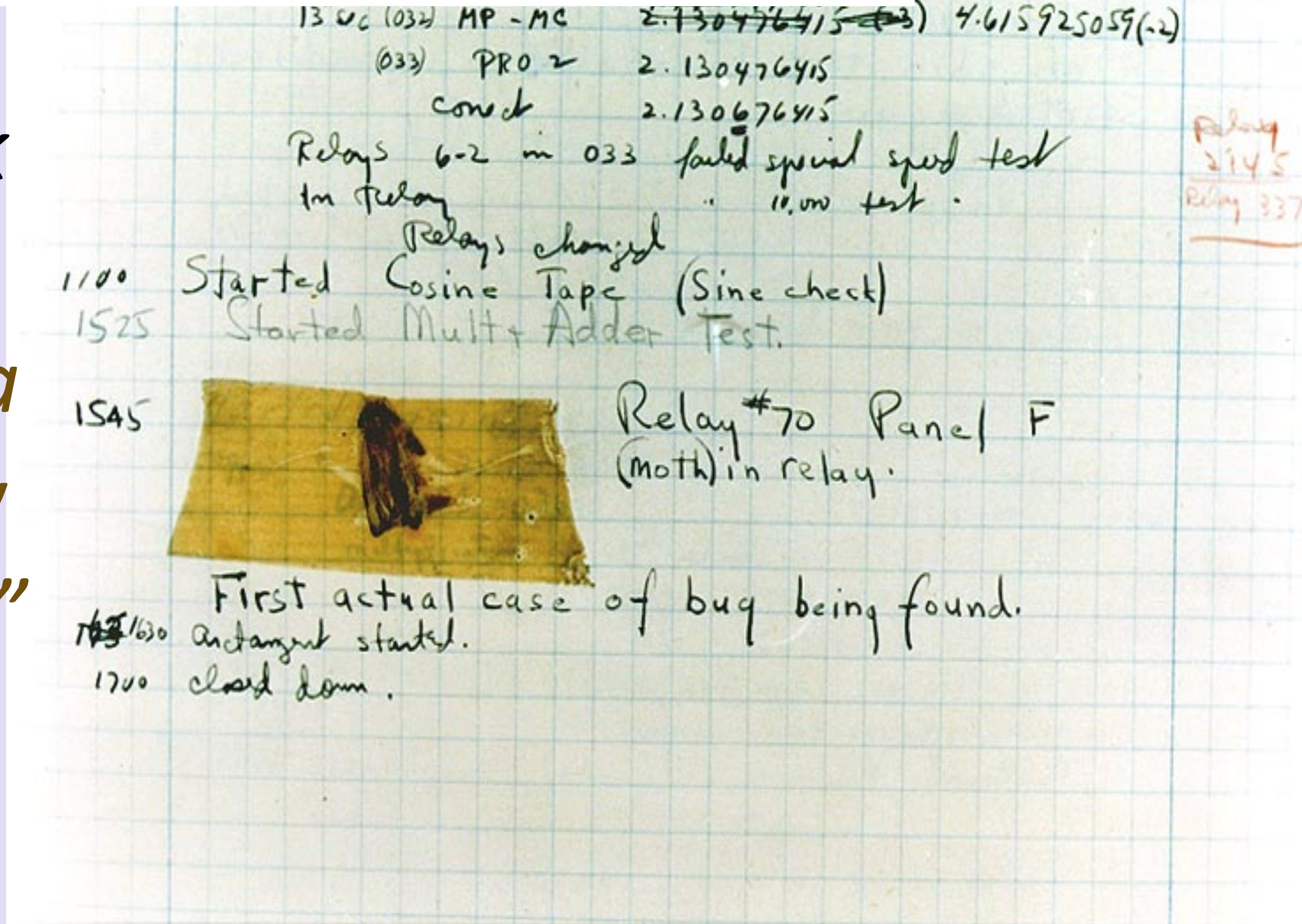
CS 64: Computer Organization and Design Logic
Lecture #6
Winter 2019

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

Legend: Adm. Grace Hopper coined the term "debugging" when a moth was removed from the computer she was working on (see below)

Reality: The term “bug” was used in engineering in the 19th century. As seen independently from various scientists, including Ada Lovelace and Thomas Edison.

This Week on “Didja Know Dat?!”



Lecture Outline

- **.data** Directives and Basic Memory Use
- Branching (Conditionals)
- Loops
- Accessing Data in Memory
- Memory Addressing

Any Questions From Last Lecture?

MIPS Peculiarity:
NOR used a NOT

- How to make a NOT function using ***NOR*** instead
- Recall: NOR = NOT OR
- Truth-Table:

A	B	A NOR B
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

Note that:

0 NOR x = NOT x

- So, in the absence of a NOT function,
  use a NOR with a 0 as one of the inputs!

.data Declaration Types

w/ Examples

```
var1:    .byte 9          # declare a single byte with value 9
var2:    .half 63         # declare a 16-bit half-word w/ val. 63
var3:    .word 9433       # declare a 32-bit word w/ val. 9433
num1:    .float 3.14      # declare 32-bit floating point number
num2:    .double 6.28     # declare 64-bit floating pointer number
str1:    .ascii "Text"    # declare a string of chars
str3:    .asciiz "Text"   # declare a null-terminated string
str2:    .space 5         # reserve 5 bytes of space (useful for arrays)
```

These are now reserved in memory and we can call them up by loading their memory address into the appropriate registers.

Highlighted ones are the ones most commonly used in this class.

Example

What does this do?

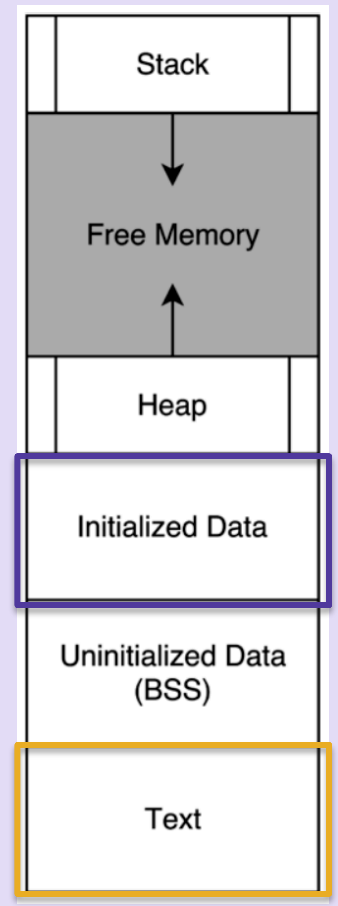
```
.data
name: .asciiz “Jimbo Jones is ”
rtn: .asciiz “ years old.\n”

.text
main:
    li $v0, 4
    la $a0, name      # la = load memory address
    syscall

    li $v0, 1
    li $a0, 15
    syscall

    li $v0, 4
    la $a0, rtn
    syscall

    li $v0, 10
    syscall
```



What goes in here? → (Initialized Data)

What goes in here? → (Text)

Conditionals

- What if we wanted to do:

```
if (x == 0) { printf("x is zero"); }
```

  – Can we write this in assembly with what we know?
    - No… we haven't covered **if-else** (aka *branching*)

- What do we need to implement this?
  – A way to *compare* numbers
  – A way to *conditionally execute* code

Relevant Instructions in MIPS

for use with branching conditionals

- Comparing numbers:

  set-less-than (`slt`)

  – Set some register (i.e. make it “1”) if a less-than comparison of some other registers is true

- Conditional execution:

  branch-on-equal (beq)

  branch-on-not-equal (bne)

  – “Go to” some other place in the code (i.e. jump)

if (x == 0) { printf("x is zero"); }

```
.data
   x_is_zero: .asciiz "x is zero"

.text
   bne $t0, $zero, after_print
   li $v0, 4
   la $a0, x_is_zero
   syscall

after_print:
   li $v0, 10
   syscall
```

Create a constant string called "**x_is_zero**"

If **$t0 != 0** go to the block labeled as "***after_print***"

(otherwise) prepare to print a string…

…and that string is inside of "x_is_zero"

End the program

Note the flow

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

DEMO !!!

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
```

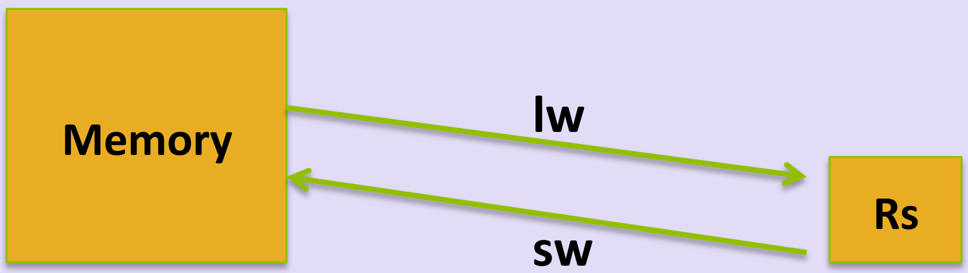
Larger Data Structures

- 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

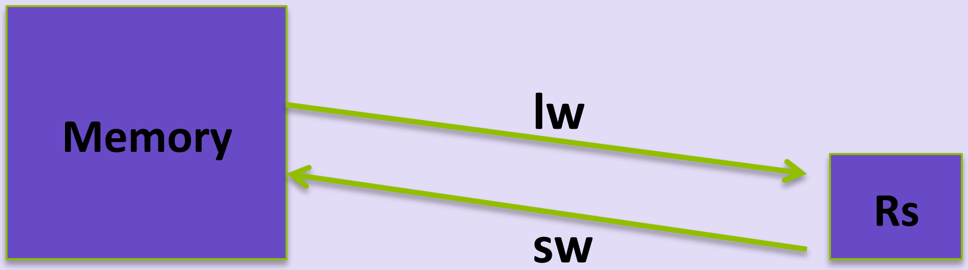
What does this do?

```
.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
```



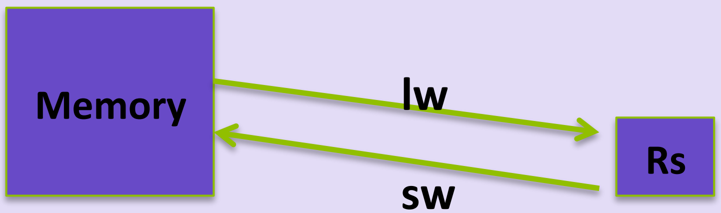
Example 4

```
.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
```



YOUR TO-DOs

- Review ALL the demo codes
  - Available via the class website
- Assignment #3
  - Due Monday!

</LECTURE>