16 Mips SPIM

Monday, October 12, 2015 10:00 AM

MIPS Simulator called SPIM

- spim
  - Not usually used in prompt
  - Been around for a pretty long time but updated recently to use the 32 bit architecture
- MIPS 32 didn't come out in 1985, but not a huge difference

SPIM

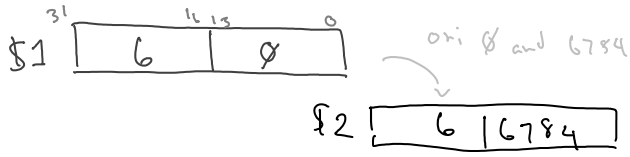
- Source code for SPIM is online
- SPIM comes with a teeny tiny fake operating system
  - Has print
  - Needs an OS to do that
- spim -file hello.s
- System call
  - Usually a way for the computer to grab the OS's attention
  - ssscall - an assembly instruction $v0
    - Telling the OS to do something for us
  - Print a string, put a 4 in $v0
  - Put the address of the string in $a0
- Hello.s
  - Need to specify instructions from data

```
        .text # from here on, I'm giving you instructions
main:
        li $v0, 4 # print_string
        la $a0, hello # load the string into $a0
        syscall
        li $v0, 10 # exit
        syscall

        .data # from here on, I'm giving you data
hello:
        .asciiz "Hello World!\n"

        # at the address hello, put the characters hello world!
        .asciiz puts a zero byte at the end of the string
```

- Stuff that starts with .
  - For the assembler, not actually given to the CPU
  - All .somethings
    - Called directives
  - li looks like an instruction
    - Li $v0, 4
      - 4 can be 32 bit.
      - But instructions can't fit a 32 number
      - Li, la, are pseudo instructions
      - The assembler being nice to you, pretending

there is an li, la isntruction
- But the assembler is actually turning it into two MIPS instructions
- Spim
- Load "hello.s"
- Step (multiple times)
  - Li becomes ori $2, $0, 4

- Li $v0, 400000
  - Lui $1, 6
    - Load upper immediate
      - Takes a 16 bit number and puts it in the upper 16 bits of a register, and sets the lower half to 0



  - Ori $2, $1, 6784
- Same thing happens for la
- Whole bunch more of pseudo instructions
  - Usually doesn't matter if using pseudo or regular instructions
  - So don't use $1, if you want to sue, make sure you're not accidentally trashing it.

Countdown.s (a loop printing out a bunch of numbers)

```
        .text
main:
        Li $s0, 10

Loop:
        li $v0, 1 # print_int
        move $a0, hello
        Syscall
        Li $v0, 4 #print)string
        La $a0, lf
        syscall

        sub $s0, $s0, 1 # assembler still knows subi
        Slt $s1, $s0,$ 0 # set t1 to result  of "t0 < 0"
        Beq $s1, $0, loop

        li $v0, 10 # exit
        syscall

        .data
lf:
        .asciiz "\n"
```

Subi immediate subtraction
Using s registers because syscall can trash t registers

Mult $t0, $t1

- Actual assembly instruction, don't give a destination register
- Can actually give a 64 bit result
- Low and high always receive result of a multiply instruction
- Implicit result $hi/$lo
- mfhi $t0, moves $hi into $t0
- mflo $t0, moves $lo into $t0
- In spim
    - mul $t7, $t0, $t1
    - Will get higher bits in $t1
    - If you need all 64 need to use the long way
    -