Printing strings in MIPS

# Printing a string

.data
.str: .asciiz "CPSC330: Computer Organizations\n" # NUL # terminated # string

.text # code section
.globl main

main:
    li $v0, 4 # system call for print_str
    la $a0, str # address of string to print
    syscall # print the string
    li $v0, 10 # system call for exit
    syscall # exit

Branching and Loops in MIPS

# What should title of the program be?

.text
.globl main

main:
    li $s0, 1 # $s0 = loop counter=lower bound
    li $s1, 10 # $s1 = upper bound of loop
    loop: move $a0, $s0 # print loop counter
    li $v0, 1
    syscall
    li $v0, 4 # print line break
    la $a0, linebrk
    syscall
    addi $s0, $s0, 1 # increase loop counter by 1
    bge $s0, $s1, loop # if $s0 <= $s1 goto loop
    li $v0, 10 # system call for exit
    syscall # exit

.linebrk: .asciiz "\n"
Example-scores.asm - Compute average of list of test scores and return in $v0.

# The list consists of scores from 0 through 100 and is
terminated with -1.
# However, if any other values occur, return -1 in $v0.
# Inputs:
# $a0 - address of list of scores
# Outputs:
# $v0 - returns the average of the test scores or -1.
# Locals:
# $t0 = score counter
# $t1 = score sum
# $t2 = temp for score
# $t3 = temp
# $t4 = holds terminator marker, -1
#--

SPIM Example

SPIM Example continued

.main: la $a0, scores
add $t0,$zero,$zero
.add $t1,$zero,$zero
.getscr: lw $t2, 0($a0)
.beq $t2, $t4, nomore
.slti $t3, $t2, 0
.bne $t3, $zero, badnum
.slti $t3, $t2, 101
.beq $t3, $zero, badnum
.add $t1, $t1, $t2
.addi $t0, $t0, 1
.addi $a0, $a0, 4
.j getscr

# Get address of list of scores. 
# Init score counter. 
# Init sum. 
# Put terminator value in $t4. 
# Get a score into $t2. 
# End of list? (score == -1)? 
# Is score <= 0? 
# Out of range, get out. 
# Out of range, leave. 
# Add score to sum. 
# Increment score counter. 
# Increment list address. 
# Go get another score.

badnum: addi $v0, $zero, -1
j pout
nomore: div $t1, $t0
.mflo $v0
.nop
.pout:
.move $a1, $v0
li $v0, 4
.call syscall

.data

MIPS code:

loop: 
slt $t0, $s0, $s1     # $t0 = 1 if g<h
bne $t0, $0, Loop     # if $t0!=0

# If not, invalid character, return -1.
# Exit on error.
# Computer average.
# Done, now print; no operation here
# Save result in $s1
# Load syscall code for print_int.
# Print string for print_int.
# Print integer.

scores:    .word 95, 92, 88, 100, 81, 90, 75, 99, 82, 79 -1
aestr: .ascii "The average of scores = "

Inequalities in MIPS

Compile by hand:
if (g < h) goto Loop;
Use this mapping:
g: $s0, h: $s1

MIPS code:
Loop: ...
st $t0, $s0, $s1     # $t0 = 1 if g<h
bne $t0, $0, Loop    # goto Loop
# if $t0!=0
# (if (g<h)) Loop:

# Branch if $t0 != 0 (g < h)
Inequalities in MIPS

Compile by hand:
if (g >= 1) goto Loop  (testing against constant)
Use this mapping:
g: $s0

MIPS code:
Loop: ...
slti $t0, $s0, 1            # $t0 = 1 if 
# $s0<1 (g<1)
beq $t0, $0, Loop       # goto Loop 
# if $t0==0 
# (if (g>=1))

Review MIPS

- The **MIPS** architecture is an example of a Reduced Instruction Set Computer architecture.

- MIPS is called load-store architecture: Main memory is accessed through load/store operations.

- MIPS (architecture) is an acronym for: Microprocessor without Interlocked Pipeline Stages. It helps in the pipelining design.

Alternative Architectures

- Design alternative:
  - provide more powerful operations
  - goal: to reduce number of instructions executed
  - danger: reduction can occur at the cost of simplicity, increasing the time a program takes to execute because instructions are slower.

  -“The path toward operation complexity is thus fraught with peril. To avoid these problems, designers have moved toward simpler instructions”

- Let’s look (briefly) at IA-32

The Intel IA – 32

- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: 57 new “MMX” instructions are added, Pentium II
- 1999: The Pentium III added another 70 instructions (SSE)
- 2001: Another 144 instructions (SSE2)
- 2003: AMD extends the architecture to increase address space to 64 bits, widens all registers to 64 bits and other changes (AMD64)
- 2004: Intel capitulates and embraces AMD64 (calls it EM64T) and adds more media extensions

- “This history illustrates the impact of the “golden handcuffs” of compatibility *adding new features as someone might add clothing to a packed bag* 
*an architecture that is difficult to explain and impossible to love*
IA-32 Overview

- **Complexity:**
  - Instructions from 1 to 17 bytes long
  - One operand must act as both a source and destination
  - One operand can come from memory
  - Complex addressing modes
    - E.g., "base or scaled index with 8 or 32 bit displacement"

- **Saving grace:**
  - The most frequently used instructions are not too difficult to build
  - Compilers avoid the portions of the architecture that are slow

"What the 80x86 lacks in style is made up in quantity, making it beautiful from the right perspective"

IA-32 Typical Instructions

- Four major types of integer instructions:
  - Data movement including move, push, pop
  - Arithmetic and logical (destination register or memory)
  - Control flow (use of condition codes / flags)
  - String instructions, including string move and string compare

IA-32 Instruction Formats

- Typical formats: (notice the different lengths)

Summary

- Instruction complexity is only one variable
  - Lower instruction count vs. higher CPI/lower clock rate
- Design Principles:
  - Simplicity favors regularity
  - Smaller is faster
  - Make the common case fast
- Instruction set architecture
  - A very important abstraction indeed!