MIPS Pseudoinstructions

Class 38

Introduction

- as you learned in the last problem of the homework, MIPS's anemic slt, beq, and bne are a pain to use
- but that's all there are in hardware
- if you're writing in machine language, you have no choice

- but you're not writing in machine language, you're writing in assembly language
- you are writing code for the assembler software, not directly for the machine hardware

Assembler

- since the assembler is software, it can be programmed to do anything the programmer wants it to do
- universally, MIPS assemblers are written with several important macro definitions built in
- for example, suppose you put this in an assembly language statement:

```
    bgt    $t0, $t1, label
```

- if you look in column 1 of the green card, there is no bgt instruction
- we wanted to have one, but it doesn't exist

Macro Definition

- built into the MIPS assembler is the following macro
- if you type in

```
bgt <rs>, <rt>, <label>
```

- the assembler rewrites this as:

```
slt  $at, <rt>, <rs>
bne  $at, $zero, <label>
```

- which was exactly the heart of the answer to the last problem

- $at is register 1, which is listed as “reserved for the compiler”
- this is exactly what that register is commonly used for

Pseudoinstructions

- bgt is called a pseudoinstruction
- it does not exist in hardware
- you don't need it, because it's just two instructions chained together
- are you really that lazy?

- the reason MIPS assemblers provide the pseudoinstruction macro is because almost every other CPU has this as a real instruction, and programmers are used to using it
- remember I said even the tiny Arduino has 14 different branch instructions

- the other reason is that sure, it's easy to write two lines
- but it's extremely easy to get confused: flipping the order, reversing the logic, using true instead of false
- it reduces error

Pseudoinstructions

Pseudoinstruction

A pseudoinstruction is a legal MIPS assembly language instruction that does not exist in the MIPS processor. It is provided as a convenience for the programmer. When you use a pseudoinstruction in a MIPS assembly language program, the assembler translates them into real MIPS instructions that perform the same task.

Task	Pseudoinstruction	Actual Instrs

If rs $<$ rt, branch to label.	`blt <rs>, <rt>, <label>`	`slt $at, <rs>, <rt>`
		`bne $at, $zero, <label>`

If rs $\leq$ rt, branch to label.	`ble <rs>, <rt>, <label>`	`slt $at, <rt>, <rs>`
		`beq $at, $zero, <label>`

If r1 $>$ r2, branch to label.	`bgt <rs>, <rt>, <label>`	`slt $at, <rt>, <rs>`
		`bne $at, $zero, <label>`

If r1 $\geq$ r2, branch to label.	`bge <rs>, <rt>, <label>`	`slt $at, <rs>, <rt>`
		`beq $at, $zero, <label>`

Copy register to register.	`move <rd>, <rs>`	`addu <rd>, $zero, <rs>`
Load non-negative constant into register.	`li <rs>, <imm>`	`ori <rs>, $zero, <imm>`

Load negative constant into register.	`li <rs>, <imm>`	`lui $at, -1`
		`ori <rs>, $at, <imm>`

Load *value* from `<label>` memory location into register. `<ofs>` is offset of that location from start of data segment; calculated by assembler.	`lw <rs>, <label>`	`lui $at, 0xnnnn`
		`lw <rs>, <ofs>($at)`

Load *address* of `<label>` memory location into register. `<ofs>` is offset of that location from start of data segment; calculated by assembler.	`la <rd>, <label>`	`lui $at, 0xnnnn`
		`ori <rd>, $at, <ofs>`