

CSE 141 Discussion

Optimizing your Project ISA

Design Concept

- So what are you supposed to turn in?
 - Your ISA and basic descriptions of your instructions (enough for someone that hasn't seen it before can understand)
 - “Before and after code sequences”
 - RISC-style code for SuperGarbage, then your ISA code
 - Analysis of overall efficiency of your code
 - Analysis of your novel instructions vs. RISC ISA
 - These require a little more details than say “add”, because we will want to know your reasoning behind having these instructions

Alpha Release

- So what are you supposed to turn in?
 - A Beta Draft of your ISA specification (see the Beta Release deadline for an example)
 - Your code for both SG and fibonacci
 - Fibonacci does not really need to have specialized instructions, this is to prove your ISA is “general purpose”
 - A detailed description of how function calls work in your ISA
 - If you had to change you ISA a lot, include an updated version of your performance analysis

Optimizing your ISA

- Grouping Instructions
 - This will save you time later when writing your control logic in CSE 141L
 - Consider having similar instructions with the same 1 or 2 high order bits in your opcode.
 - Example:
 - bne opcode: 11000, jmp opcode 11010
 - add opcode 00110, sub opcode 00100

Optimizing your ISA

- Don't like the number of registers you have?
 - Specialized registers might be a solution
 - These registers have a specific purpose and (generally) are tied to specific uses
 - Example: \$sp (stack pointer) in MIPS
 - Example for your ISA: Whenever you use an arithmetic function, you can dump the result into an accumulator-type register, and if you want to use that value, just move the value to a general register
- ```
reset //resets the accumulator register ($ar) to $zero
adda $s1 // $ar = $ar + $s1
addai 10 // $ar = $ar + 10
movea $s2 // $s2 = $ar
```
- It may take more instructions BUT remember you are writing for your smaller bit ISA versus the 32-bit ISA of MIPS

# Optimizing your ISA

- Don't over-generalize
  - Make it “easy” to write your program. Remember, the first thing you should have done was write the program in MIPS (you don't have to look for an optimized MIPS implementation either, it can be naïve)
  - As such, don't have instructions that are “too broad”
    - Example: Go ahead and write a nor instruction, you probably don't need an “or” or “not” instruction separately

# Optimizing your ISA

- But...don't over-specialize
  - Sure you may know that you always do a  $pc=pc+4$  and that's the only add you have here, but do you need a “add4” instruction?
  - Don't be alarmed if it doesn't “look” like you have any novel instructions from MIPS. The novelty may be that you execute the instruction differently/more efficiently even if the name is the same.

# Dealing with Tradeoffs

- I don't have enough space for immediates!
  - I have a branch if equal instruction that looks like so:

|                 |          |          |               |
|-----------------|----------|----------|---------------|
| Opcode (4 bits) | Reg1 (2) | Reg2 (2) | Immediate (3) |
|-----------------|----------|----------|---------------|

- Well...I can only move 3 back or 3 forward with this...so that's no good.
- Solution: branch to a jmp function

|                 |               |
|-----------------|---------------|
| Opcode (4 bits) | Immediate (7) |
|-----------------|---------------|

# Dealing with Tradeoffs

Example:

```
// ...
```

```
be $s1 $s2 3
```

```
addi $s1 4
```

```
ba 2 //branch always, might be necessary
```

```
jmp -30
```

```
// ...
```

# Dealing with Tradeoffs

- I don't have enough instructions!
  - If you are doing an 11-bit ISA and you run into this problem, it isn't too late to switch.
  - If you are doing a 14-bit ISA...
    - Function codes are your friend (see MIPS R-type instructions)
    - Example:

|                 |          |          |             |
|-----------------|----------|----------|-------------|
| Opcode (5 bits) | Reg1 (3) | Reg2 (3) | Nothing (3) |
|-----------------|----------|----------|-------------|

- VS:

|                 |          |          |                   |
|-----------------|----------|----------|-------------------|
| Opcode (5 bits) | Reg1 (3) | Reg2 (3) | Function Code (3) |
|-----------------|----------|----------|-------------------|

# Project Clarifications

- Yes the data you are working with in your ISA is 34-bits wide, assume that MIPS can handle that without extra loads or instructions.
- In the last discussion we talked about the instructions *in* and *out*. There is no equivalent in MIPS (they are x86 instructions), but you can just use *in* and *out* when writing your MIPS code
  - Also, while the example for how to write *in* was “*in* [dest] [channel[3:0]] “, the only requirement is that the channel be at least 4 bits long, you can assign a nonaddressable register to this instruction if you need the space.