The FCC is about to implement a database of white spaces—unused spectrum between television channels. New devices can contact the database to see if they're allowed to broadcast. New opportunities for wireless communications!

Guest Lecturer
Sean Soleymann

I-Format Problems (0/3)

• Problem 0: Unsigned # sign-extended?
  • addiu, sltiu, sign-extends immediates to 32 bits. Thus, # is a "signed" integer.

• Rationale
  • addiu so that can add w/out overflow
    • See K&R pp. 230, 305
  • sltiu suffers so that we can have easy HW
    • Does this mean we'll get wrong answers?
      • Nope, it means assembler has to handle any unsigned immediate $2^{15} \leq n < 2^{16}$ (i.e., with a 1 in the 15th bit and 0s in the upper 2 bytes) as it does for numbers that are too large. ⇒

I-Format Problem (1/3)

• Problem:
  • Chances are that addi, lw, sw and slti will use immediates small enough to fit in the immediate field.
  • ...but what if it's too big?
  • We need a way to deal with a 32-bit immediate in any I-format instruction.

I-Format Problems (3/3)

• Solution to Problem (continued):
  • So how does lui help us?
  • Example:
    addiu $t0, $t0, 0xABABCD
    ...becomes
    lui $at, 0xABAB
    ori $at, $at, 0xCDCD
    addu $t0, $t0, $at
  • Now each I-format instruction has only a 16-bit immediate.
  • Wouldn't it be nice if the assembler would this for us automatically? (later)
Branches: PC-Relative Addressing (1/5)

- Use I-Format

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
</table>

- **opcode** specifies beq versus bne
- **rs** and **rt** specify registers to compare
- What can immediate specify?
  - immediate is only 16 bits
  - PC (Program Counter) has byte address of current instruction being executed; 32-bit pointer to memory
  - So immediate cannot specify entire address to branch to.

Branches: PC-Relative Addressing (2/5)

- How do we typically use branches?
  - Answer: if-else, while, for
  - Loops are generally small: usually up to 50 instructions
  - Function calls and unconditional jumps are done using jump instructions (j and jal), not the branches.

Branches: PC-Relative Addressing (3/5)

- Solution to branches in a 32-bit instruction: PC-Relative Addressing
- Let the 16-bit immediate field be a signed two's complement integer to be added to the PC if we take the branch.
- Now we can branch ± 2^{15} bytes from the PC, which should be enough to cover almost any loop.
- Any ideas to further optimize this?

Branches: PC-Relative Addressing (4/5)

- Note: Instructions are words, so they're word aligned (byte address is always a multiple of 4, which means it ends with 00 in binary).
  - So the number of bytes to add to the PC will always be a multiple of 4.
  - So specify the immediate in words.
- Now, we can branch ± 2^{15} words from the PC (or ± 2^{17} bytes), so we can handle loops 4 times as large.

Branches: PC-Relative Addressing (5/5)

- Branch Calculation:
  - If we don't take the branch:
    PC = PC + 4 = byte address of next instruction
  - If we do take the branch:
    PC = (PC + 4) + (immediate * 4)
- Observations
  - Immediate field specifies the number of words to jump, which is simply the number of instructions to jump.
  - Immediate field can be positive or negative.
  - Due to hardware, add immediate to (PC+4), not to PC; will be clearer why later in course

Branch Example (1/3)

MIPS Code:

```assembly
Loop: beq $9,$0,End
      addu $8,$8,$10
      addiu $9,$9,-1
      j Loop
End:
```

- beq branch is I-Format:
  - opcode = 4 (look up in table)
  - rs = 9 (first operand)
  - rt = 0 (second operand)
  - immediate = ???

Garcia, Fall 2011 © UCB
Branch Example (2/3)

- **MIPS Code:**
  ```
  Loop: beq $9,$0,End
  addu $8,$8,$10
  addiu $9,$9,-1
  j Loop
  End:
  ```

- **Immediate Field:**
  - Number of instructions to add to (or subtract from) the PC, starting at the instruction following the branch.
  - In `beq` case, immediate = 3

Branch Example (3/3)

- **MIPS Code:**
  ```
  Loop: beq $9,$0,End
  addu $8,$8,$10
  addiu $9,$9,-1
  j Loop
  End:
  ```

- **Decimal representation:**
  ```
  4 9 0 3
  ```

- **Binary representation:**
  ```
  000100 01001 00000 0000000000000011
  ```

Questions on PC-addressing

- Does the value in branch field change if we move the code?
- What do we do if destination is > $2^{15}$ instructions away from branch?
- Why do we need different addressing modes (different ways of forming a memory address)? Why not just one?

Administrivia

- **Project 1, Part 1 due Sunday, 11:59 PM**
  - Worth ~15% of total project grade
  - (Warning! Part 2 is way more time-consuming, so prepare accordingly.)
- HW1 grades now posted; HW2 should be up Monday
  - Find your reader on the course website

J-Format Instructions (1/5)

- For branches, we assumed that we won’t want to branch too far, so we can specify change in PC.
- For general jumps (`j` and `jal`), we may jump to anywhere in memory.
- Ideally, we could specify a 32-bit memory address to jump to.
- Unfortunately, we can’t fit both a 6-bit opcode and a 32-bit address into a single 32-bit word, so we compromise.

J-Format Instructions (2/5)

- **Define two “fields” of these bit widths:**
  ```
  6 bits | 26 bits
  ```

- As usual, each field has a name:
  ```
  opcode | target address
  ```

- **Key Concepts**
  - Keep opcode field identical to R-format and I-format for consistency.
  - Collapse all other fields to make room for large target address.
J-Format Instructions (3/5)

- For now, we can specify 26 bits of the 32-bit bit address.
- Optimization:
  - Note that, just like with branches, jumps will only jump to word aligned addresses, so last two bits are always 00 (in binary).
  - So let's just take this for granted and not even specify them.

J-Format Instructions (4/5)

- Now specify 28 bits of a 32-bit address
- Where do we get the other 4 bits?
  - By definition, take the 4 highest order bits from the PC.
  - Technically, this means that we cannot jump to anywhere in memory, but it's adequate 99.999...% of the time, since programs aren't that long
  - only if straddle a 256 MB boundary
  - If we absolutely need to specify a 32-bit address, we can always put it in a register and use the jr instruction.

J-Format Instructions (5/5)

- Summary:
  - New PC = { PC[31..28], target address, 00 }  
  - Understand where each part came from!
- Note: { , , } means concatenation
  - { 4 bits, 26 bits, 2 bits } = 32 bit address
  - { 1010, 11111111111111111111111111111100 } = 10101111111111111111111111111100
  - Note: Book uses ||

Peer Instruction Question

(for A,B) When combining two C files into one executable, recall we can compile them independently & then merge them together.

1) Jump insts don't require any changes.  
2) Branch insts don't require any changes.

In conclusion

- MIPS Machine Language Instruction:
  - 32 bits representing a single instruction
- Branches use PC-relative addressing, Jumps use absolute addressing.
- Disassembly is simple and starts by decoding opcode field. (more in a week)