Addresses in Branches

- Instructions:
  - `bne $t4, $t5, Label` Next instruction is at Label if $t4 ≠ $t5
  - `beq $t4, $t5, Label` Next instruction is at Label if $t4 = $t5

- Formats:
  - Could specify a register (like lw and sw) and add it to address
    - use Instruction Address Register (PC = program counter)
    - most branches are local (principle of locality)
  - Jump instructions just use high order bits of PC
    - address boundaries of 256 MB

To summarize:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bne</code></td>
<td>Branch on not equal</td>
</tr>
<tr>
<td><code>beq</code></td>
<td>Branch on equal</td>
</tr>
</tbody>
</table>

**MIPS Registers**

- `s0-s7, t0-t9, zero` Fast locations for data. In MIPS, data must be in registers to perform arithmetic.
- `a0-a3, v0-v1, gp` 32 registers
- `fp, sp, ra, at`  reserved for the assembler to handle large constants.

**MIPS Memory**

- `memory[0], ...` Sequential words differ by 4. Memory holds data structures, such as arrays, and stores saved register information.

**MIPS Addressing**

1. Immediate addressing
2. Register addressing
3. Base addressing
4. PC-relative addressing
5. Pseudodirect addressing

Alternative Architectures

- Design alternative:
  - provide more powerful operations
  - goal is to reduce number of instructions executed
  - danger is a slower cycle time and/or a higher CPI

  "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
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 75 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”

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”

Registers and Data Addressing

IA-32 Overview

• Registers in the 32-bit subset that originated with 80386

IA-32 Register Restrictions

• Registers are not “general purpose” – note the restrictions below
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)
  - JE EIP + displacement
  - CALL
  - MOV EBX, [EDI + 45]
  - PUSH ESI
  - ADD EAX, #6765
  - TEST EDX, #42

Summary

- Instruction complexity is only one variable
  - lower instruction count vs. higher CPI / lower clock rate
- Design Principles:
  - simplicity favors regularity
  - smaller is faster
  - good design demands compromise
  - make the common case fast
- Instruction set architecture
  - a very important abstraction indeed!
• Bits are just bits (no inherent meaning)
  — conventions define relationship between bits and numbers
• Binary numbers (base 2)
  0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  decimal: \(0 \ldots 2^n - 1\)
• Of course it gets more complicated:
  numbers are finite (overflow)
  fractions and real numbers
  negative numbers
  e.g., no MIPS subi instruction; addi can add a negative number
• How do we represent negative numbers?
  i.e., which bit patterns will represent which numbers?

- Sign Magnitude:         One’s Complement     Two’s Complement
  0000 = +0 0000 = +0 0000 = +0
  0001 = +1 0001 = +1 0001 = +1
  0100 = +2 0100 = +2 0100 = +2
  0111 = +3 0111 = +3 0111 = +3
  1000 = -0 1000 = -3 1000 = -4
  1001 = -1 1001 = -2 1001 = -3
  1100 = -2 1100 = -1 1100 = -2
  1111 = -3 1111 = -0 1111 = -1

- Issues: balance, number of zeros, ease of operations
- Which one is best? Why?

- 32 bit signed numbers:
  0000 0000 0000 0000 0000 0000 0000 0000 = 0
  0000 0000 0000 0000 0000 0000 0000 0001 = +1
  0000 0000 0000 0000 0000 0000 0000 0010 = +2
  ... 0111 0111 0111 0111 0111 0111 0111 0111 = +2,147,483,646
  0000 0000 0000 0000 0000 0000 0000 0000 = -0
  0000 0000 0000 0000 0000 0000 0000 0001 = -1
  0000 0000 0000 0000 0000 0000 0000 0010 = -2
  ... 0111 0111 0111 0111 0111 0111 0111 0111 = -2,147,483,647

- Negating a two’s complement number: invert all bits and add 1
  remember: “negate” and “invert” are quite different!
- Converting n bit numbers into numbers with more than n bits:
  - MIPS 16 bit immediate gets converted to 32 bits for arithmetic
  - copy the most significant bit (the sign bit) into the other bits
  
  0010 -> 0000 0010
  1010 -> 1111 1010
  “sign extension” (lbu vs. lb)
Addition & Subtraction

• Just like in grade school (carry/borrow 1s)
  \[
  \begin{array}{c}
  0111 \\
  + 0110 \\
  \hline
  10001
  \end{array}
  \]

• Two's complement operations easy
  – subtraction using addition of negative numbers
  \[
  \begin{array}{c}
  0111 \\
  + 1010 \\
  \hline
  0001
  \end{array}
  \]

• Overflow (result too large for finite computer word):
  – e.g., adding two n-bit numbers does not yield an n-bit number
  \[
  \begin{array}{c}
  0111 \\
  + 0001 \\
  \hline
  1000 \text{ note that overflow term is somewhat misleading,}
  \end{array}
  \]
  \[
  \begin{array}{c}
  \text{it does not mean a carry “overflowed”}
  \end{array}
  \]