Registers

September 27, 2000

CSC201 Section 002

Fall, 2000

Temporary Storage Locations in the CPU

- CPU: ALU + Control Unit + Registers
- Primary advantage is improving program execution speed
  - Register access time: 1-2 ns
  - Memory access time: 20-100 ns
- Secondary advantage: shorter, simpler instruction formats
  - Register address size: 4-8 bits
  - Memory address size: 32-64 bits

Address Size (Number of Bits)

- Memory addresses are long (32 bits or greater)
- Registers are a small set of storage locations; fewer bits for address
  - Example: 64 registers --> log_2(64) = 6 bit addresses
- So, for a 3-address instruction
  - with memory operands, 3*32 = 96 bits + opcode size
  - with register operands, 3*6 = 18 bits + opcode size
- Shorter instructions
  - smaller code size, less memory needed
  - fewer memory fetches when executing the program

Register Space

- 64 registers wouldn't store a lot of data!
- So, these are just "temporary" storage locations
  - there must be a way to move data between registers and memory
  - extra instructions required to do this copying!
- LOAD instruction -- copy from memory into a register
- STORE instruction -- copy from a register into memory
- If only load/store instructions can reference memory, it's called a LOAD/STORE architecture

Load/Store Architectures

- Most common design today!
- Easier to optimize, or pipeline (more later?)
- The Pentium is an exception in today's world; *not* a load/store architecture

Number of Operands Per Instruction

- Fewer operands -- shorter, simpler instructions; easier to optimize?
- More operands -- more powerful instructions, fewer needed
- Most (current) architectures: 2 or 3 operands per instruction
  - Pentium architecture: 2 operands

3-Address Architectures

- Most modern machines allow 3 operands, e.g.
  - add x, y, z ; x <- (y) + (z)
- Or (register-based)
  - load reg10, y
  - load reg11, z
  - add reg12, reg11, reg10
  - store x, reg12
- Well-suited for binary operations, which are the most common

2-Address Architecture

- One operand is "reused": both a source and a destination
- 2-address requires more instructions than 3-address
  - move x, y ; 2-address instruction set
  - add x, z ; x is both a source and a destination

1-address Architecture

- If there's only 1 operand, how can we do binary operations?
- The destination is *implicit*, and one of the inputs is *implicit*
- Where is this implicit operand? In a special register called the "accumulator"
- So, for the add...
  - load z ; put z into the accumulator
  - add y ; add y to the contents of the accumulator
  - ; accumulator is both a source and destination operand
  - store x ; store the accumulator value into operand x

0-Address Architecture (!)

- No operands -- everything implicit!
- Obviously, there has to be some instruction that references a memory location
  - load x ; copy operand x from memory into a special register
  - store x ; copy special register contents into memory
- All other operands are in a special set of registers called the stack
- Binary operations: implicit inputs are top 2 elements on the stack, result stored as top element of stack
- Unary operations: implicit input and output is top element of stack
- We'll look at stacks later...

Finally...

- Once a computer has registers (and they ALL do!), then there can be lots of interesting uses of these registers