Lecture 3

now with 100% less beard
Announcements

Homework due tonight! (11:55 pm)
dropbox.cse.sc.edu
Example from class is on website (example1.c)
Late policy

5% for the first 2 days late
10% for the next 3 days late
Not excepted after 5 days (unless you talk to me about it)
Time to start MIPS!
What is a register?

- A register is a small amount of storage built into the CPU.
- The arithmetic and logic unit (ALU) uses this very fast storage to hold data it is working on:
  - Other things use registers, but for now we only care about the ALU.
- Data gets pulled into the registers for work, the pushed back out when it is finished.
Here is a memory hierarchy
You have seen these!

- When we were playing with C code last class you would have seen some registers.
- In the assembly code that was generated we saw these, we will look at these again now.
MIPS: 3 address, reg to reg

- MIPS uses three-address instructions for data manipulation
  - First address = destination
  - Next two = sources
- MIPS is a register to register architecture
  - This means Destination and source must be registers
  - Special instructions required to access main memory
Registers in MIPS

- Arithmetic instructions use register operands
- MIPS has 32 registers, each with 32 bits
  - Numbered 0-31
  - With 32 registers it takes 5 bits to reference all 32 ($2^5 = 32$)
- Assembler names for registers
  - $t0, t1, ..., t9$ for temporary variables
  - $s0, s1, ..., s7$ for saved variables
Main memory is used for composite data
- Dynamic data, arrays, objects

To apply arithmetic (use the ALU)
- Load values from memory into registers
- Store result from register into memory

Memory is byte addressed
- Each address identifies a byte (8 bits)
- There are $2^{32}$ memory bytes in MIPS (32 bit)
  - memory[0], memory[1],...,memory[4294967295]
The registers are 32 bits, so 32 bit data is called a “word”

Words are aligned in memory
  - 1 word is 4 bytes (32 bits), so each address must be a multiple of 4
  - memory[0], memory[4],...,memory[4294967292]
Storing some data

We have the 32 bit word 0x1b2d3ac4
How do we store this into our memory?
Well, there are 2 ways

- Big Endian (this is what MIPS is)
  - Most significant (leftmost) byte gets stored at smallest address for word
    - 0x1b2d3ac4
    - memory[0] = 0x1b
    - memory[1] = 0x2d
    - memory[2] = 0x3a
    - memory[3] = 0xc4
Second way, Little Endian

- For this one we do the reverse, least significant byte in smallest address (x86)
  - 0x1b2d3ac4
    - memory[0] = 0xc4
    - memory[1] = 0x3a
    - memory[2] = 0x2d
    - memory[3] = 0x1b
Example

- 0x56c4a901
- The first 5 words are already filled
- How would this look in memory for:
  - Big Endian?
  - Little Endian?
Pop Quiz

- You can use your notes
- 0x00112233
- What does the first word look like for big endian?
  - memory[0] = ?
  - memory[1] = ?
  - memory[2] = ?
  - memory[3] = ?
Pass ‘em up

Quiz 1 grading rubric:
  80%: Is your name on the paper?
  20%: Is the correct answer on the paper?
Arithmetic Operations

- Add and subtract, 3 operands
  - Remember, 3 registers, 1 dest., 2 source
- `add a, b, c`
  - put `b + c` into `a`
  - `a = (b + c);`
- All arithmetic operations have this form
Example

- Some C code:
  - `a = (b + c) - (d + e);`
- `a, b, c, d, and e` are assigned to `$s0, $s1, $s2, $s3, and $s4` respectively
- Compiled MIPS code:
  - `add $t0, $s1, $s2  # temp t0 = b+c`
  - `add $t1, $s3, $s4  # temp t1 = d+e`
  - `sub $s0, $t0, $t1  # a = t0 - t1`
Memory Operands

- How does MIPS handle arrays?
- C code:
  - `a = b + C[8];`
    - `a` is in `$s0`, `b` is in `$s1`, `C`’s base address is `$s2`
- MIPS code
  - Index of 8, 4 bytes per word, $8 \times 4 = 32$ bytes offset
    - `lw $t0, 32($s2)`  # load word
    - `add $s0, $s1, $t0`
Memory Operands Example

- Convert the following C code to MIPS
What about this?

- C code:
    - b in $s0, A in $s1
C to MIPS


- lw $t0, 16($s1)  # 4 * 4 bytes = 16 offset
- add $t0, $s0, $t0  # add, store in $t0
- sw $t0, 32($s1)  # store in 32/4 = 8
lw and sw

- **Load word:**
  - lw (register dest), (memory source)
- **Store word**
  - sw (register source), (memory dest)
Immediate Operands

- Want to add constants?
  - addi $s2, $s2, 4
- Want to subtract constants?
  - addi $s2, $s2, -4
Constant Zero

- MIPS has a register saved for the constant 0
  - $zero
- Useful for common operations
  - moving between registers
    - add $t1, $s2, $zero