Levels of Representation/Interpretation

- **High Level Language:** Program (e.g., C)
  - Compiler
  - Intermediate Language Program (e.g., MIPS)
  - Machine Language Program (MIPS)
- **Machine Interpretation**
  - **Hardware Architecture Description (e.g., block diagrams)**
  - **Logic Circuit Description (Circuit Schematic Diagrams)**

The Language a Computer Understands

- **Why not all the same? Why not all different?**
  - Single ISA (to rule them all):
    - Leverage common compilers, operating systems, etc.
    - BUT fairly easy to retarget these for different ISAs (e.g., Linux, gcc)
  - Multiple ISAs:
    - Specialized instructions for specialized applications
    - Different tradeoffs in resources used (e.g., functionality, memory demands, complexity, power consumption, etc.)
    - Competition and innovation is good, especially in emerging environments (e.g., mobile devices)

MIPS: Instruction Set for CS 61C

- MIPS is a real-world ISA (see [www.mips.com](http://www.mips.com))
  - Standard instruction set for networking equipment
  - Was also used in original Nintendo-64!
  - Elegant example of a Reduced Instruction Set Computer (RISC) instruction set
  - Invented by John Hennessy @ Stanford
    - Why not Berkeley/Sun RISC invented by Dave Patterson? Ask him!
RISC Design Principles

- Basic RISC principle: "A simpler CPU (the hardware that interprets machine language) is a faster CPU" (CPU → Core)
- Focus of the RISC design is reduction of the number and complexity of instructions in the ISA
- A number of the more common strategies include:
  - Fixed instruction length, generally a single word; Simplifies process of fetching instructions from memory
  - Simplified addressing modes; Simplifies process of fetching operands from memory
  - Fewer and simpler instructions in the instruction set; Simplifies process of executing instructions
  - Only load and store instructions access memory; E.g., no add memory to register, add memory to memory, etc.
  - Let the compiler do it. Use a good compiler to break complex high-level language statements into a number of simple assembly language statements

Mainstream ISAs

- ARM (Advanced RISC Machine) is most popular RISC
  - In every smart phone-like device (e.g., iPhone, iPad, iPod, ...)
- Intel 80x86 is another popular ISA and is used in Macbook and PCs (Core i3, Core i5, Core i7, ...)
  - x86 is a Complex Instruction Set Computer (CISC)
  - 20x ARM sold vs. 80x86 (i.e., 5 billion vs. 0.3 billion)

MIPS Green Card

Inspired by the IBM 360 “Green Card”

MIPS Instructions

- Every computer does arithmetic
- Instruct a computer to do addition:
  \[
  \text{add } a, b, c
  \]
  - Add \(b\) to \(c\) and put sum into \(a\)
- 3 operands: 2 sources + 1 destination for sum
- One operation per MIPS instruction
- How do you write the same operation in C?
Guess More MIPS instructions

• Subtract \( c \) from \( b \) and put difference in \( a \)?
  \[ \text{sub } a, b, c \]

• Multiply \( b \) by \( c \) and put product in \( a \)?
  \[ \text{mul } a, b, c \]

• Divide \( b \) by \( c \) and put quotient in \( a \)?
  \[ \text{div } a, b, c \]

Example Instructions

• MIPS instructions are inflexible, rigid:
  – Just one arithmetic operation per instruction
  – Always with three operands
• How to write this C expression in MIPS?
  \[ a = b + c + d + e \]
  \[ a = b + c + d + e; \]
  \[ \text{add } a, b, c \]
  \[ \text{add } t1, d, e \]
  \[ \text{add } t2, c, t1 \]
  \[ \text{add } a, b, t2 \]

Comments in MIPS

• Can add comments to MIPS instruction by putting # that continues to end of line of text
  \[ \text{add } a, b, c \]
  \[ \text{add } a, b, c \# b + c \]
  \[ \text{is placed in } a \]
  \[ \text{add } a, b, d \]
  \[ \text{# } b + c + d \]
  \[ \text{is now in } a \]
  \[ \text{add } a, e \]
  \[ \text{# } b + c + d + e \]
  \[ \text{is in } a \]
• Are extremely useful in assembly code!

C to MIPS

• What is MIPS code that performs same as?
  \[ a = b + c; \]
  \[ \text{add } a, b, c \]
  \[ d = a - e; \]
  \[ \text{sub } d, a, e \]

• What is MIPS code that performs same as?
  \[ f = (g + h) - (i + j); \]
  \[ \text{add } t1, i, j \]
  \[ \text{add } t2, g, h \]
  \[ \text{sub } f, t2, t1 \]

For a given function, which programming language likely takes the most lines of code? (most to least)

- Scheme, MIPS, C
- C, Scheme, MIPS
- MIPS, Scheme, C
- MIPS, C, Scheme
Computer Hardware Operands

- High-Level Programming languages: could have millions of variables
- Instruction sets have fixed, small number
- Called registers
  - “Bricks” of computer hardware
  - Fastest way to store data in computer hardware
  - Visible to (the “assembly language”) programmer
- MIPS Instruction Set has 32 integer registers

Why Just 32 Registers?

- RISC Design Principle: Smaller is faster
  - But you can be too small …
- Hardware would likely be slower with 64, 128, or 256 registers
- 32 is enough for compiler to translate typical C programs, and not run out of registers very often
  - ARM instruction set has only 16 registers
  - May be faster, but compiler may run out of registers too often (aka “spilling registers to memory”)

Names of MIPS Registers

- For registers that hold programmer variables: $s0, s1, s2, …
- For registers that hold temporary variables: $t0, t1, t2, …

Names of MIPS Registers

- Suppose variables f, g, h, i, and j are assigned to the registers $s0, s1, s2, s3, and s4, respectively. What is MIPS for f = (g + h) - (i + j);
  - add $t1, $s3, $s4
  - add $t2, $s1, $s2
  - sub $s0, $t2, $t1

Size of Registers

- Bit is the atom of Computer Hardware: contains either 0 or 1
  - True “alphabet” of computer hardware is 0, 1
  - Will eventually express MIPS instructions as combinations of 0s and 1s (in Machine Language)
- MIPS registers are 32 bits wide
- MIPS calls this quantity a word
  - Some computers use 16-bit or 64-bit wide words
  - E.g., Intel 8086 (16-bit), MIPS64 (64-bit)

Administrivia

- If on wait list, have to add class by Monday or will be dropped from wait list
  - 4 lab sections have open space (including W 9-11pm)
Data Structures vs. Simple Variables

• In addition to registers, a computer also has memory that holds millions/billions of words.
• Memory is a single dimension array, starting at 0.
• To access memory, need an address (like an array index).
• But MIPS arithmetic instructions only operate on registers!
• Solution: instructions specialized to transfer words (data) between memory and registers.
• Called data transfer instructions.

Memory Addresses are in Bytes

• Lots of data is smaller than 32 bits, but rarely smaller than 8 bits – works fine if everything is a multiple of 8 bits.
• 8-bit item is called a byte (1 word = 4 bytes).
• Memory addresses are really in bytes, not words.
• Word addresses are 4 bytes apart.
  – Word address is same as leftmost byte.

Transfer from Memory to Register

• MIPS instruction: Load Word, abbreviated lw.
• Assume A is an array of 100 words, variables g and h map to registers $s1 and $s2, the starting address/base address of the array A is in $s3.
• int A[100];
  g = h + A[3];
• Becomes:
  lw $t0, 3($s3) # Temp reg $t0 gets A[3]
  add $s1, $s2, $t0 # g = h + A[3]

Transfer from Register to Memory

• MIPS instruction: Store Word, abbreviated sw.
• Assume A is an array of 100 words, variables g and h map to registers $s1 and $s2, the starting address, or base address, of the array A is in $s3.
• Turns into:
  lw $t0, 12($s3) # Temp reg $t0 gets A[3]
  add $t0, $s2, $t0 # t0 = h + A[3]
  add $s1, $s2, $t0 # A[10] = h + A[3]
Transfer from Register to Memory

- MIPS instruction: Store Word, abbreviated sw
- Assume A is an array of 100 words, variables g and h map to registers $s1$ and $s2$, the starting address, or base address, of the array A is in $s3$
- Turns into
  \[
  \begin{align*}
  &\text{lw } $t0,12($s3) \ # \text{Temp reg } t0 \text{ gets } A[3] \\
  &\text{add } $t0,$s2,$t0 \ # \ t0 = h + A[3] \\
  &\text{sw } $t0,40($s3) \ # \ A[10] = h + A[3]
  \end{align*}
  \]

Speed of Registers vs. Memory

- Given that
  - Registers: 32 words (128 Bytes)
  - Memory: Billions of bytes (2 GB to 8 GB on laptop)
- and the RISC principle is...
  - Smaller is faster
- How much faster are registers than memory??
  - About 100-500 times faster!
    - in terms of latency of one access

Which of the following is TRUE?

- add $t0,$t1,4($t2) is valid MIPS
- Can byte address 8GB with a MIPS word
- $s0+imm$ must be a multiple of 4 for lw $t0,imm($s0) to be valid
- If MIPS halved the number of registers available, it would be twice as fast

Example If Statement

- Assuming translations below, compile if block
  \[
  \begin{align*}
  &f \rightarrow $s0 \quad g \rightarrow $s1 \quad h \rightarrow $s2 \\
  &i \rightarrow $s3 \quad j \rightarrow $s4
  \end{align*}
  \]
  \[
  \text{if (i == j)} \quad \text{bne } $s3,$s4,\text{Exit}
  \]
  \[
  f = g + h; \quad \text{add } $s0,$s1,$s2
  \]
  \[
  \text{Exit:}
  \]
  - May need to negate branch condition

Computer Decision Making

- Based on computation, do something different
- In programming languages: if-statement
  - Sometimes combined with gotos and labels
- MIPS: if-statement instruction is
  \[
  \text{beq register1,register2,L1}
  \]
  means go to statement labeled L1 if (value in register1) = (value in register2).....otherwise, go to next statement
  - beq stands for branch if equal
  - Other instruction: bne for branch if not equal

Types of Branches

- Branch – change of control flow
- Conditional Branch – change control flow depending on outcome of comparison
  - branch if equal (beq) or branch if not equal (bne)
- Unconditional Branch – always branch
  - a MIPS instruction for this: jump (j)
Making Decisions in MIPS

- Assuming translations below, compile
  
  \[
  \begin{align*}
  f & \rightarrow s0 \quad g \rightarrow s1 \quad h \rightarrow s2 \\
  i & \rightarrow s3 \quad j \rightarrow s4 \\
  \text{if (i == j)} & \quad \text{bne } s3, s4, \text{Else} \\
  f & = g + h; \quad \text{add } s0, s1, s2 \\
  \text{else} & \quad j \text{ Exit} \\
  f & = g - h; \quad \text{Else: sub } s0, s1, s2 \\
  \text{Exit:} & 
  \end{align*}
  \]

Which of the following is FALSE?

- Can make an unconditional branch from a conditional branch instruction
- Can make a “for” loop with only j
- Can make a “for” loop with only beq
- Code for “else” part can come before code for “then” in translation of “if”

And In Conclusion ...

- Computer words and vocabulary are called instructions and instruction set respectively
- MIPS is example RISC instruction set in this class
- Rigid format: 1 operation, 2 source operands, 1 destination
  - add, sub, mul, div, and, or, sll, srl
  - lw, sw to move data to/from registers from/to memory
- Simple mappings from arithmetic expressions, array access, if-then-else in C to MIPS instructions