Overview

- C operators, operands
- Variables in Assembly: Registers
- Comments in Assembly
- Addition and Subtraction in Assembly
- Memory Access in Assembly
Assembly Language

- Basic job of a CPU: execute lots of *instructions*
- Instructions are the primitive operations that the CPU may execute.
- Different CPUs implement different sets of instructions. The set of instructions a particular CPU implements is an *Instruction Set Architecture (ISA)*
  - Examples: Intel 80x86 (Pentium 4), IBM/Motorola PowerPC (Macintosh), MIPS, Intel IA64, ...
Instruction Set Architectures

- Early trend was to add more and more instructions to new CPUs to do elaborate operations
  - VAX architecture had an instruction to multiply polynomials!
- RISC philosophy (Cocke IBM, Patterson, Hennessy, 1980s) – Reduced Instruction Set Computing
  - Keep the instruction set small and simple, makes it easier to build fast hardware.
  - Let software (compiler) do complicated operations by composing simpler ones.
MIPS Architecture

- MIPS – semiconductor company that built one of the first commercial RISC architectures
- We will study the MIPS architecture in some detail in this class
- Why MIPS instead of Intel 80x86?
  - MIPS is simple, elegant. Don’t want to get bogged down in gritty details.
  - MIPS widely used in embedded apps
  - There are more embedded computers than PCs
Assembly Variables: Registers

- Unlike HLL like C or Java, assembly cannot use variables
  - Why not? Keep Hardware Simple
- Assembly Operands are registers
  - limited number of special locations built directly into the hardware
  - operations can only be performed on these!
- Benefit: Since registers are directly in hardware, they are very fast
  (faster than 1 billionth of a second)
Assembly Variables: Registers

- Drawback: Since registers are in hardware, there are a predetermined number of them
  - Solution: MIPS code must be very carefully put together to efficiently use registers

- 32 registers in MIPS
  - Why 32? Smaller is faster

- Each MIPS register is 32 bits wide
  - Groups of 32 bits called a word in MIPS
Assembly Variables

- Registers are numbered from 0 to 31
- Each register can be referred to by number or name
- Number references:
  $0, \; $1, \; $2, \; \ldots \; $30, \; $31
Assembly Variables: Registers

- By convention, each register also has a name to make it easier to code
- For now:
  - $16 - $23 → $s0 - $s7
    (correspond to C variables)
  - $8 - $15 → $t0 - $t7
    (correspond to temporary variables)
  - Later will explain other 16 register names
- In general, use names to make your code more readable
In C (and most High Level Languages) variables declared first and given a type

Example:
```
int fahr, celsius;
char a, b, c, d, e;
```

Each variable can ONLY represent a value of the type it was declared as (cannot mix and match `int` and `char` variables).

In Assembly Language, the registers have no type; operation determines how register contents are treated.
Comments in Assembly

- Another way to make your code more readable: comments!
- Hash (#) is used for MIPS comments
  - anything from hash mark to end of line is a comment and will be ignored
- Note: Different from C
  - C comments have format /* comment */, so they can span many lines
Assembly Instructions

- In assembly language, each statement (called an Instruction), executes exactly one of a short list of simple commands.
- Unlike in C (and most other High Level Languages), each line of assembly code contains at most 1 instruction.
- Instructions are related to operations (=, +, -, *, /) in C or Java.
MIPS Addition and Subtraction

- Syntax of Instructions:
  1  2, 3, 4
  where:
  1) operation by name
  2) operand getting result ("destination")
  3) 1st operand for operation ("source1")
  4) 2nd operand for operation ("source2")

- Syntax is rigid:
  - 1 operator, 3 operands
  - Why? Keep Hardware simple via regularity
Addition and Subtraction of Integers

❖ Addition in Assembly
❖ Example: \texttt{add $s0, $s1, $s2} \ (\text{in MIPS})
   Equivalent to: \texttt{a = b + c} \ (\text{in C})
where MIPS registers $s0, s1, s2$ are associated with C variables \texttt{a, b, c}

❖ Subtraction in Assembly
❖ Example: \texttt{sub $s3, $s4, $s5} \ (\text{in MIPS})
   Equivalent to: \texttt{d = e - f} \ (\text{in C})
where MIPS registers $s3, s4, s5$ are associated with C variables \texttt{d, e, f}
How do the following C statement?
\[ a = b + c + d - e; \]

Break into multiple instructions
- `add $t0, $s1, $s2 # temp = b + c`
- `add $t0, $t0, $s3 # temp = temp + d`
- `sub $s0, $t0, $s4 # a = temp - e`

Notice: A single line of C may break up into several lines of MIPS.

Notice: Everything after the hash mark on each line is ignored (comments)
Addition and Subtraction of Integers

- How do we do this?
  \[ f = (g + h) - (i + j); \]

- Use intermediate temporary register
  ```
  add $t0,$s1,$s2      # temp = g + h
  add $t1,$s3,$s4      # temp = i + j
  sub $s0,$t0,$t1      # f=(g+h)-(i+j)
  ```
Register Zero

- One particular immediate, the number zero (0), appears very often in code.
- So we define register zero ($0 or $zero) to always have the value 0; e.g.
  - \texttt{add} $s0, s1, \$zero \ (\text{in MIPS})
  - \texttt{f = g} \ (\text{in C})
  - where MIPS registers \$s0, \$s1 are associated with C variables \texttt{f, g}

- defined in hardware, so an instruction
  - \texttt{addi} \$zero, $zero, 5

will not do anything!
Immediates

- Immediates are numerical constants.
- They appear often in code, so there are special instructions for them.

- Add Immediate:
  \[
  \text{addi } \$s0, \$s1, 10 \quad (\text{in MIPS})
  \]
  \[
  f = g + 10 \quad (\text{in C})
  \]
  where MIPS registers \$s0, \$s1 are associated with C variables \(f, \ g\)

- Syntax similar to \text{add} instruction, except that last argument is a number instead of a register.
Immediates

- There is no Subtract Immediate in MIPS: Why?
- Limit types of operations that can be done to absolute minimum
  - if an operation can be decomposed into a simpler operation, don’t include it
  - \( \text{addi } \ldots, -X = \text{ subi } \ldots, X \Rightarrow \text{ so no subi} \)
  - \( \text{addi } \$s0, \$s1, -10 \) (in MIPS)
    \( f = g - 10 \) (in C)
where MIPS registers \( \$s0, \$s1 \) are associated with C variables \( f, g \)
Integer Multiplication

- Paper and pencil example (unsigned):
  - **Multiplicand**: 1000 8
  - **Multiplier**: x1001 9
  - 1000
  - 0000
  - 0000
  - 0000
  - +1000
  - 01001000

- \( m \) bits \( \times \) \( n \) bits = \( m + n \) bit product
Multiplication

- In MIPS, we multiply registers, so:
  - 32-bit value \times 32-bit value = 64-bit value

- Syntax of Multiplication (signed):
  - \texttt{mult register1, register2}
  - Multiplies 32-bit values in those registers & puts 64-bit product in special result regs:
    - puts product \texttt{upper half in hi, lower half in lo}
  - \texttt{hi} and \texttt{lo} are 2 registers separate from the 32 general purpose registers
  - Use \texttt{mfhi register} & \texttt{mflo register} to move from \texttt{hi, lo} to another register
Multiplication

Example:
- in C: \( a = b \times c; \)
- in MIPS:
  - let b be $s2; let c be $s3; and let a be $s0 and $s1 (since it may be up to 64 bits)
    \[
    \begin{align*}
    \text{mult } & \texttt{$s2,$s3} \quad \# b*c \\
    \text{mfhi } & \texttt{$s0} \quad \# \text{upper half of} \\
    \text{mflo } & \texttt{$s1} \quad \# \text{lower half of}
    \end{align*}
    \]
  - \# product into $s0 \# product into $s1

Note: Often, we only care about the lower half of the product.
Integer Division

- Paper and pencil example (unsigned):

\[
\begin{array}{c|c}
\text{Divisor} & \text{1001} \\
\text{Dividend} & 10001010 \\
\hline
\text{Quotient} & 1001 \\
\text{Remainder} & 10
\end{array}
\]

\[\text{Dividend} = \text{Quotient} \times \text{Divisor} + \text{Remainder}\]
Division

- Syntax of Division (signed):
  - `div register1, register2`
  - Divides 32-bit register 1 by 32-bit register 2:
  - Puts remainder of division in `hi`, quotient in `lo`

- Implements C division (`/`) and modulo (`%`)

- Example in C:
  ```c
  a = c / d;
b = c % d;
  ```

- In MIPS:
  ```
a ↔ $s0; b ↔ $s1; c ↔ $s2; d ↔ $s3
  div $s2, $s3  # lo=c/d, hi=c%d
  mflo $s0     # get quotient
  mfhi $s1     # get remainder
  ```
Conclusion

- In MIPS Assembly Language:
  - Registers replace C variables
  - One Instruction (simple operation) per line
  - Simpler is Better
  - Smaller is Faster

- Instructions so far:
  - add, addi, sub, mult, div

- Registers so far:
  - C Variables: $s0 - $s7
  - Temporary Variables: $t0 - $t9
  - Zero: $zero