Today’s class

- More assembly language programming
Data transfer instructions

- `lw` – load word from memory into a register (note that the address must be a word address, divisible by 4)
- `lb` – load byte
- `sw` – store word into memory from a register (address must be for a word)
- `sb` – store byte
Addition (signed)

- \texttt{add reg\textsubscript{dest},reg\textsubscript{src\textsubscript{1}},reg\textsubscript{src\textsubscript{2}}}
  - Performs \( \text{reg\textsubscript{dest}} = \text{reg\textsubscript{src\textsubscript{1}}} + \text{reg\textsubscript{src\textsubscript{2}}} \)
  - Uses register addressing (no memory reference)

- \texttt{addi reg\textsubscript{dest},reg\textsubscript{src},const}
  - Performs \( \text{reg\textsubscript{dest}} = \text{reg\textsubscript{src}} + \text{const} \)
  - Add immediate, because const is part of instruction (no memory reference)

- \texttt{add reg\textsubscript{dest},reg\textsubscript{src},const}
  - Will generate an \texttt{addi} instruction by the assembler

- \texttt{add reg\textsubscript{dest},const}
  - Will generate \texttt{addi reg\textsubscript{dest},reg\textsubscript{dest},const}
Addition (unsigned)

- `addu reg_dest, reg_src1, reg_src2`
  - Performs `reg_dest = reg_src1 + reg_src2`
- `addiu reg_dest, reg_src, const`
  - Performs `reg_dest = reg_src + const`
- `addu reg_dest, reg_src, const`
  - Will generate an `addiu` instruction by the assembler
- `addu reg_dest, const`
  - Will generate `addiu reg_dest, reg_dest, const`
Subtraction

- **`sub reg_{dest}, reg_{src1}, reg_{src2}`**
  - Performs $\text{reg}_{\text{dest}} = \text{reg}_{\text{src1}} - \text{reg}_{\text{src2}}$
  - This is a signed subtraction

- **`subu reg_{dest}, reg_{src1}, reg_{src2}`**
  - Performs $\text{reg}_{\text{dest}} = \text{reg}_{\text{src1}} - \text{reg}_{\text{src2}}$
  - This is an unsigned subtraction

- **No `subi` instruction**
  - Can do an `addi` with the negative of what you want to subtract
Multiplication

- Result of multiplying two 32-bit numbers can be 64 bits big.
- Have two special registers called `lo` and `hi` to hold the two 32-bit parts of the result.
- \texttt{mult \ reg_{src1},reg_{src2}} will put the low 32 bits of the product in `lo` and the high 32 bits in `hi`.
- Use \texttt{mflo \ reg_{dest}} (move from `lo`) to get the low 32 bits into a register.
- Use \texttt{mfhi \ reg_{dest}} (move from `hi`) to get the high 32 bits into a register.
- The pseudo-instruction \texttt{mul \ reg_{dest},reg_{src1},reg_{src2}} will get the low 32 bits of the product into `reg_{dest}`.
Division

- \texttt{div \text{reg}_{\text{src}1},\text{reg}_{\text{src}2}} will put the integer quotient of \text{reg}_{\text{src}1}/\text{reg}_{\text{src}2} in \text{lo} and the remainder in \text{hi}

- The pseudo-instruction \texttt{div \text{reg}_{\text{dest}},\text{reg}_{\text{src}1},\text{reg}_{\text{src}2}} will get the integer quotient into \text{reg}_{\text{dest}}
Example program

- Program temperature.asm converts Celsius temperatures to Fahrenheit
- Study it and demo it
In-class exercise

- Modify the temperature conversion program so it converts from Fahrenheit to Celsius instead.

\[ \degree Celsius = \frac{5}{9}(\degree Fahrenheit - 32) \]
Unconditional branching

- `j addr` – jumps to the indicated address and continues execution from there
- `jr reg` – jumps to the address stored in the specified register and continues execution from there
Conditional branching

- \( \text{beq\ } \text{reg}_{\text{src1}}, \text{reg}_{\text{src2}}, \text{addr} \) – branch to addr if \( \text{reg}_{\text{src1}} \) equals \( \text{reg}_{\text{src2}} \)
- \( \text{beqz\ } \text{reg}, \text{addr} \) – branch to addr if \( \text{reg} \) equals 0
- \( \text{bne\ } \text{reg}_{\text{src1}}, \text{reg}_{\text{src2}}, \text{addr} \) – branch to addr if \( \text{reg}_{\text{src1}} \) does not equal \( \text{reg}_{\text{src2}} \)
- \( \text{blt\ } \text{reg}_{\text{src1}}, \text{reg}_{\text{src2}}, \text{addr} \) – branch to addr if \( \text{reg}_{\text{src1}} \) is less than \( \text{reg}_{\text{src2}} \)

The assembler will let the second operand be a constant instead of a register if desired.

There are many more – see Appendix B (page 146) of Waldron.
Example programs

- Program length.asm prints out the length of a character string
- Program replace.asm replaces lower case ‘a’ with upper case ‘A’ in a string
- Study them and demo them
In-class exercise

- Write a program to count how many times the letter ‘e’ appears in a string.