ECE 251: Computer Architecture

Week 03 Notes - Assembly, Memory, and Procedures

Prof Rob Marano

Spring 2026

1. Translating Assembly to Machine Code

- **The Stored-Program Concept**: Instructions are ultimately just 32-bit binary numbers stored in memory.
- **R-Type (Register Format)**: Used for pure arithmetic (`add`, `sub`, `and`).
  - `[op (6), rs (5), rt (5), rd (5), shamt (5), funct (6)]`
- **I-Type (Immediate Format)**: Used for data transfer and constants (`lw`, `sw`, `addi`).
  - `[op (6), rs (5), rt (5), constant/address (16)]`
- **J-Type (Jump Format)**: Used for long-distance control flow jumps (`j`, `jal`).
  - `[op (6), target address (26)]`

2. The von Neumann Memory Map

A modern general-purpose computer compartmentalizes its memory layout:

1. **Text Segment**: Read-only segment housing the compiled machine code (instructions).
2. **Data Segment**: Houses global variables and static data allocated before runtime.
3. **Heap**: Dynamic memory allocated during runtime (e.g. `malloc`). Grows upwards.
4. **Stack**: Dynamic memory used for local variables and procedure calls. Grows downwards.

MIPS32 Memory Scale

A 32-bit architecture supports 2^{32} byte-addresses. $\frac{2^{32}\text{ bytes}}{2^{30}\text{ bytes/GB}} = \textbf{4 Gigabytes}$ total addressable space.

3. Making Decisions (Control Flow)

CPUs must make choices to branch, loop, and execute code conditionally.

- **Conditional Branches**:
  - `beq $t1, $t2, Target`: Branch if Equal.
  - `bne $t1, $t2, Target`: Branch if Not Equal.
- **Inequalities** $(<, >)$:
  - MIPS does not have a dedicated *branch-less-than* instruction.
  - It uses `slt` (Set Less Than) to toggle a boolean flag register, which is then verified by a subsequent `bne` / `beq`.
- **Delayed Branching**: The instruction *immediately* following a branch is always executed while the target is calculated (Pipeline optimization).

4. Supporting Procedures (Functions)

Procedures facilitate code abstraction and reuse without infinite copy-pasting.

- **Jump and Link (`jal`)**: Jumps to the procedure's address while saving the *Return Address* (`PC + 4`) into register `$ra`.
- **Jump Register (`jr $ra`)**: Used at the end of a procedure to jump exactly back to the caller.
- **The Stack Pointer (`$sp`)**:
  - If a procedure needs more registers than are available, it must "spill" existing register data onto the Stack in memory.
  - Shrink the stack (allocate space) by subtracting 4-byte words from `$sp`.
  - Restore the registers from memory, then grow the stack back up by adding to `$sp` before calling `jr $ra`.