Programming a Computer

Programming

- The process of creating instructions for a computer to follow
- The goal should be well defined and measurable
- The executor is capable of doing exactly what is told
- The steps must be described in a language the executor understands
  - Clear syntax and semantics
- Each step can involve great detail
  - Break down tasks into more manageable steps
  - Define each step with basic operations
  - Ensure that instructions are clear and tailored to the executor's abilities

Computer Language

- All instructions must be expressed as 0s and 1s
- The presence of an electric voltage (5 V) represents a 1, while an absence represents a 0
  - Logic gates can be designed to manipulate logic states on a circuit board
- Ex. Python -> compiler -> object code -> linker -> binary executable program

Machine Language

- Binary representation used to communicate with computers
- MIPS is a RISC-style 32 bit processor
  - The first 6 bits determine what operation to perform

Assembly Language

- A symbolic representation of binary encoding
- Compiles into executable hex codes
- See MIPS Assembly

Labels

- Used to mark memory
- Typically used for jumping and memory references

Orders

- Instructions

- Ex. add, sub, addi

Mnemonics

- Used to represent instruction opcodes
- Human readable shorthand

Argument

- Value passed to a function or procedure

Directives

- Tells how to process the code

Comments

- Text following the mark `;` in a program line

Instruction Types

R Type

- 3 register operands
- Used for arithmetic and logical operations
- opcode (6), rs (5), rt (5), rd (5), shamt (5), funct (6)
- `<mnemonic> rd, rs, rt`

I Type

- 1 register operand
- 1 immediate value
- Often used for data transfer, branching, and arithmetic operations with immediate values
- opcode (6), rs (5), rt (5), immediate (16)
- `<mnemonic rt, rs, imm`

J Type

- 1 immediate value
- Typically used for unconditional jumps to specific addresses
- opcode (6), address (26)
- `<mnemonic> <address>`

Why Write in Assembly

- When speed / size is critically important

- Ex. automatic emergency braking systems
- A compiler can introduce uncertainty about time cost of operations
- Assembly language has tight control over which instructions execute
- In embedded applications, reducing a program's size so that it fits in fewer memory chips, reducing the cost of the embedded computer

High Level Languages

Human readable

Abstraction

Portability

Rich libraries and frameworks

Popular examples