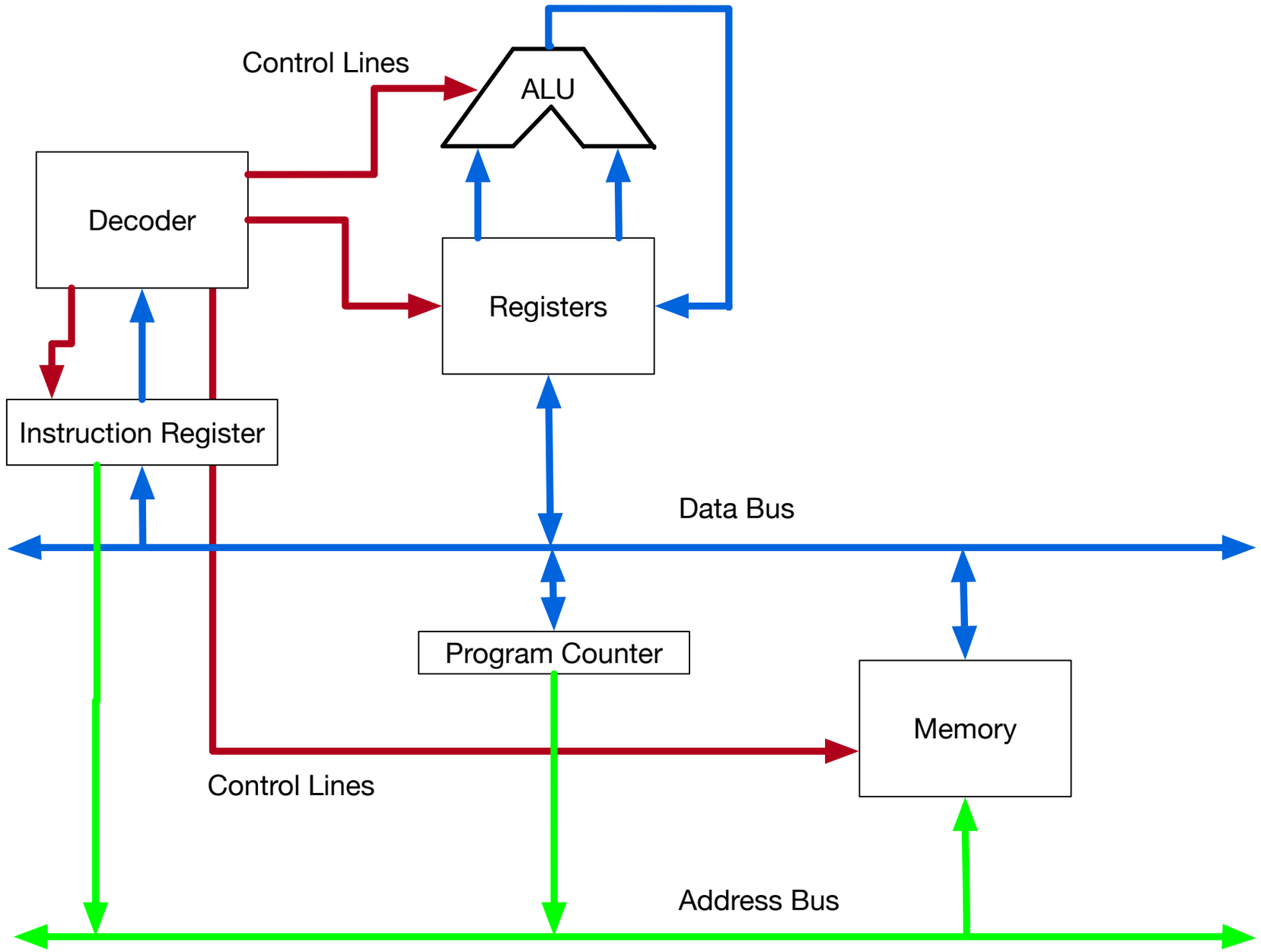
Software

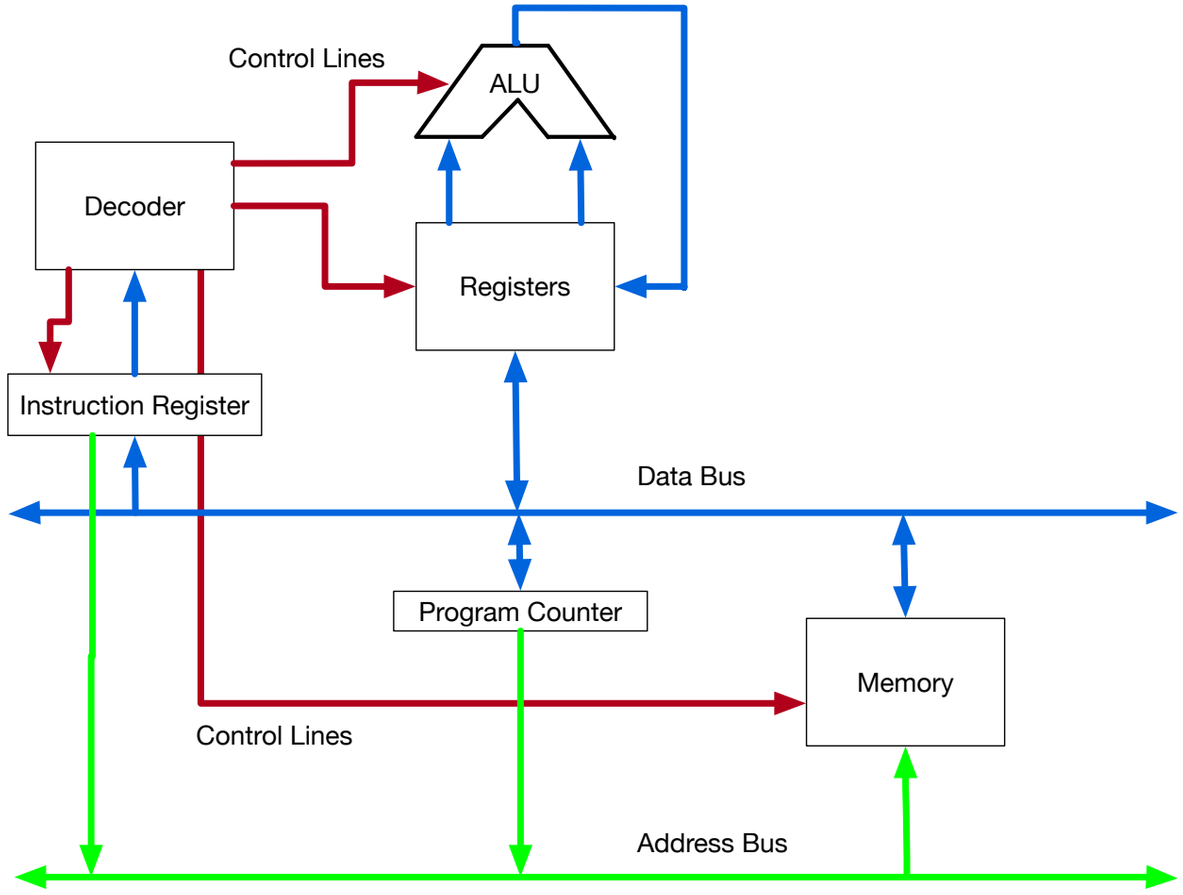
Thomas Schwarz, SJ

Working of Idealized CPU



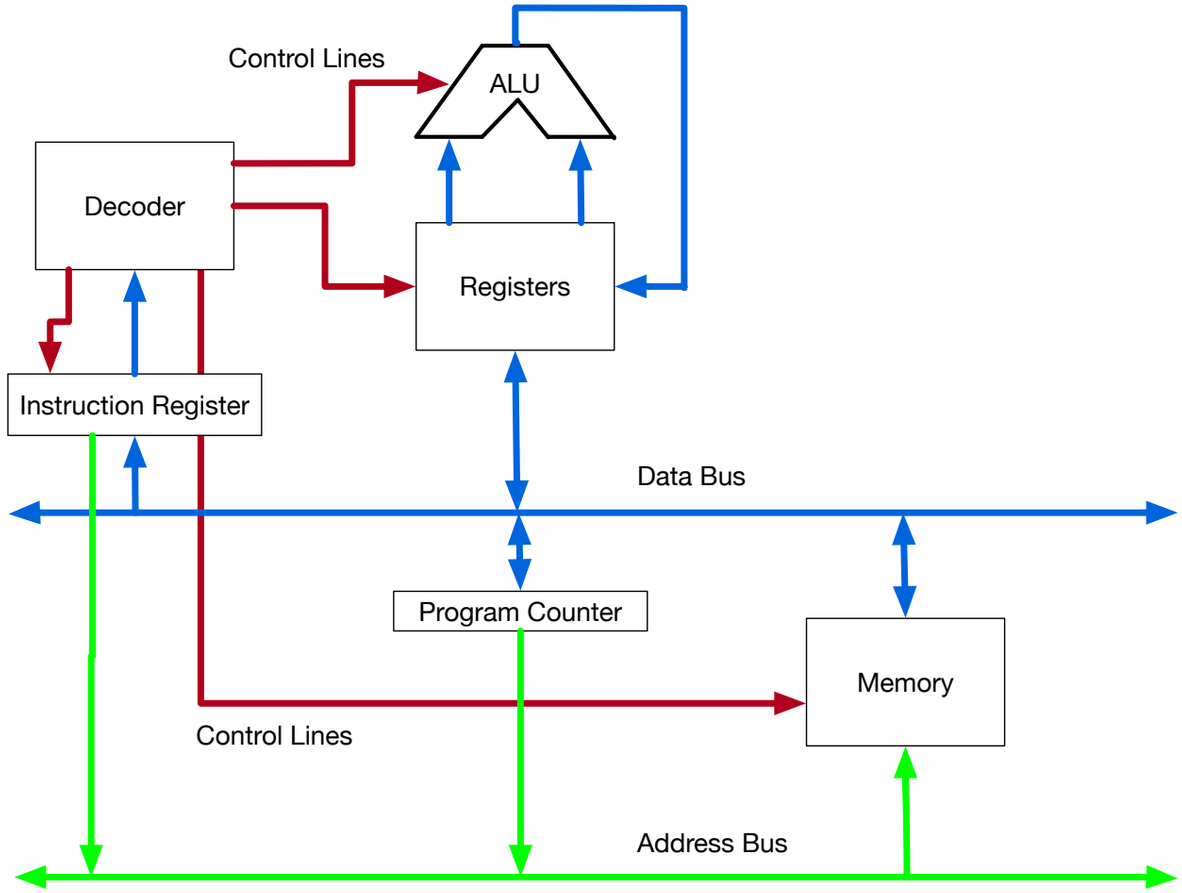
Working of Idealized CPU

- Instruction is loaded into instruction register



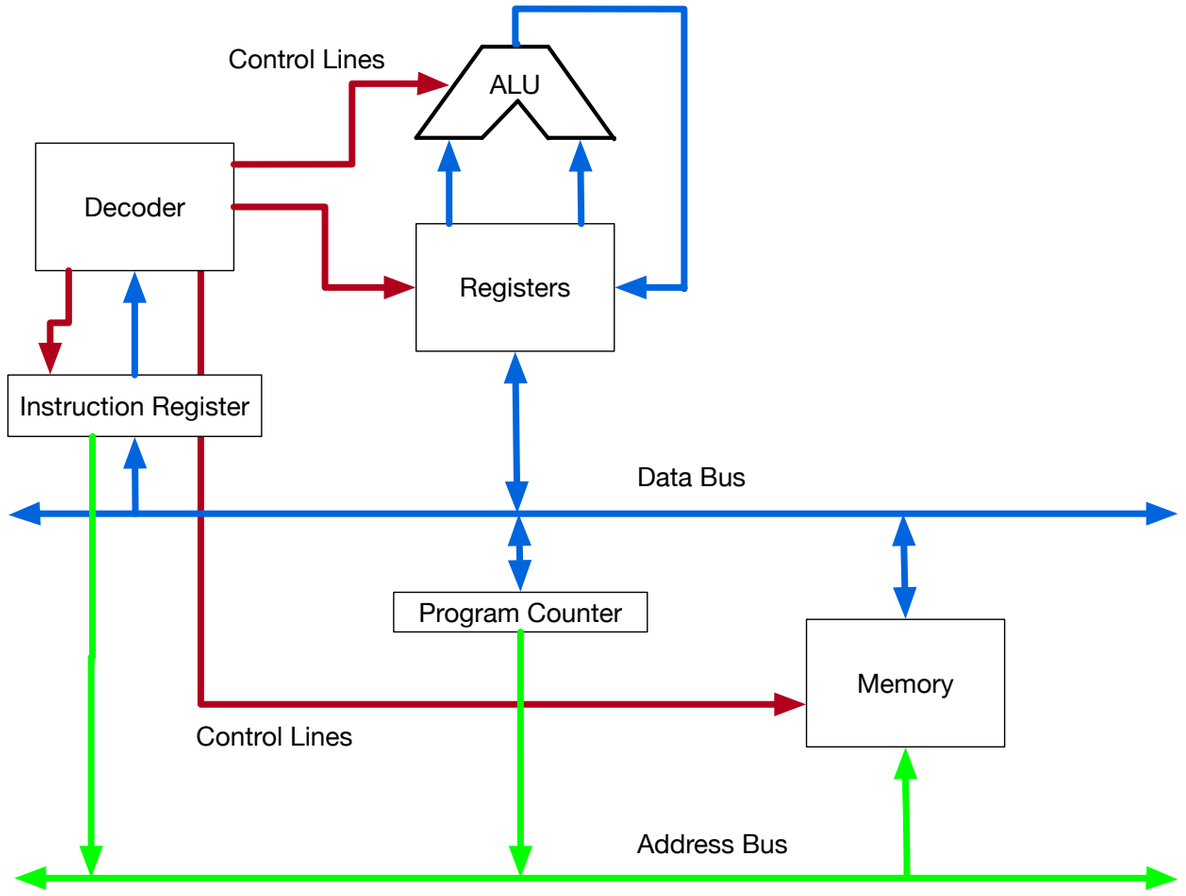
Working of Idealized CPU

- Decoder decodes instruction
- If arithmetic operation:
  - select operand and result registers
  - send opcode to ALU
- If memory access:
  - Select read or write
  - Address is calculated from instruction register
  - Destination / Origin in register



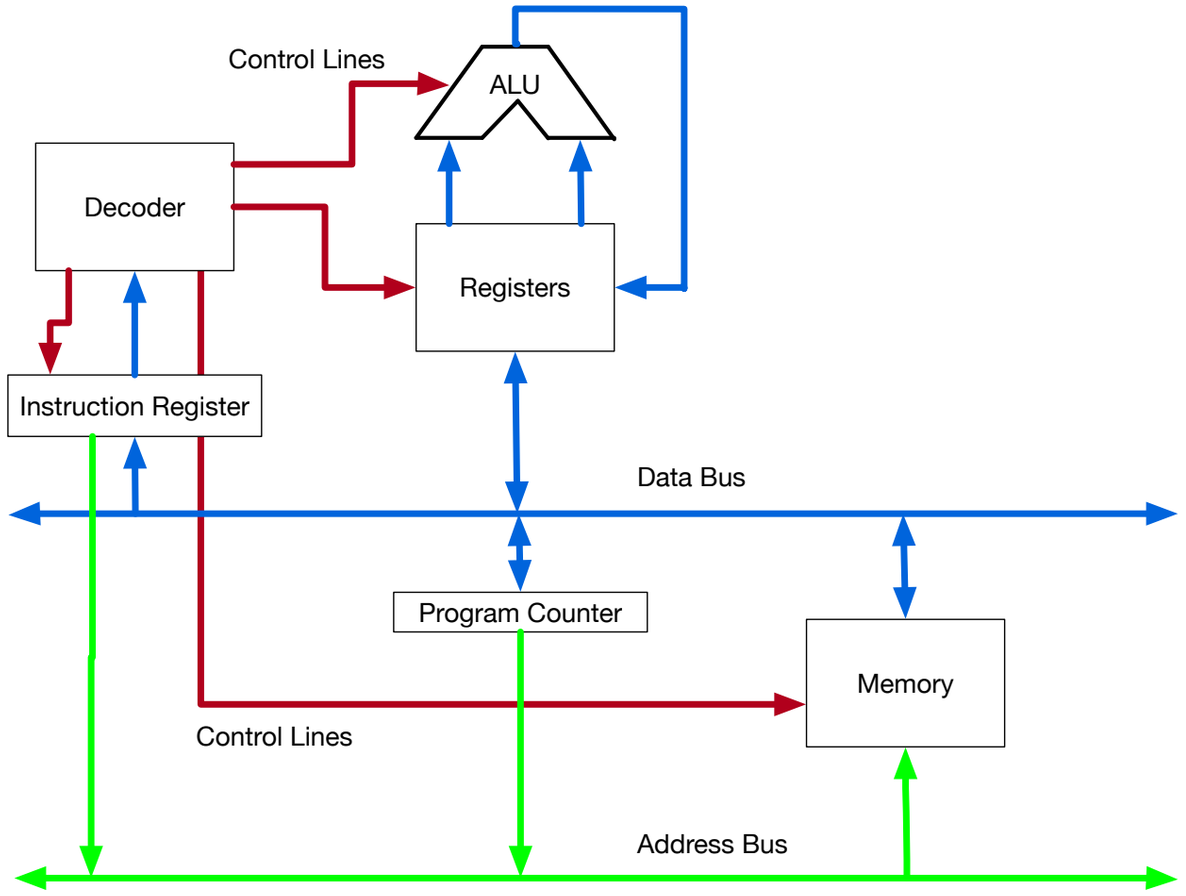
Working of Idealized CPU

- ALU executes instruction
- OR: Memory is read / written



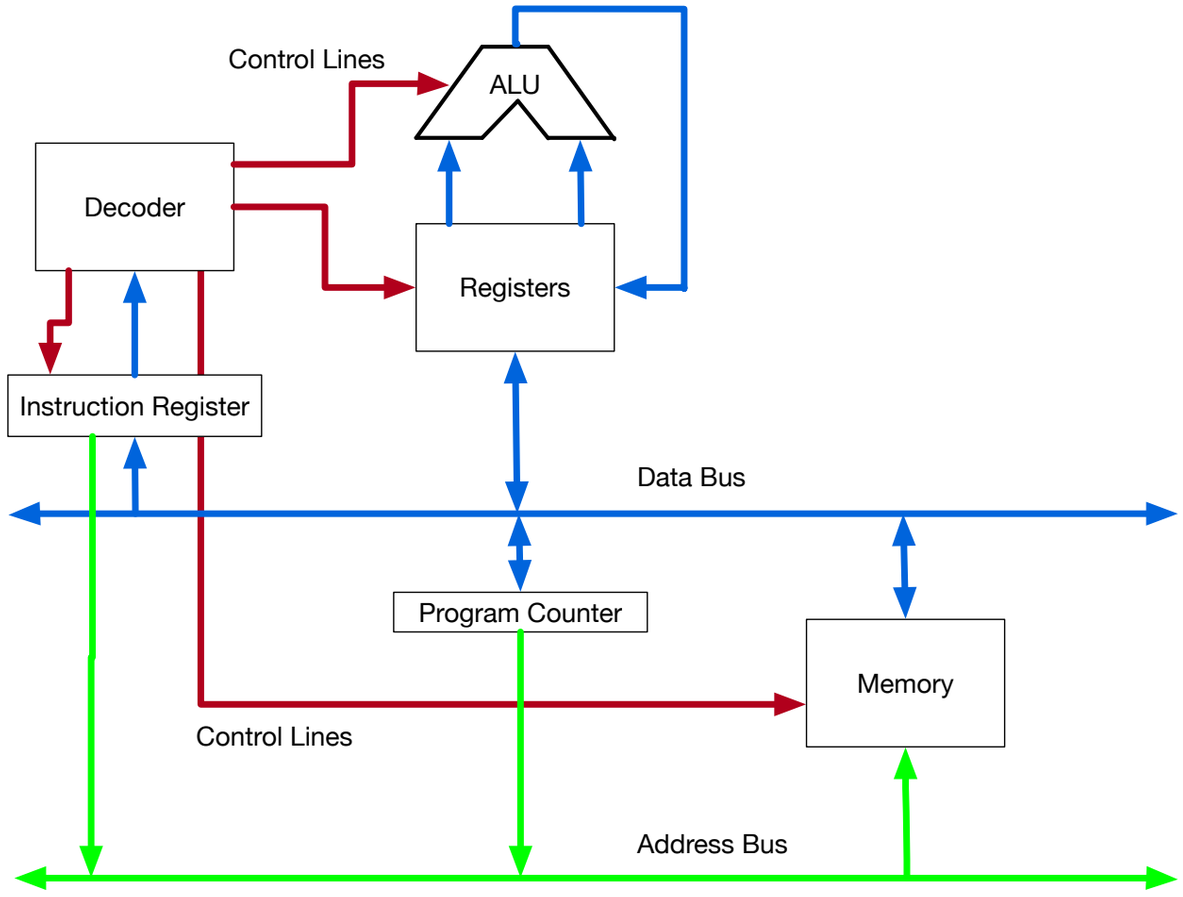
Working of Idealized CPU

- Program Counter is incremented or a new value is loaded from ALU result



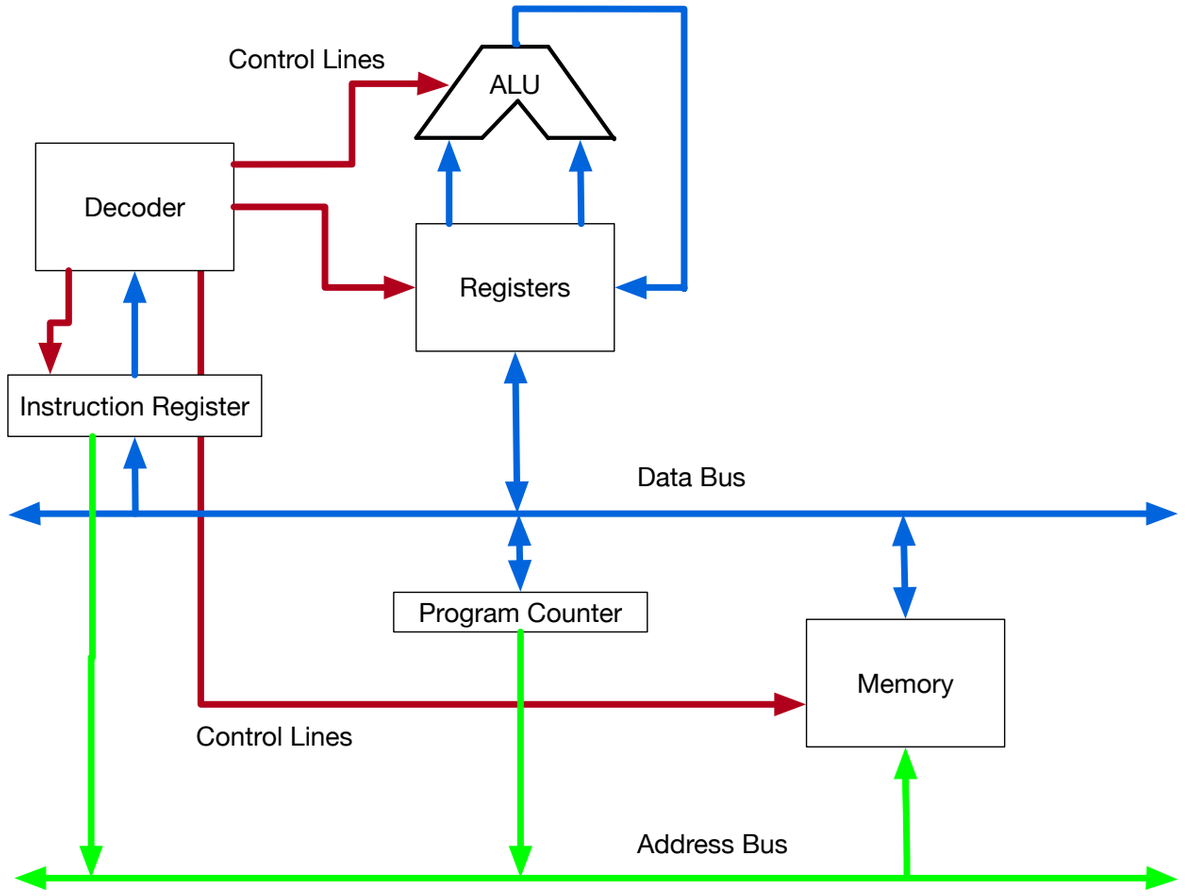
Working of Idealized CPU

- Fetch:
  - Load next instruction from Memory into Instruction Register
  - Address is in Program Counter



Working of Idealized CPU

- Cycle repeats ...



Machine Code

- **Instruction Set Architecture (ISA)**
  - Set of instructions
- **Complex Instruction Set Code (CISC)**
  - Arbitrary length instructions
  - Lots of instructions
- **Reduced Instruction Set Code (RISC)**
  - Smaller set of fixed length instructions

Machine Code

- Computer instructions
- Presented in binary or hexadecimal
- Architecture dependent

```
554889E5 48897DF8 488B45F8 488905A5 5800005D C3662E0F 1F840000 00000090
554889E5 4883EC10 48897DF8 488975F0 488B7DF8 488B75F0 E8131200 004883C4
105DC366 2E0F1F84 00000000 000F1F00 554889E5 4883EC10 48897DF8 8975F448
8B7DF88B 75F4E8D5 01000048 83C4105D C3662E0F 1F840000 0000000F 1F440000
554889E5 4883EC10 48897DF8 488B7DF8 E8DB0100 004883C4 105DC30F 1F440000
554889E5 4883EC10 48897DF8 488975F0 488B7DF8 488B75F0 E8E30100 004883C4
105DC366 2E0F1F84 00000000 000F1F00 554889E5 4883EC10 48897DF8 488B7DF8
E8FB0100 004883C4 105DC30F 1F440000 554889E5 4883EC10 48897DF8 488B7DF8
E85B0300 004883C4 105DC30F 1F440000 554889E5 488D3DC5 57000048 8D35DEFE
FFFFE819 00000048 8B3D8A57 0000E87D 0000005D C3662E0F 1F840000 00000090
```

Assembly Language

- Can be written by human hands
- But this is very error-prone
- Use Assembly code:
  - Code in simple commands uses mnemonics

```
INC COUNT          ; Increment the memory variable COUNT
MOV TOTAL, 48      ; Transfer the value 48 in the
                   ; memory variable TOTAL
ADD AH, BH         ; Add the content of the
                   ; BH register into the AH register
AND MASK1, 128     ; Perform AND operation on the
                   ; variable MASK1 and 128
ADD MARKS, 10      ; Add 10 to the variable MARKS
MOV AL, 10         ; Transfer the value 10 to the AL register
```

Assembly Language

- To translate from Assembly to Machine Code:
  - Use a program: the assembler
- Assembly language is a bit more portable than machine code
  - But still dependent on the architecture

Higher Level Language

- Programming in Assembly is still difficult and error prone
  - Search for "automatic programming"
    - E.g. MIDAC Input Translation Program:
      - Makes it easier by taking care of memory addressing (which was complex)
    - But automatic programming creates code that runs 5 to 10 times slower
  - Costs for programmers equals costs of computing system

Higher Level Language

- Early idea:
  - create a virtual computer
  - write programs for this virtual computer
  - have the programs translated to assembly for a real computer

Higher Level Language

- Backus, Ziller, Nutt, et al. at IBM, 1958: "Programming Research Group"
- Invent first successful high level programming language
- (There were earlier examples, e.g. Zuse's *Plankalkül*, that were unknown to the group)
- Ideas:
    - Variables, Functions, Expressions, Subscripted variables, Arithmetic formulas, DO formulas: early version of loops

Higher Level Language

- Results in **For**mula **Trans**lator
  - Still around and controlled by ANSI
  - Key success is building a *Compiler*
    - Takes code in Fortran and translates it to Assembly

Higher Level Language

- Modules and Libraries
  - Libraries consist of modules that are useful in many different settings
  - A big program uses libraries extensively
  - Modules refer functions and constants in other modules
  - After modules are compiled:
    - Use a **Linker** to translate these references into code

Higher Level Language

- After Fortran:
  - **Algo**ritmic **L**anguage 58 (Zürich)
  - **Co**mmon **B**usiness **O**riented **L**anguage: Cobol
    - Created based on Flowmatic and Comtran (IBM)
- And many, many more

Scripting Languages

- Java: Originally designed for interactive television (Gosling at SUN)
- Recognized for its capacity to run on any platform
- Uses a **J**ava **V**irtual **M**achine (JVM)
  - A software program that simulates a computer
  - Java program is translated into Java Byte Code
  - Java Byte Code is simulated by the JVM

Scripting Languages

- Portability:
  - To make a platform capable of running Java
    - Build a Java Virtual Machine
    - Any JVM can execute the same byte code
- Security:
  - Idea: Limit what a JVM can do: a ***sandbox***
  - Then we can run Java Byte Code on our machine without worries
  - Even if we downloaded it from websites

Scripting Languages

- Java Security:
  - Turns out that a useful JVM needs privileges that destroy this sandbox
  - Therefore: Your browser will no refuse to download and execute Java Byte Code.

Scripting Languages

- Python
  - Uses Python Byte Code to execute on a Python Virtual Machine
  - Makes it portable