PROGRAMMING IN PYTHON I

Basics of Programming (Optional)

Andreas Schörgenhumer

Institute for Machine Learning

JOHANNES KEPLER UNIVERSITY LINZ

Institute for Machine Learning

Copyright Statement

Machine Code

- Instructions to the machine (e.g., the **controller**) are encoded in bits
  - Machine Code is often visualized as a sequence of **hexadecimal numbers**

bit pattern	decimal	hexadecimal	command
0000	0	0	MOVE
0001	1	1	ADD
0010	2	2	MULTIPLY
0011	3	3	...
0100	4	4	...
0101	5	5	...
0110	6	6	...
0111	7	7	...
1000	8	8	...
1001	9	9	...
1010	10	A	...
1011	11	B	...
1100	12	C	...
1101	13	D	...
1110	14	E	...
1111	15	F	...

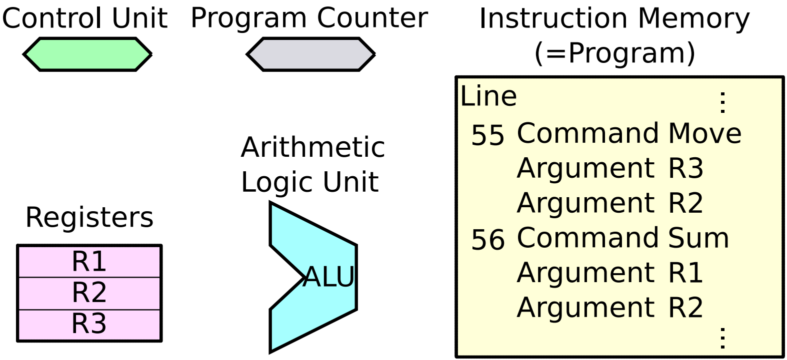
SIMPLIFIED EXECUTION OF A PROGRAM

Setup

- This section will give you a rough idea about how a program is executed
- There are several hardware parts in a controller:

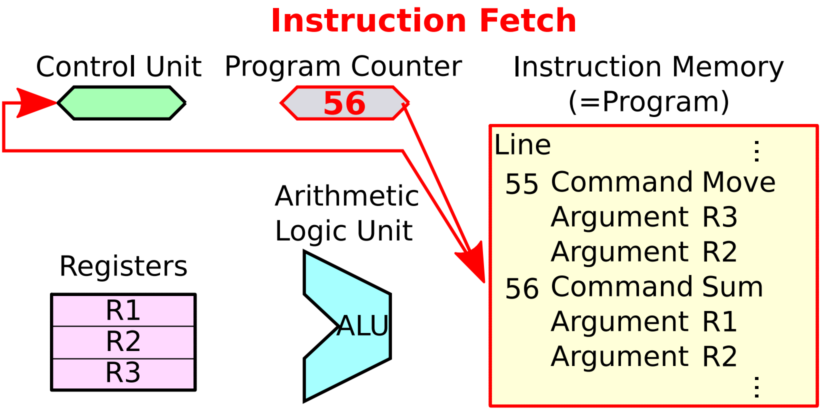
1. **Instruction Memory (IM):** The program in machine code (e.g. stamp-card or flash-storage)
2. **Program Counter (PC):** Holds a value that represents the line in the code (starts at 0)
3. **Registers (R):** (Temporary) storage to store bit patterns
4. **Arithmetic Logic Unit (ALU):** Circuit that performs arithmetic operations
   - These operations often use the same specific registers as input/output (=wired to the registers)
5. **Control Unit (CU):** Circuit that activates registers, ALU, and Program Counter based on current bit pattern in code

Simplified Execution of a Program (1)



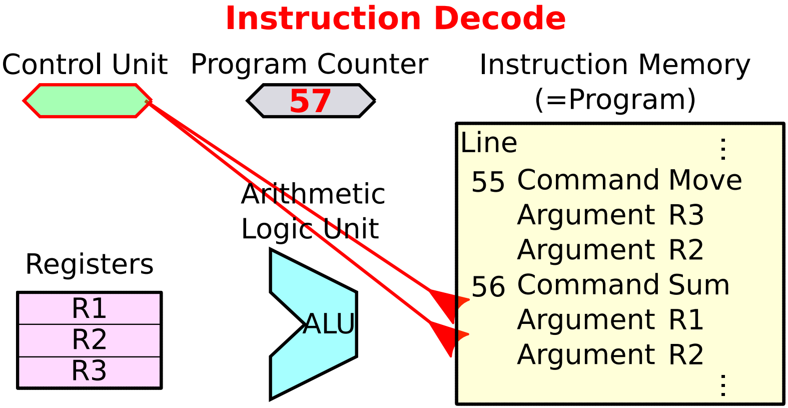
This is our processor and on the right side we see our program (**IM**)

Simplified Execution of a Program (2)



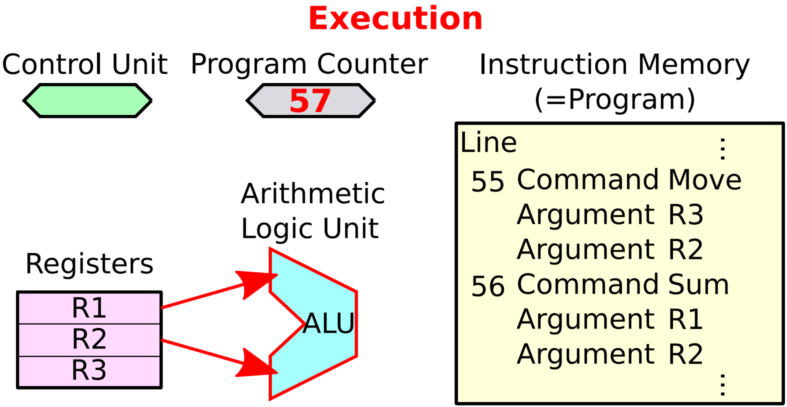
CU fetches bit pattern from **IM** at line number stored in **PC** (e.g. 56). **CU** increases **PC** by one (e.g. to 57).

Simplified Execution of a Program (3)



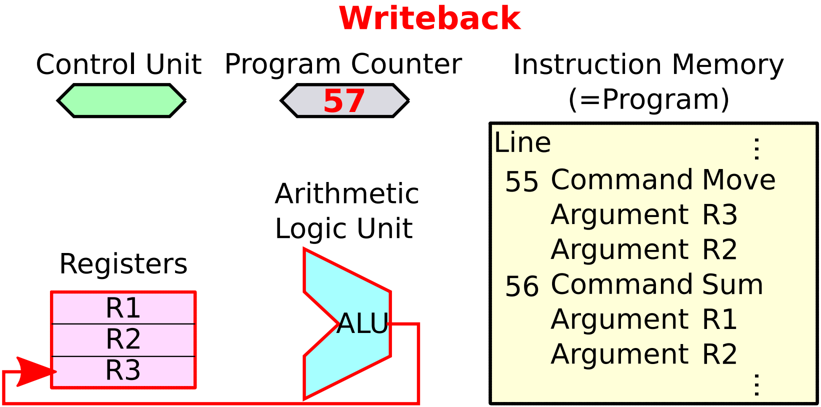
Bit pattern `Command Sum` triggers **CU** to fetch next 2 bit patterns in code (adress R1 and R2) and set **ALU** to summation.

Simplified Execution of a Program (4)



ALU performs arithmetic summation with values (bit patterns) from inputs R1 and R2.

Simplified Execution of a Program (5)



Result value (bit pattern) from **ALU** is stored in R3. In next step, **CU** will fetch the next bit pattern.

Simplified Execution of a Program (6)

- As you can see, summing up two values effectively can require more than one line of Machine Code
  1. Move values to registers
  2. Perform summation
  3. Move result from register to somewhere else
- This can get tedious and complex/difficult to read and is highly dependent on hardware

$\rightarrow$ We would often like to get rid of (**abstract from**) these details/hardware

ABSTRACTION AND LANGUAGES

Abstraction and Languages (1)

Machine code	Instructions as bits
Assembly	Abstracts from Machine Code: More readable than bits; still close to hardware; potentially very fast (full control, special hardware functionalities accessible)
C, etc.	Abstracts from Assembly: Readable code; abstracts registers and instructions; fast because whole program is compiled and optimized at once (possibly for specific CPU architecture)

Abstraction and Languages (2)

C#, Java	Abstracts further: Readable, convenient code but often no idea about actual instructions happening; medium/fast and compiled at once (to architecture independent intermediary format)
Python, R	Interpreted languages:[1] Lines of code are executed one-by-one (i.e., *interpreted*) $\rightarrow$ generally, no compilation of whole program but only individual lines; slow if not using specialized packages

[1] Strictly speaking, it is actually the *implementation* of the language that holds the interpreted/compiled property. The default Python implementation, CPython, first compiles the code into bytecode which is then interpreted by a virtual machine. So contrary to the official documentation, Python (or rather its implementation) is actually compiled. See this discussion for more details.

HARDWARE

Hardware (1)

- **CPU**: Central processing unit (the actual main processor)
- **RAM**: Random-access memory (the working memory as volatile[1] storage)
- **GPU**: Graphics processing unit (computer graphics, image processing, deep learning, etc.)
- **SSD**: Solid-state drive (non-volatile[1] storage via integrated circuit)
- **HDD**: Hard disk drive (non-volatile[1] storage via rotating disks)

[1] Volatile memory needs constant power in order to retain data.

Hardware (2)

- To use a computer efficiently, you have to think about which parts to use for which task
- **CPU** (general computations) vs. **GPU** (dedicated to, e.g., matrix operations)
- **RAM** (small, fast) vs. **SSD/HDD** (large, slow)

Hardware – Approximate Operation Times[1]

System Event	Actual Latency	Scaled Latency
1 CPU cycle	0.4ns	1min
Level 1 cache	0.9ns	2.25min
DDR RAM	100ns	4.17h
SSD I/O	50–150µs	86–260d
Rotational disk	1–10ms	4.76–47.56yr

[1] Intel Xeon processor E5 v4, 2.4GHz, latency times