Developer productivity tools

—

Python

STSCI 4780 Lab

Tom Loredo

Developer productivity tools

- Console/terminal apps and shells
- Code editors and IDEs
- Documentation browsers
- Code snippet managers & programmer notebooks

Terminals and shells

- Programming = using language to tell computer what to do
- Shell = Most basic and interactive way to direct your computer with language
  - Language is more expressive than any GUI could be
  - Modern shells enable construction of processing chains via redirection
- Console/terminal/command-line = Software for interactive textual interaction with computer
- Shell = underlying program that interprets typed commands, acts, and responds
- Popular shells:
  - Linux, Mac: Bash, zsh (in Terminal or iTerm terminals)
  - Win: CMD, git-bash (in Console2 terminal); PowerShell; Anaconda Prompt

Code editors & IDEs

E.g.: Spyder, Sublime Text, Atom, Visual Code, Eclipse

Language-aware features

- Syntax highlighting
- Automatic indentation
- Refactoring tools
- Code-completion
- IDEs: Shell/interpreter interaction, browsers, debuggers...

Benefits

- Better code comprehension
- Less typing:
  - Faster coding
  - More readable code -> fewer bugs
- Faster development cycle...

Documentation browsers

- Mac: Dash for Mac - Documentation Browser, Snippet Manager - Kapeli
- Linux, Win: Zeal - offline API documentation browser
- Win: Velocity - The Documentation and Docset Viewer for Windows
- Ubuntu Linux: Devhelp — Ubuntu Apps Directory
- Web browser (incl. offline): DevDocs (also Chrome & Firefox extensions)

Web browser-based DevDocs: http://devdocs.io/

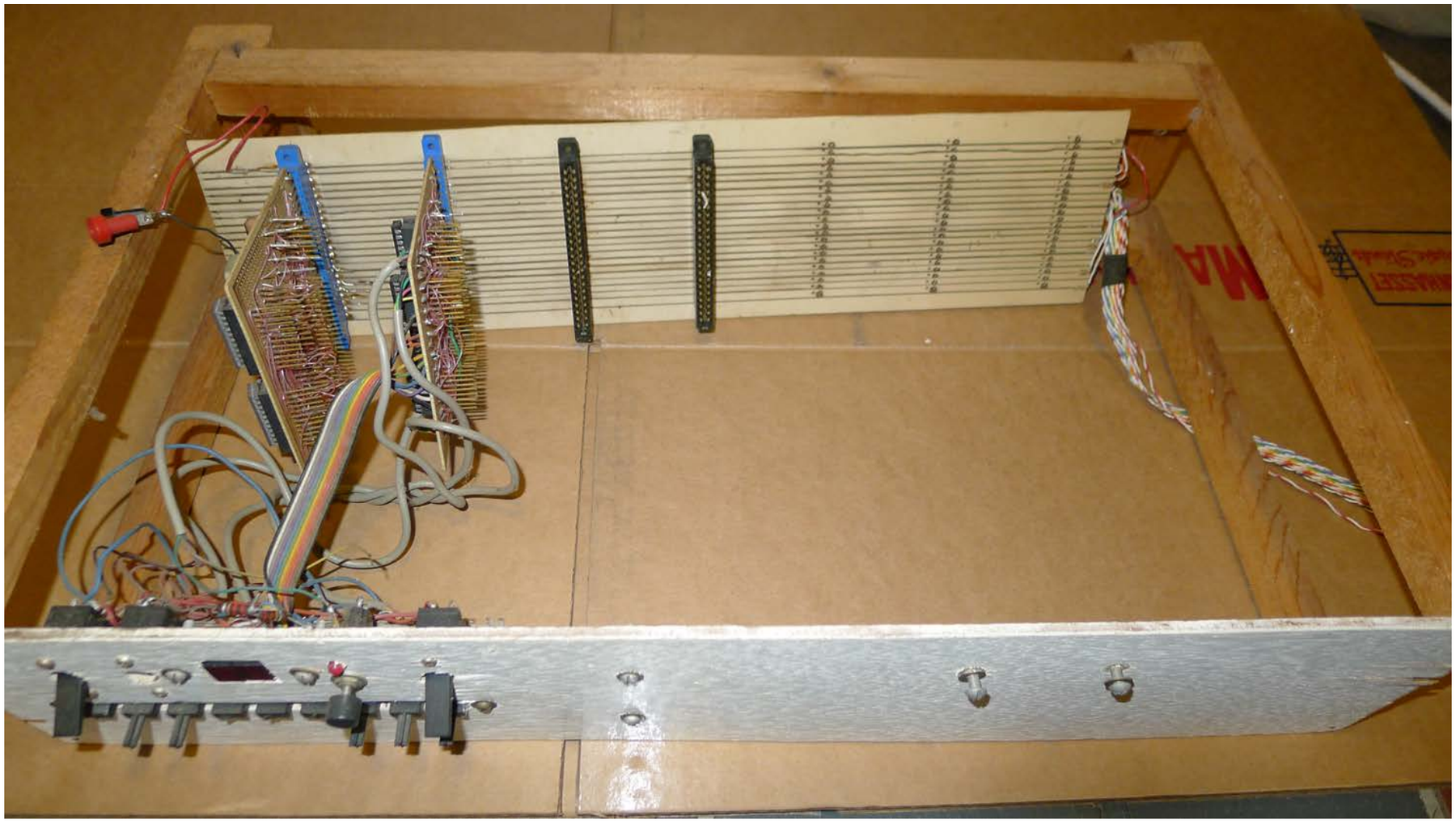
Code snippet managers

- Code editors often have snippet library capability (often as an add-on package/plug-in)
- Mac:
  - Dash for Mac - Documentation Browser, Snippet Manager (somewhat awkward interface)
  - SnippetsLab
  - CodeBox (discontinued?)
- Win/Linux/Mac:
  - QSnipps - Code Snippet Manager Tool
  - Gisto - Manage gists (GitHub-hosted snippets) from desktop
  - Look for lists: 5 Code Snippet Managers, Best Code Snippet Managers, Top 10 Open Source snippets managers

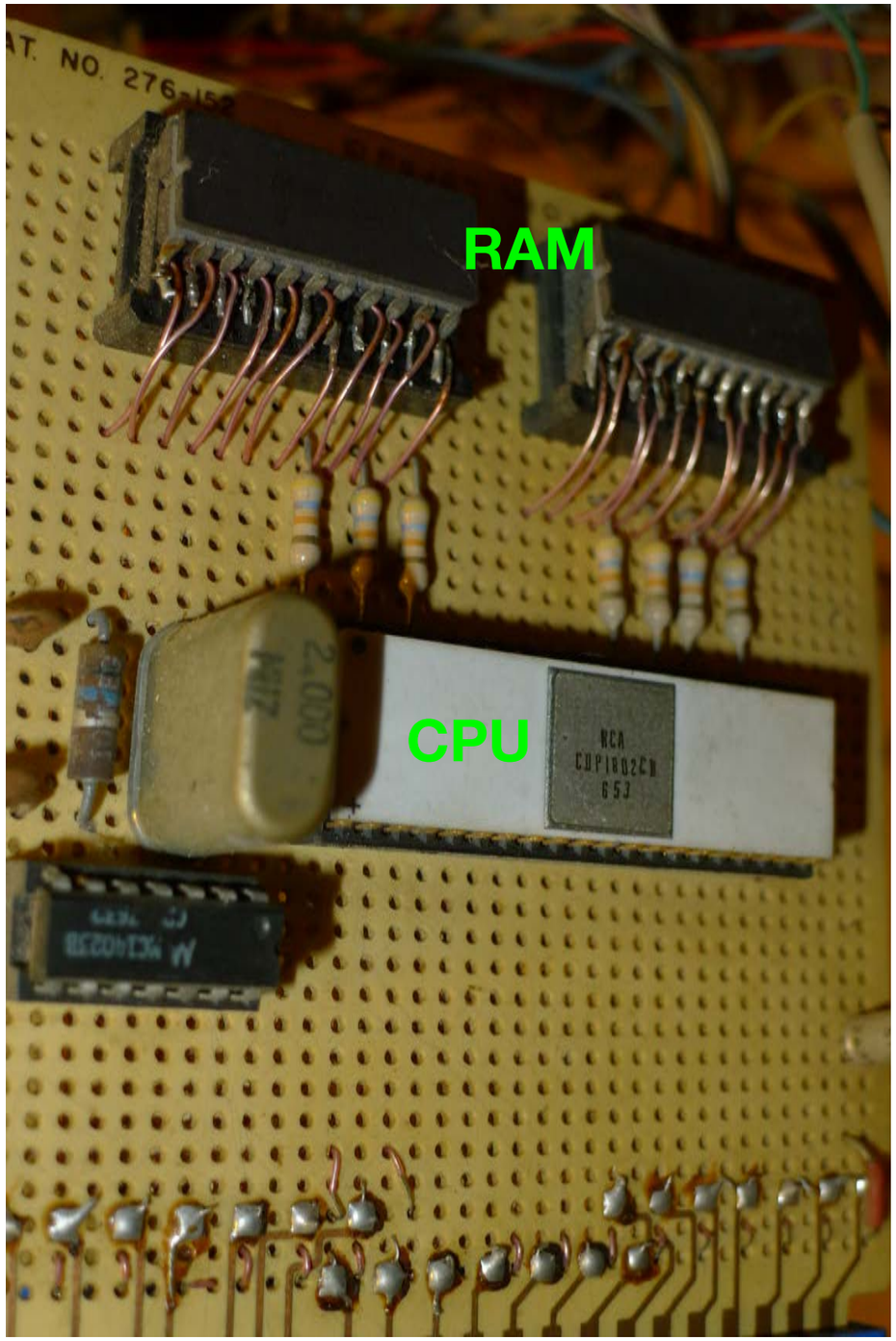
Programmer notebooks

- Combines support of styled text (e.g., via Markdown) with basic code editing, syntax highlighting
- Some resemble Jupyter notebooks with cells of content
- Mac: Quiver (unsupported?)
- Win/Linux/Mac: Joplin, Boostnote, MedleyText...
- 7 Best Note-Taking Apps for Programmers...
- Use a regular cloud-synced note-taking app: Evernote, OneNote, Apple Notes, Simplenote....

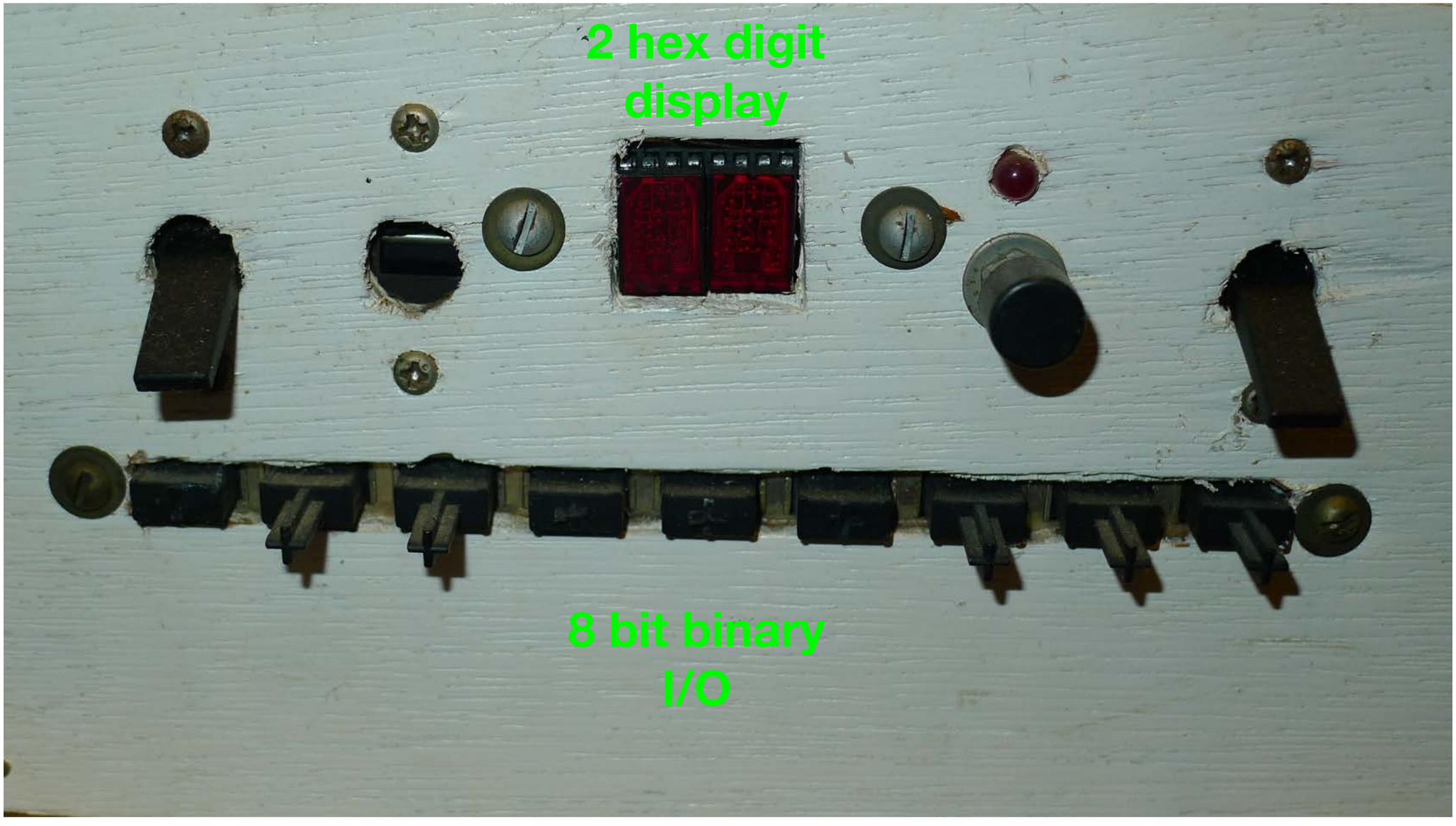
Programming languages



Tom's COSMAC Elf, built ca. 1976



Wire-wrap and solder construction



Front panel

Machine & assembly code

Low-level programming languages

A Short Course In Programming
by Tom Pittman (1980)

Address

```
                ..  PROGRAM 2.1 -- MEMORY CLEAR
                ..
0000 90 CLEAR:  GHI 0    .. REGISTER 0 HAS 0001
0001 AE         PLO 14   .. MAKE RE=0000
0002 BE         PHI 14
0003 EE LOOP:   SEX 14   .. EACH TIME, R14 IS -1
0004 73         STXD     .. D STILL HAS 00
0005 30         BR LOOP  .. GO BACK FOR ANOTHER
0006 03
```

Assembly code

Machine code

```
                ..  PROGRAM 2.2 -- MEMORY SEQUENCER
                ..
0000 90 SEQ:    GHI 0    .. THIS PART IS
0001 AE         PLO 14   ..   JUST LIKE CLEAR
0002 BE         PHI 14
0003 EE LOOP:   SEX 14
0004 8E         GLO 14   .. THIS IS ADDRESS VALUE
0005 73         STXD     ..   SO DATA=ADDRESS
0006 30         BR LOOP  .. REPEAT UNTIL DONE
0007 03


                ..  PROGRAM 2.3 -- SLOW BLINK
                ..
0000 91 BLINK:  GHI 1    .. LOOK AT TIMER IN R1
0001 CE         LSZ      .. IS ZERO ONLY 1/256
0002 7A         REQ      .. IF NOT 00, Q OFF
0003 38         SKP
0004 7B         SEQ      .. WHILE ZERO, Q ON
0005 11         INC 1    .. BUMP COUNTER
0006 30         BR BLINK..   THEN REPEAT
0007 00
```

From http://www.cosmacelf.com

Human assembler

APPLE COMPUTER CO. 4-6-76 S. Wozni

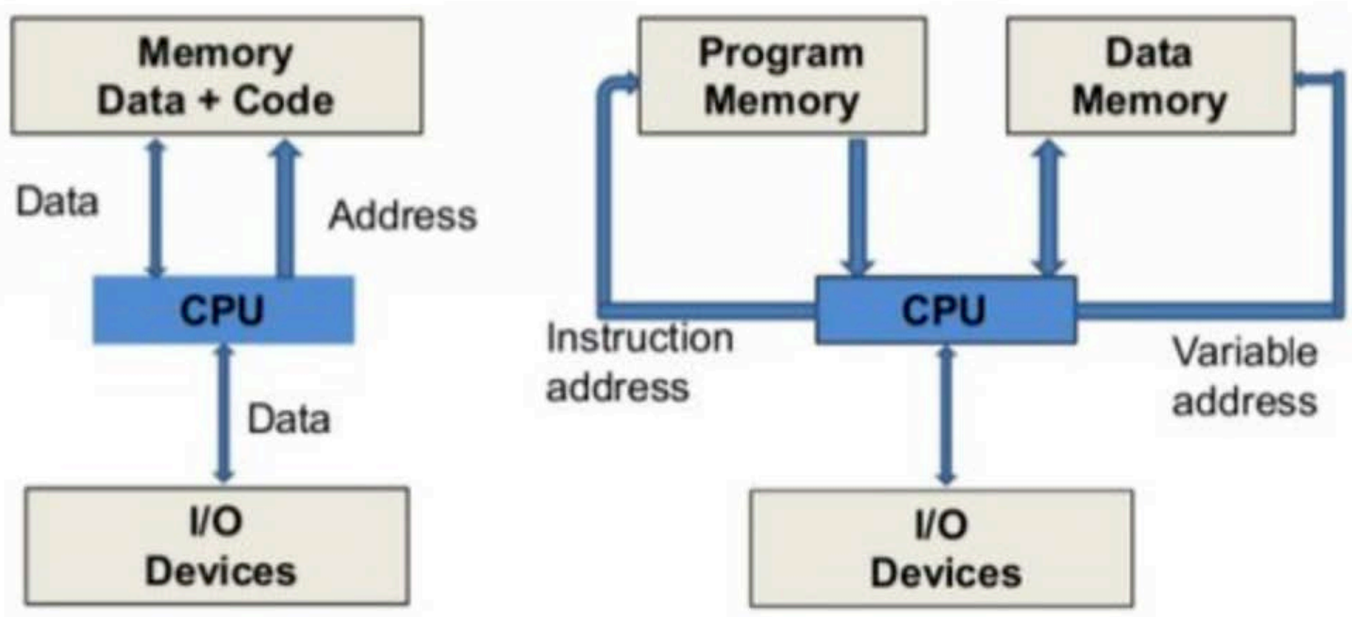
```
300  18        ADD    CLC              Clear carry.
301  A2 02             LDX  #$02        Index for 3-byte add.
303  B5 09     ADD1   LDA(z)M1,X(09)
305  75 05            ADC(z)M2,X(05)   Add a byte of Mant2 to Mant1.
307  95 09            STA(z)M1,X(09)
309  CA               DEX              Advance index to next more signif. byte.
30A  10 F7            BPL  ADD1(-09)   Loop until done.
30C  60               RTS              Return.

30D  06 03        MD1    ASL(z)SIGN(03)   Clear LSB of SIGN..
30F  20 12 03            JSR  ABSWAP(312) Abs Val of Mant1, then swap with Mant2.
312  24 09        ABSWAP BIT(z)M1(09)     Mant1 neg?
314  10 05               BPL  ABSWAP1(+05)   No, swap with Mant2 and return.
316  20 84 03            JSR  FCOMPL(384)    Yes, complement it.
319  E6 03               INC(z)SIGN(03)   Incr. SIGN, complementing LSB.
31B  38        ABSWAP1 SEC              Set carry for return to MUL/DIV.
31C  A2 04     SWAP   LDX  #$04         Index for 4-byte swap.
31E  94 0B     SWAP1  STY(z)E-1,X(0B)
320  B5 07               LDA(z)X1-1,X(07)   Swap a byte of Exp/Mant1 with
322  B4 03               LDY(z)X2-1,X(03)     Exp/Mant2 and leave a copy of
324  94 07               STY(z)X1-1,X(07)     Mant1 in E (3 bytes). E+3 used.
326  95 03               STA(z)X2-1,X(03)
328  CA                  DEX              Advance index to next byte.
329  D0 F3               BNE  SWAP1(-0D)  Loop until done.
32B  60                  RTS              Return.

32C  C6 08     NORM1  DEC(z)X1(08)     Decrement Exp1.
32E  06 0B             ASL(z)M1+2(0B)
330  26 0A             ROL(z)M1+1(0A)   Shift Mant1 (3 bytes) left.
332  26 09             ROL(z)M1(09)
334  A5 09     NORM   LDA(z)M1(09)     High-order Mant1 byte.
336  C9 C0             CMP  #$C0        Upper two bits unequal?
338  30 04             BMI  RTS1(+04)     Yes, return with Mant1 normalized.
33A  A5 08             LDA(z)X1(08)     Exp1 zero?
33C  D0 EE             BNE  NORM1(-12hex)   No, continue normalizing.
33E  60        RTS1   RTS              Return.
```

Memory: Instructions & data

- Computer memory holds both instruction codes ("code") and data
- Two main hardware architectures: von Neumann, Harvard



Von Neumann Machine

Harvard Machine

High-level languages

In file add.c:

```
int add(int i,int j)
{
    int p = i + j;
    return p;
}
```

Compiler

In file add.s:

```
add:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp      //create space for integer p
    movl 8(%ebp),%edx  //8(%ebp) refers to i
    addl 12(%ebp), %edx    //12(%ebp) refers to j
    movl %edx, -4(%ebp)    //-4(%ebp) refers to p
    movl -4(%ebp), %eax    //store return value in eax
    leave          //i.e. to movl %ebp, %esp; popl %ebp ret
```

From http://repo.hackerzvoice.net/depot_madchat/coding/ramankutty.html

Also ***interpreted*** languages (translate on-the-fly)

Very high-level languages

Python

```python
import numpy
nbr_values = 8192
n_iter = 100000

a = numpy.ones(nbr_values).astype(numpy.float32)
for i in range(n_iter):
    a = numpy.sin(a)
```

C

From StackOverflow

```c
#include <math.h>
#include <stdlib.h>

float *sin_array(const float *input, size_t elements)
{
    int i = 0;
    float *output = malloc(sizeof(float) * elements);
    for (i = 0; i < elements; ++i) {
        output[i] = sin(input[i]);
    }
    return output;
}
```

```c
#include <math.h>
#include <stdlib.h>

extern float *sin_array(const float *input, size_t elements)

int main(void)
{
    int i;
    int nbr_values = 8192;
    int n_iter = 100000;
    float *x = malloc(sizeof(float) * nbr_values);
    for (i = 0; i < nbr_values; ++i) {
        x[i] = 1;
    }
    for (i=0; i<n_iter; i++) {
        float *newary = sin_array(x, nbr_values);
        free(x);
        x = newary;
    }
    return 0;
}
```

- Higher level of abstraction
- Often compiled to a "virtual machine" with VMs implemented for various hardware
- VHLLs are sometimes domain-specific (Stan!)

Python ecosystem

The Python language(s)

- Python 2 (2.7.14)
- Python 3 (3.6.4)—Mostly minor language changes, major ABI changes
- Standard Libraries

Python implementations

- ***CPython*** — Reference implementation
- Jython — In Java
- IronPython — In C#
- *Performance-tuned:* PyPy & RPython, Stackless Python (fast, lightweight threads)
- *Microcontroller:* pyboard
- Others...

"PyData stack"

- Fast array processing (in/out of core)
  - ***NumPy*** (successor to Numeric, Numarray)
  - Blaze (possible successor to NumPy)
  - **Pandas**
- Scientific computing libraries
  - ***SciPy***
  - **PyMC**
  - Scikit-Learn
- Plotting
  - ***matplotlib***
  - Seaborn...
  - Bokeh (browser-based)
- Accelerators
  - Numba
  - Cython
- Symbolic computation: SymPy (see also: Sage)
- Enhanced interactive shell and notebook: ***IPython***

The Python language

- **Origins**
  - Guido van Rossum (Google, DropBox), Monty Python fan, BDFL
  - Productivity + Pedagogy: ABC...
- **Multi-paradigm**
  - Paradigm = How code is structured & executed
  - Imperative/procedural, object-oriented, functional
- **Object-oriented implementation**
- **Modes of use**
  - Interactive
  - Scripts, programs

Extending Python—Code re-use

- ***Functions*** and ***classes*** in a script
- ***Module*** - Functions and classes in a single file meant for **import**
  - **plain** (in Python) and **extension** (in C, C+ +, Fortran)
- ***Package*** — collection of modules in a folder (with special structure)

Built-in data types

- ***Numbers:*** integers, floats
- ***Sequences:***
  - strings, tuples, lists
    - 0-based indexing
    - slice notation
    - NumPy arrays have similar interface
- ***Mappings:*** dictionaries ("dicts")

Notable features

- Variables are ***labels***, not containers
- ***Mutable*** and ***immutable*** data types
- ***Functions*** with **keyword args**, default values
- ***Whitespace*** for organizing blocks of code (no braces!)
- Object ***classes*** and ***instances***
  - *State:* per-instance memory for **data** based on class template
  - *Behavior:* single copy of **methods** code in class
  - **"dot notation"** for data and method attributes