PILLAR PLUGGEDIN PRESENTS

How Computers Really Work

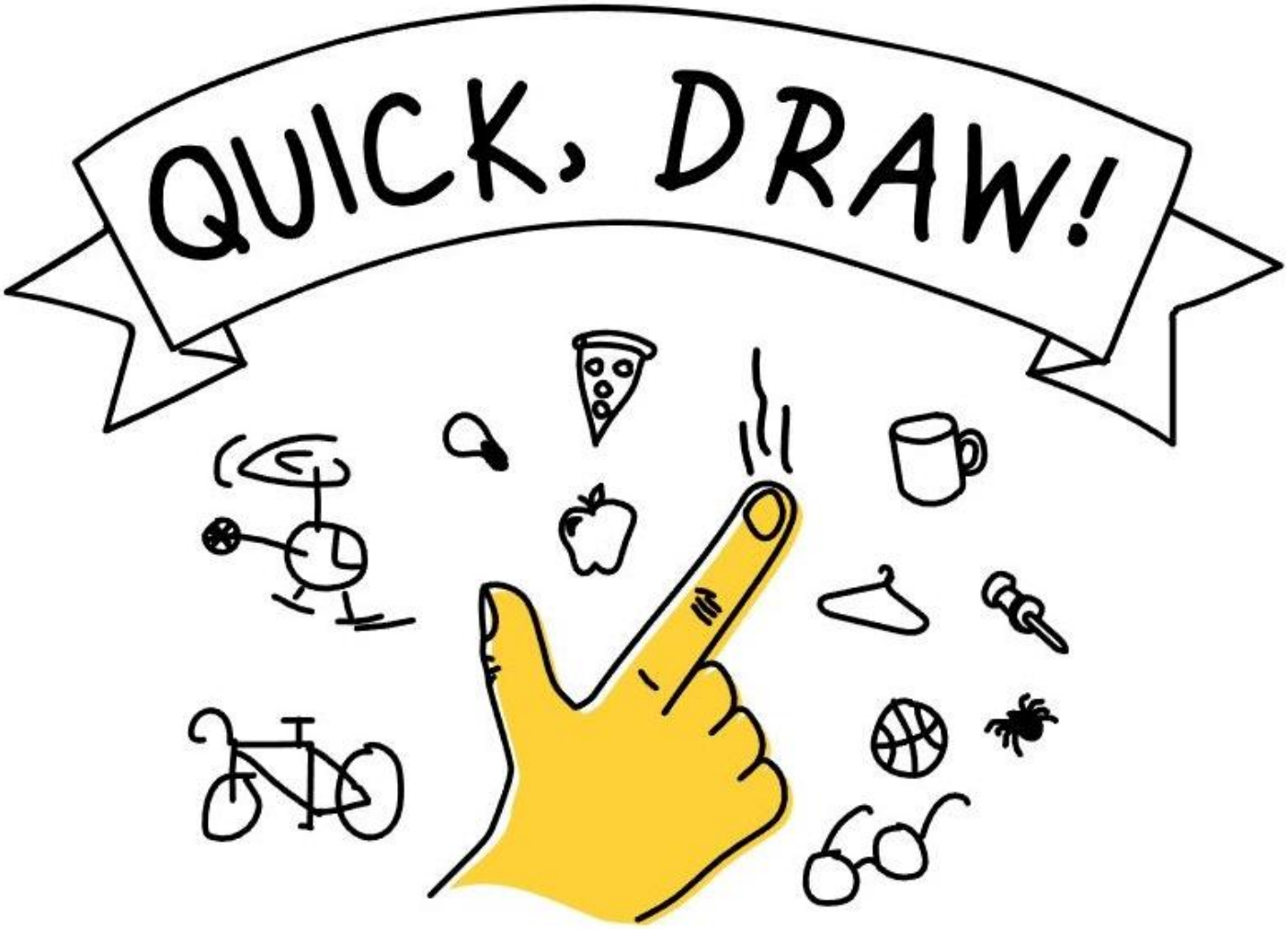
Revealing the magic behind the machine.

pillar

Computer

/kəmˈpyo͞odər/

noun - an electronic device for storing and processing data, typically in binary form, according to instructions given to it in a program


QUICK, DRAW!

Computer

/kəmˈpyo͞odər/

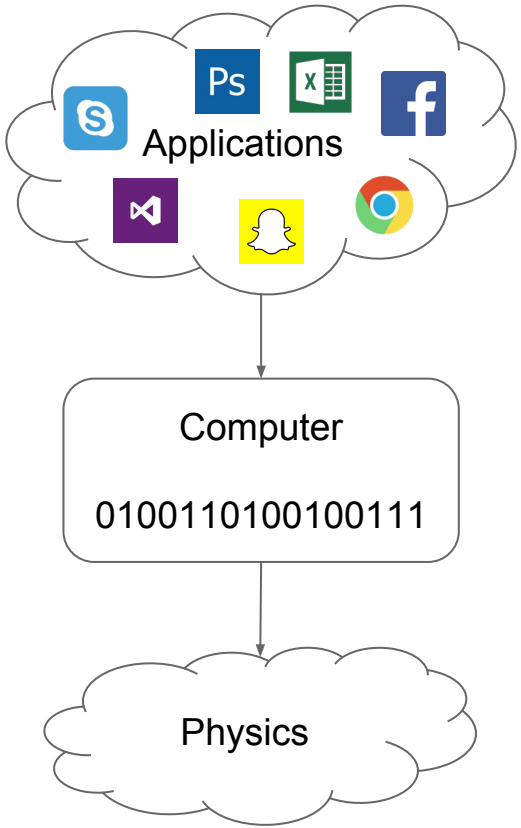
noun - a magical device that identifies our drawings of everyday objects

Computer

/kəmˈpyo͞odər/

noun - a series of abstraction layers that allow powerful operations to be done without thinking about them

How does a computer work?



High Level Language
Operating System
Compiler
Virtual Machine
Assembler
Binary Machine Language

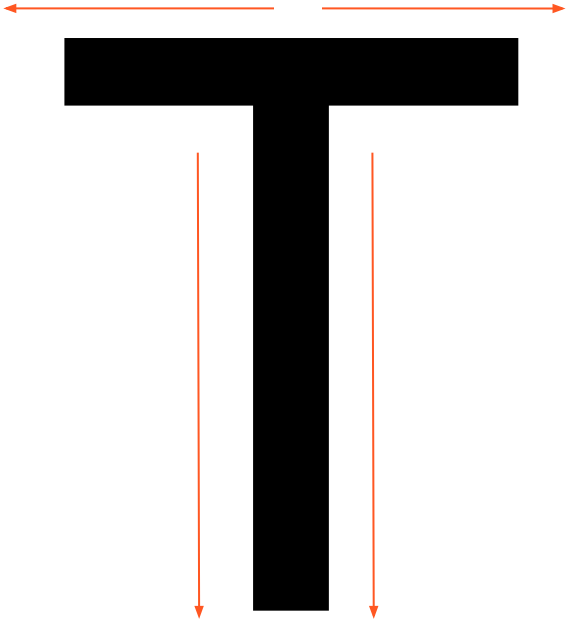
ISA

Computer Architecture
ALU
Memory
Boolean Arithmetic
Sequential Logic
Boolean Logic

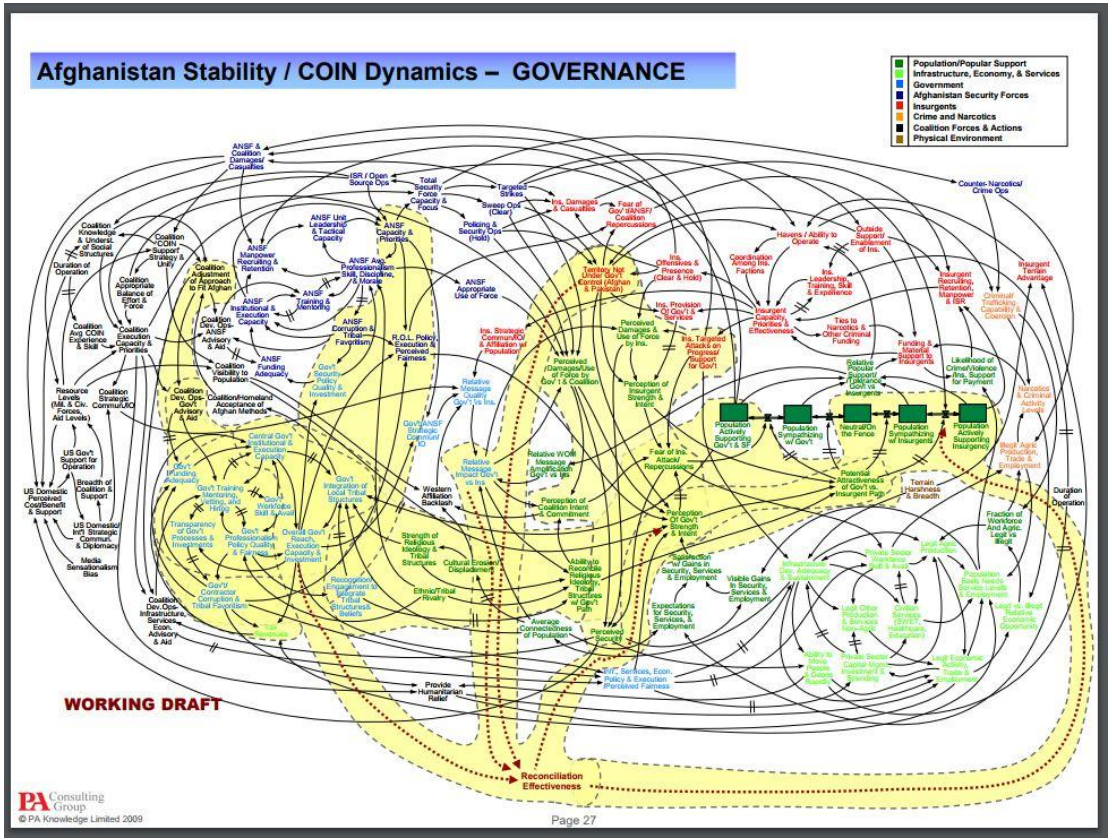
Software

Hardware

T-shaped expertise



What is an abstraction layer?



Group Activity: Language

1) Even though there were many cookies on the dish, I only ate three.

2) He left home early in the morning, so he wouldn't miss the train.

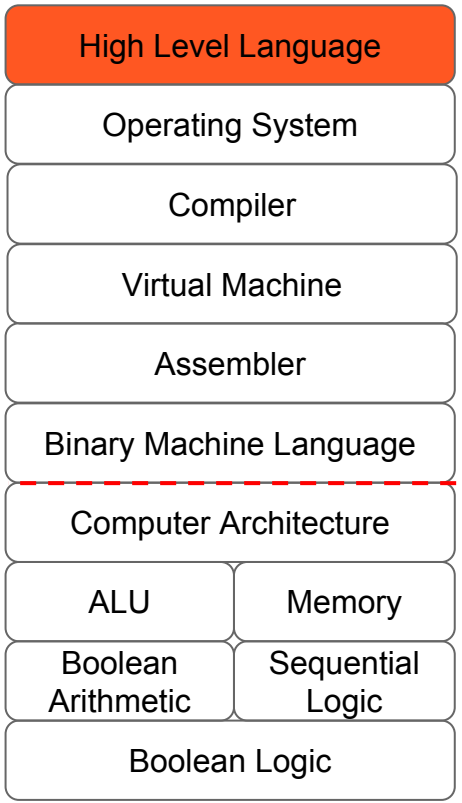
Impact of an abstraction layer?

The level of abstraction at which you work ultimately determines the level of abstraction at which you are able to think.

“Civilization progresses by extending the number of operations that we can perform without thinking about them.”

- Alfred North Whitehead - Mathematician and Philosopher

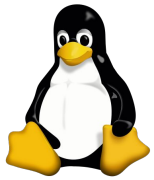
High Level Language




```
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

http://norvig.com/spell-correct.html

Operating System



- High Level Language
- **Operating System**
- Compiler
- Virtual Machine
- Assembler
- Binary Machine Language
- Computer Architecture
- ALU
- Memory
- Boolean Arithmetic
- Sequential Logic
- Boolean Logic

```
NtOpenChannel
NtOpenDirectoryObject
NtOpenEnlistment
NtOpenEvent
NtOpenEventPair
NtOpenFile
NtOpenIoCompletion
NtOpenJobObject
NtOpenKey
NtOpenKeyEx
NtOpenKeyTransacted
NtOpenKeyTransactedEx
NtOpenKeyedEvent
NtOpenMutant
NtOpenObjectAuditAlarm
NtOpenPartition
NtOpenPrivateNamespace
NtOpenProcess
NtOpenProcessToken
NtOpenProcessTokenEx
NtOpenRegistryTransaction
NtOpenResourceManager
```

http://j00ru.vexillium.org/ntapi/

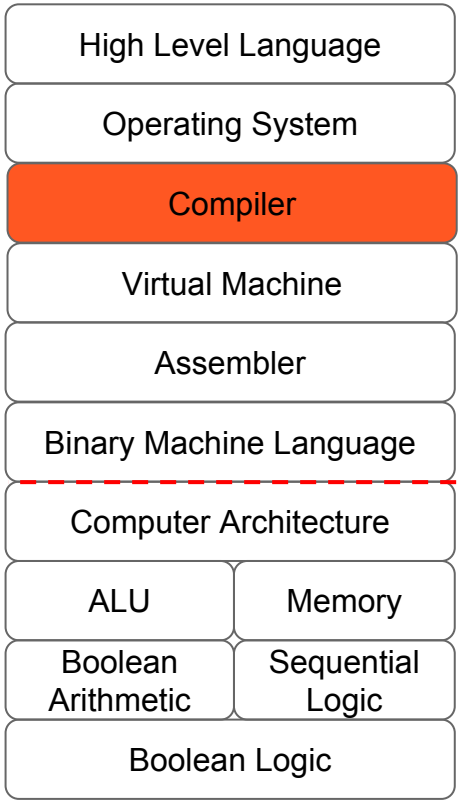
```
accept(2)
accept4(2)
access(2)
acct(2)
add_key(2)
adjtimex(2)
alarm(2)
alloc_hugepages(2)
bdflush(2)

bind(2)
bpf(2)
brk(2)
cacheflush(2)
capget(2)
capset(2)
chdir(2)
chmod(2)
chown(2)
```

`man syscalls`

```
{ int nosys(void); }
{ void exit(int rval)
{ int fork(void); }
{ user_ssize_t read(i
{ user_ssize_t write(
{ int open(user_addr_
{ int close(int fd);
{ int wait4(int pid,
{ int nosys(void); }
{ int link(user_addr_
{ int unlink(user_addr
{ int nosys(void); }
{ int chdir(user_addr
{ int fchdir(int fd);
{ int mknod(user_addr
{ int chmod(user_addr
{ int chown(user_addr
{ int nosys(void); }
```

https://opensource.apple.com/source/xnu/xnu-1504.3.12/bsd/kern/syscalls.master

Compiler



hello.c

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
   printf("Hello, World!");
}
```

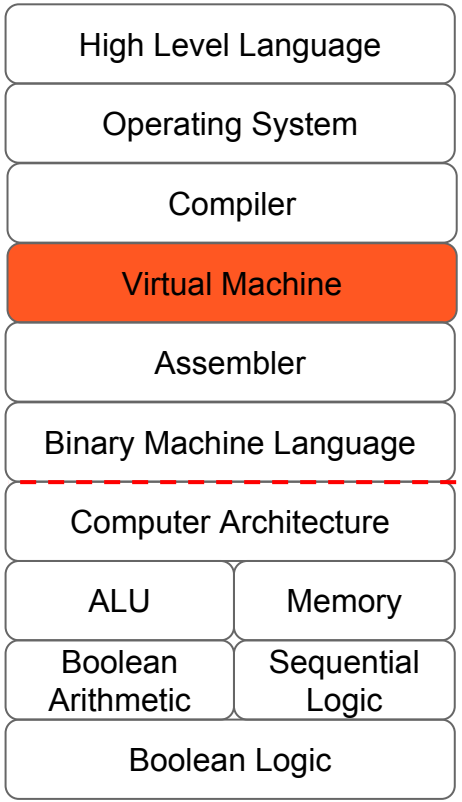
> gcc -S -masm=intel hello.c

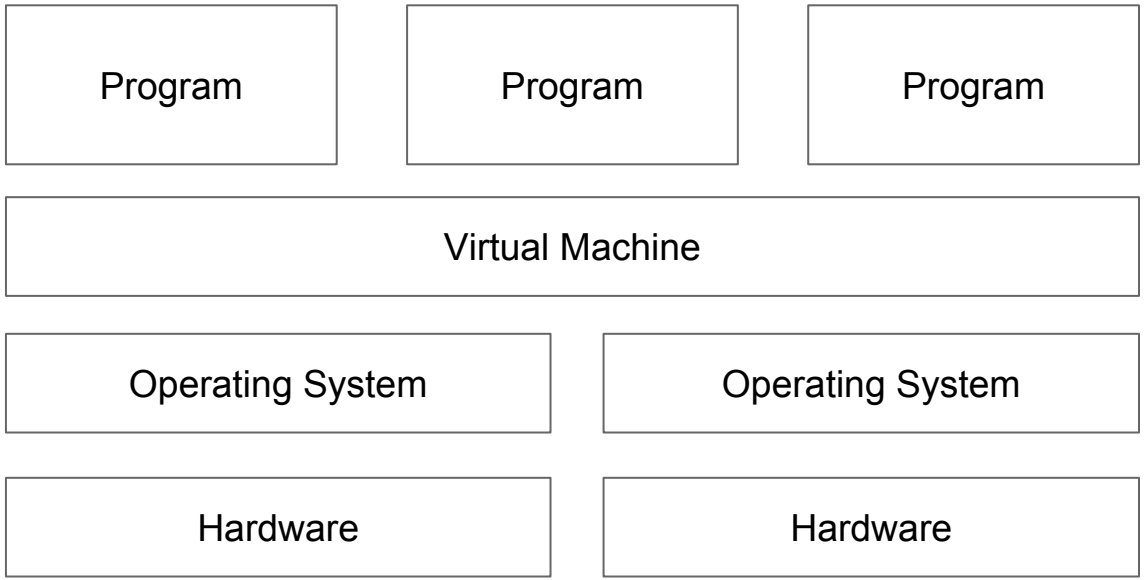
Compiler

hello.s

```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        movabs  rax, .L.str
        mov     dword ptr [rbp - 4], edi
        mov     qword ptr [rbp - 16], rsi
        mov     rdi, rax
        mov     al, 0
        call    printf
        xor     ecx, ecx
        mov     dword ptr [rbp - 20], eax
        mov     eax, ecx
        add     rsp, 32
        pop     rbp
        ret

.L.str:
        .asciz  "Hello, World!"
```

Virtual Machine





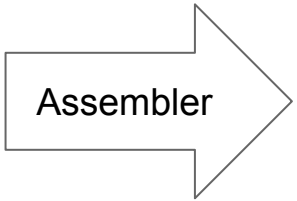
Assembler

- High Level Language
- Operating System
- Compiler
- Virtual Machine
- **Assembler**
- Binary Machine Language
- Computer Architecture
- ALU
- Memory
- Boolean Arithmetic
- Sequential Logic
- Boolean Logic

hello.s

```
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 32
        movabs  rax, .L.str
        mov     dword ptr [rbp - 4], edi
        mov     qword ptr [rbp - 16], rsi
        mov     rdi, rax
        mov     al, 0
        call    printf
        xor     ecx, ecx
        mov     dword ptr [rbp - 20], eax
        mov     eax, ecx
        add     rsp, 32
        pop     rbp
        ret

.L.str:
        .asciz  "Hello, World!"
```

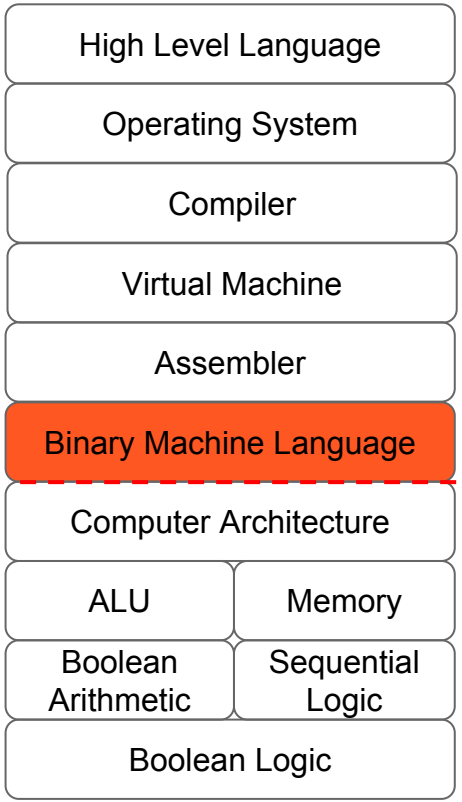
```
> gcc hello.s
```



a.out

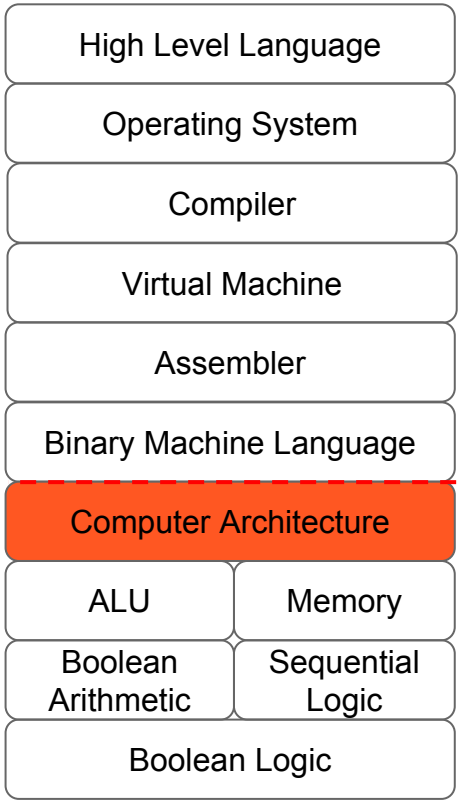
```
415e 415f c390 662e 0f1f 8400 0000 0000
f3c3 0000 4883 ec08 4883 c408 c300 0000
0100 0200 4865 6c6c 6f2c 2057 6f72 6c64
2100 0000 011b 033b 3000 0000 0500 0000
0cfe ffff 7c00 0000 4cfe ffff 4c00 0000
42ff ffff a400 0000 6cff ffff c400 0000
dcff ffff 0c01 0000 1400 0000 0000 0000
017a 5200 0178 1001 1b0c 0708 9001 0710
```

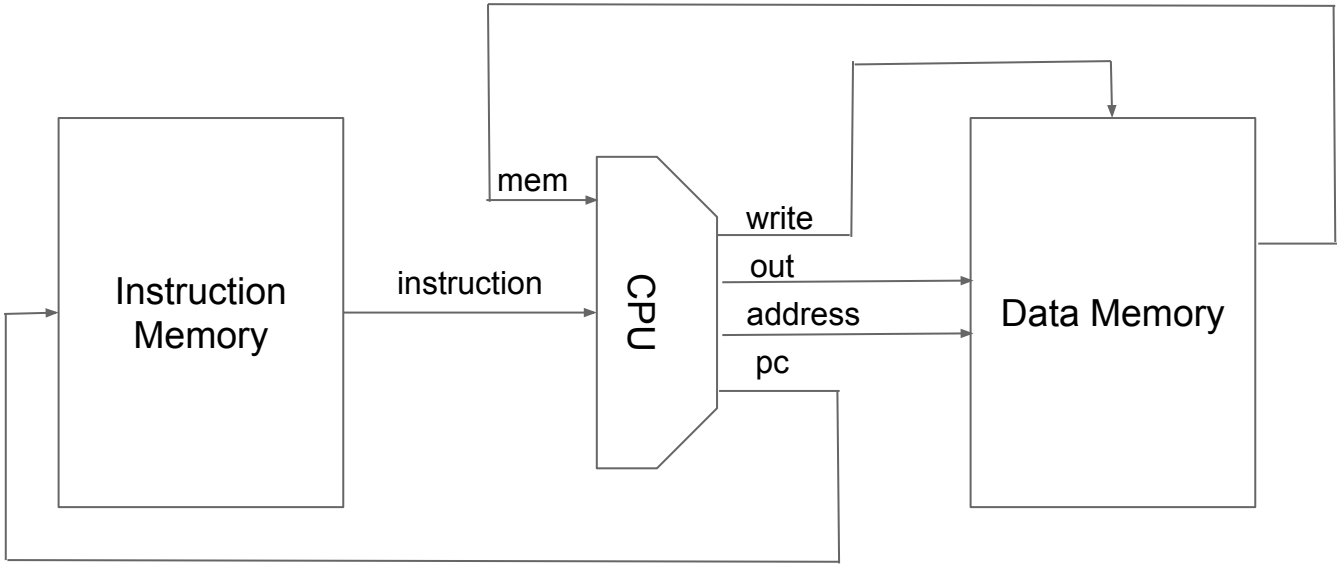
Binary Machine Language



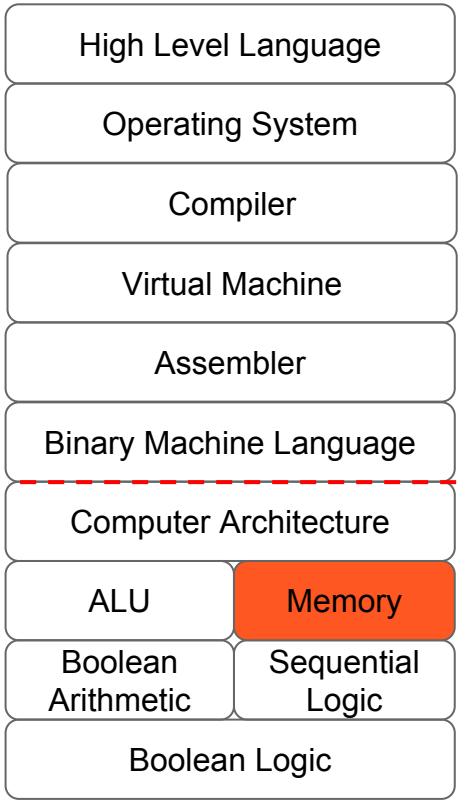

```
01100111 01101100 01101111 01100010 01100001 01101100
01011111 01100100 01110100 01101111 01110010 01110011
01011111 01100001 01110101 01111000 01011111 01100110
01101001 01101110 01101001 01011111 01100001 01110010
01110010 01100001 01111001 01011111 01100101 01101110
01110100 01110010 01111001 00000000 01100110 01110010
01100001 01101101 01100101 01011111 01100100 01110101
01101101 01101101 01111001 00000000 01011111 01011111
01100110 01110010 01100001 01101101 01100101 01011111
01100100 01110101 01101101 01101101 01111001 01011111
01100100 01110101 01101101 01101101 01111001 01011111
01101001 01101110 01101001 01110100 01011111 01100001
01110010 01110010 01100001 01111001 01011111 01100101
01101110 01110100 01110010 01111001 00000000 01101000
01100101 01101100 01101100 01101111 00101110 01100011
00000000 01011111 01011111 01000110 01010010 01000001
01001101 01000101 01011111 01000101 01001110 01000100
```

Computer Architecture

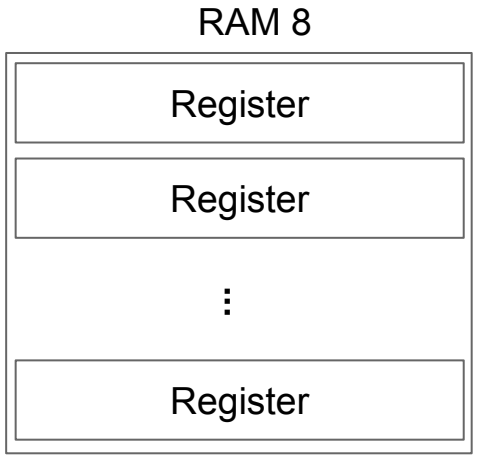


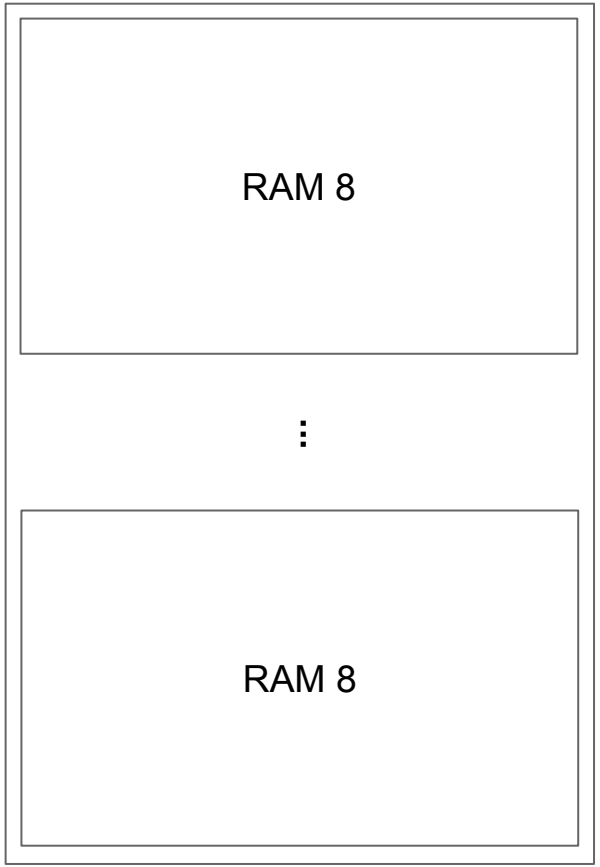


Memory


High Level Language
Operating System
Compiler
Virtual Machine
Assembler
Binary Machine Language
Computer Architecture
ALU
Memory
Boolean Arithmetic
Sequential Logic
Boolean Logic

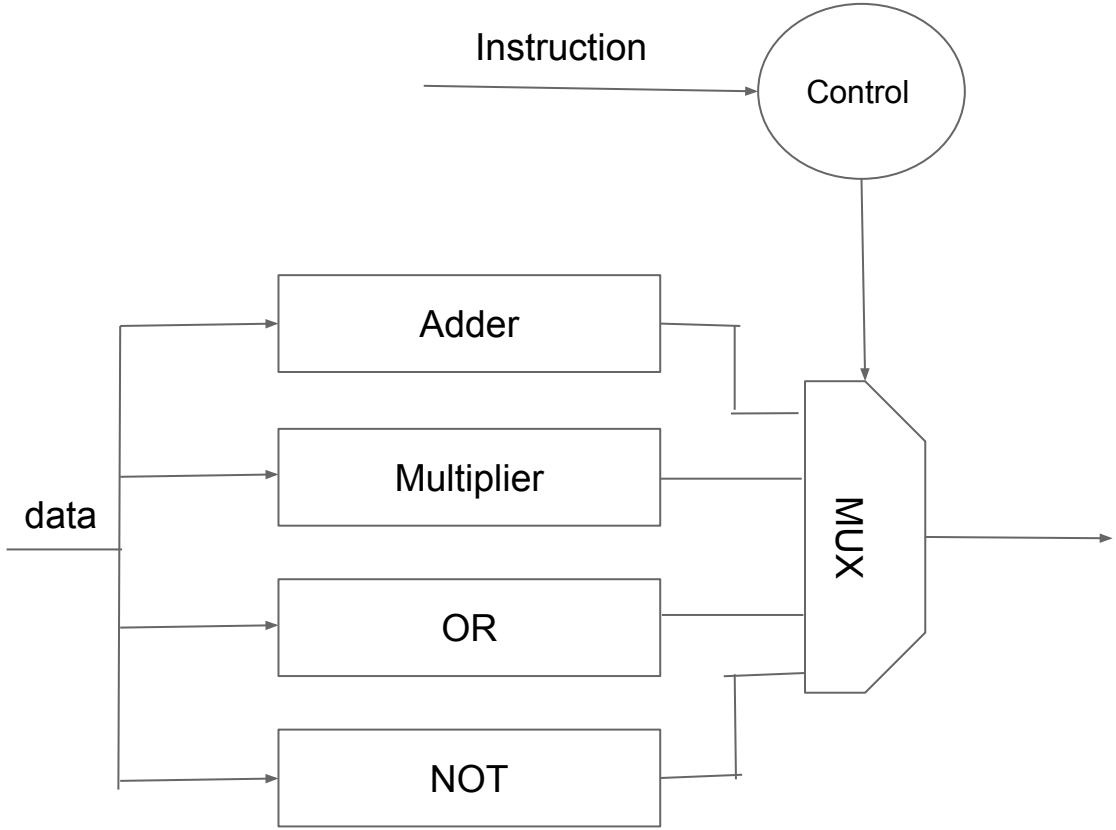

Register
DFF
DFF
...
DFF


RAM 8
Register
Register
Register

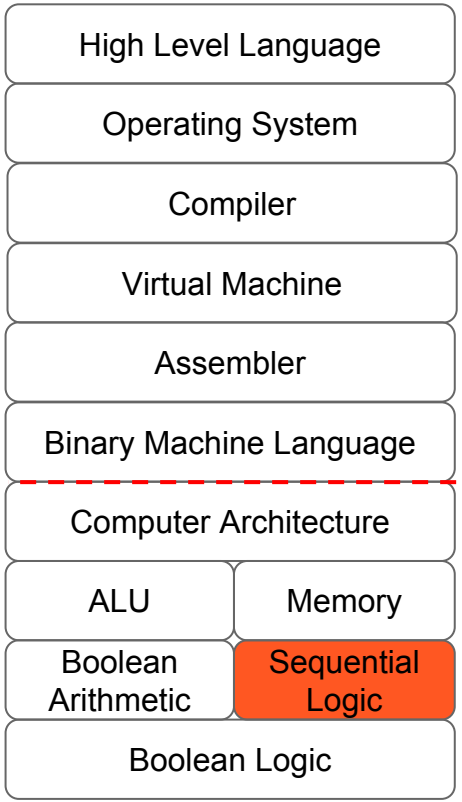

RAM 8
RAM 8

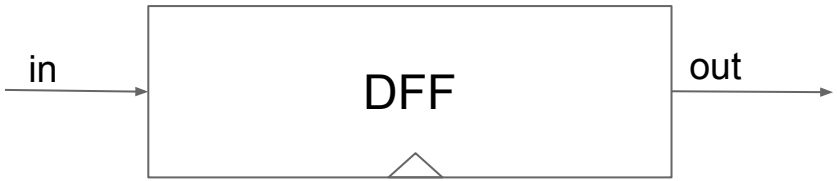
ALU

- High Level Language
- Operating System
- Compiler
- Virtual Machine
- Assembler
- Binary Machine Language
- Computer Architecture
- **ALU** | Memory
- Boolean Arithmetic | Sequential Logic
- Boolean Logic



Sequential Logic





$$out(t) = in(t-1)$$

Boolean Arithmetic

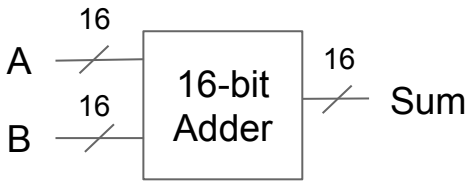
- High Level Language
- Operating System
- Compiler
- Virtual Machine
- Assembler
- Binary Machine Language

- Computer Architecture
- ALU
- Memory
- Boolean Arithmetic
- Sequential Logic
- Boolean Logic

2's complement binary values

0	0000		
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001
		-8	1000

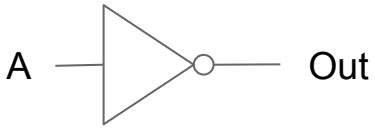
… 1011
+ … 0010
… 1101



Boolean Logic

- High Level Language
- Operating System
- Compiler
- Virtual Machine
- Assembler
- Binary Machine Language

- Computer Architecture
- ALU
- Memory
- Boolean Arithmetic
- Sequential Logic
- Boolean Logic



A	Out
0	1
1	0

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

- AND
- OR
- NOT
- XOR
- NAND
- NOR
- MUX
- DEMUX

Code

Why?

Credits

The Elements of Computing Systems by Noam Nisan and Shimon Schocken
http://nand2tetris.org/
https://gcc.godbolt.org/
http://norvig.com/spell-correct.html

Questions?

RETRO

1

What went well?

2

What didn’t go well?

3

What can be improved?

Build Your Creative Thinking Toolbox!

Goals, uses and intended outcomes of various types of frameworks.

PLUGGED IN

AUGUST 22, 2017 5:30-7:30PM

pillar TECHNOLOGY

RSVP: bit.ly/ihrpithink

Thank You

Michael Patterson <mpatterson@pillartechnology.com>

PILLAR PLUGGEDIN