Admin

- Fifth quiz opens on Monday

- Sudoku assignment:
  - Deadline: **1st March 4pm**
  - Cut-off date: **15th March 4pm**
  - **You can submit your work without penalty until 15th March 4pm** - I recommend you submit asap so you don't fall behind

- Virtual Contact Hours
  - New (non-assessed) worksheet on Monday
  - We will have a meeting for the new worksheet
  - You may ask for help with Sudoku assignment in VCH but make it clear in your question

*The number of **sequential operations** (time-steps) is lower in one algorithm than another*

What are these operations?

Operations performed by a Random Access Machine

For Review Seminar

What happens when you run your JavaScript code in Node.js

```
function lengthQueue(queue) {
    if (queue.isEmpty()) {
        return 0;
    }
    var secondQueue = new Queue();
    var count = 0;
    while (queue.isEmpty() === false) {
        count++;
        secondQueue.enqueue(queue.head());
        queue.dequeue();
    }
    while (secondQueue.isEmpty() === false) {
        queue.enqueue(secondQueue.head());
        secondQueue.dequeue();
    }
    return count;
}
```

“Text file” stored in memory

What happens when you run your JavaScript code in Node.js

```
function lengthQueue(queue) {
    if (queue.isEmpty()) {
        return 0;
    }
    var secondQueue = new Queue();
    var count = 0;
    while (queue.isEmpty() === false) {
        count++;
        secondQueue.enqueue(queue.head());
        queue.dequeue();
    }
    while (secondQueue.isEmpty() === false) {
        queue.enqueue(secondQueue.head());
        secondQueue.dequeue();
    }
    return count;
}
```

"Text file" stored in memory

V8 JavaScript Engine
Compiles to Machine Code
"Just-in-time Compilation"

Node <yourfile>.js

What happens when you run your JavaScript code in Node.js

```
function lengthQueue(queue) {
    if (queue.isEmpty()) {
        return 0;
    }
    var secondQueue = new Queue();
    var count = 0;
    while (queue.isEmpty() === false) {
        count++;
        secondQueue.enqueue(queue.head());
        queue.dequeue();
    }
    while (secondQueue.isEmpty() === false) {
        queue.enqueue(secondQueue.head());
        secondQueue.dequeue();
    }
    return count;
}
```

"Text file" stored in memory

V8 JavaScript Engine
Compiles to Machine Code
"Just-in-time Compilation"

Node <yourfile>.js

```
80483b4:    55                      push   %ebp
80483b5:    89 e5                   mov    %esp,%ebp
80483b7:    83 e4 f0                and    $0xfffffff0,%esp
80483ba:    83 ec 20                sub    $0x20,%esp
80483bd:    c7 44 24 1c 00 00 00    movl   $0x0,0x1c(%esp)
80483c4:    00
80483c5:    eb 11                   jmp    80483d8 <main+0x24>
80483c7:    c7 04 24 b0 84 04 08    movl   $0x80484b0,(%esp)
80483ce:    e8 1d ff ff ff          call   80482f0 <puts@plt>
80483d3:    83 44 24 1c 01          addl   $0x1,0x1c(%esp)
80483d8:    83 7c 24 1c 09          cmpl   $0x9,0x1c(%esp)
80483dd:    7e e8                   jle    80483c7 <main+0x13>
80483df:    b8 00 00 00 00          mov    $0x0,%eax
80483e4:    c9                      leave
80483e5:    c3                      ret
80483e6:    90                      nop
80483e7:    90                      nop
80483e8:    90                      nop
80483e9:    90                      nop
80483ea:    90                      nop
```

Set of *basic instructions* stored in memory
Everything is basically in hexadecimal

What happens when you run your JavaScript code in Node.js

```
function lengthQueue(queue) {
    if (queue.isEmpty()) {
        return 0;
    }
    va
    va
    wh
                                    ));

    }
    wh                              lse) {
                                    ));
        secondQueue.dequeue();
    }
    return count;
}
```

Readable by humans

"Text file" stored in memory

V8 JavaScript Engine
Compiles to Machine Code
"Just-in-time Compilation"

Node <yourfile>.js

```
80483b4:    55                      push   %ebp
80483b5:    89 e5                   mov    %esp,%ebp
80483b7:    83 e4 f0                and    $0xfffffff0,%esp
80483ba:    83 ec 20                sub    $0x20,%esp
80483bd:    c7 44 24 1c 00 00 00    movl   $0x0,0x1c(%esp)
80483c4:
80483c5:                                   <main+0x24>
80483c7:                                   00,(%esp)
80483ce:                                   <puts@plt>
80483d3:                                   c(%esp)
80483d8:                                   c(%esp)
80483dd:                                   <main+0x13>
80483df:                                   x
80483e4:
80483e5:
80483e6:
80483e7:    90                      nop
80483e8:    90                      nop
80483e9:    90                      nop
80483ea:    90                      nop
```

Readable by microprocessors

Set of *basic instructions* stored in memory
Everything is basically in hexadecimal

What happens when you run your JavaScript code in Node.js

```
function lengthQueue(queue) {
    if (queue.isEmpty()) {
        return 0;
    }
    va
    va
    wh
                                    ));

    }
    wh                              lse) {
                                    ));
        secondQueue.dequeue();
    }
    return count;
}
```

Readable by humans

"Text file" **stored in memory**

V8 JavaScript Engine
Compiles to Machine Code
"Just-in-time Compilation"

Node <yourfile>.js

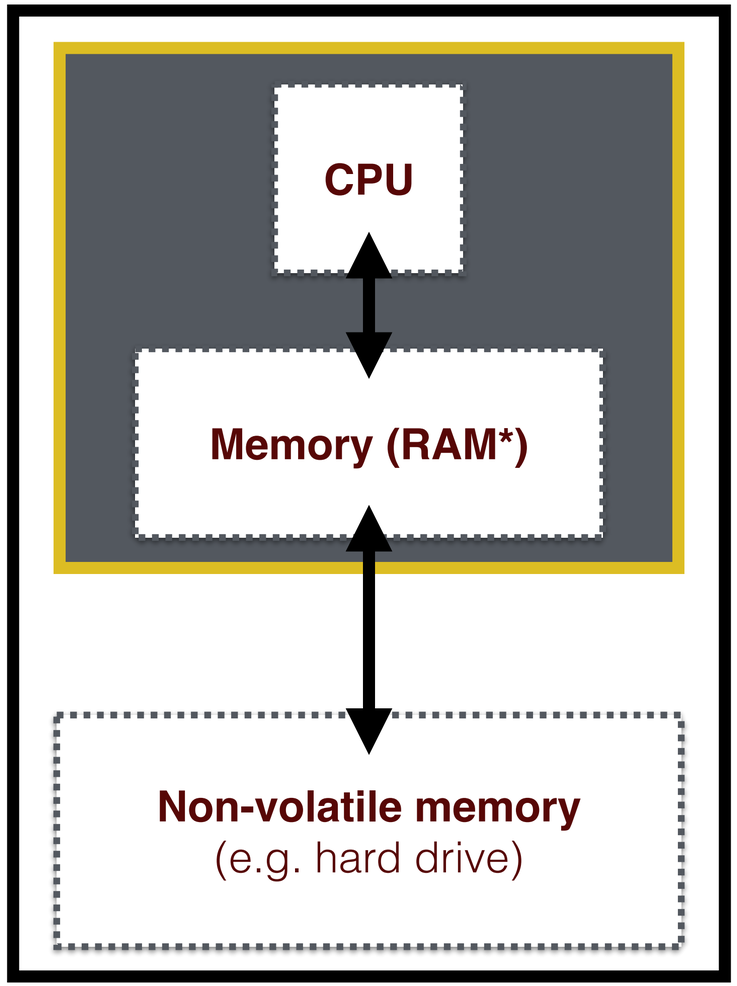
```
80483b4:    55                      push   %ebp
80483b5:    89 e5                   mov    %esp,%ebp
80483b7:    83 e4 f0                and    $0xfffffff0,%esp
80483ba:    83 ec 20                sub    $0x20,%esp
80483bd:    c7 44 24 1c 00 00 00    movl   $0x0,0x1c(%esp)
80483c4:
80483c5:                                                <main+0x24>
80483c7:                                                00,(%esp)
80483ce:                                                <puts@plt>
80483d3:                                                c(%esp)
80483d8:                                                c(%esp)
80483dd:                                                <main+0x13>
80483df:                                                x
80483e4:
80483e5:
80483e6:
80483e7:    90                      nop
80483e8:    90                      nop
80483e9:    90                      nop
80483ea:    90                      nop
```

Readable by microprocessors

Programs are data:
Instructions stored in memory

Set of basic instructions **stored in memory**
Everything is basically in hexadecimal

We abstract away details to an architecture

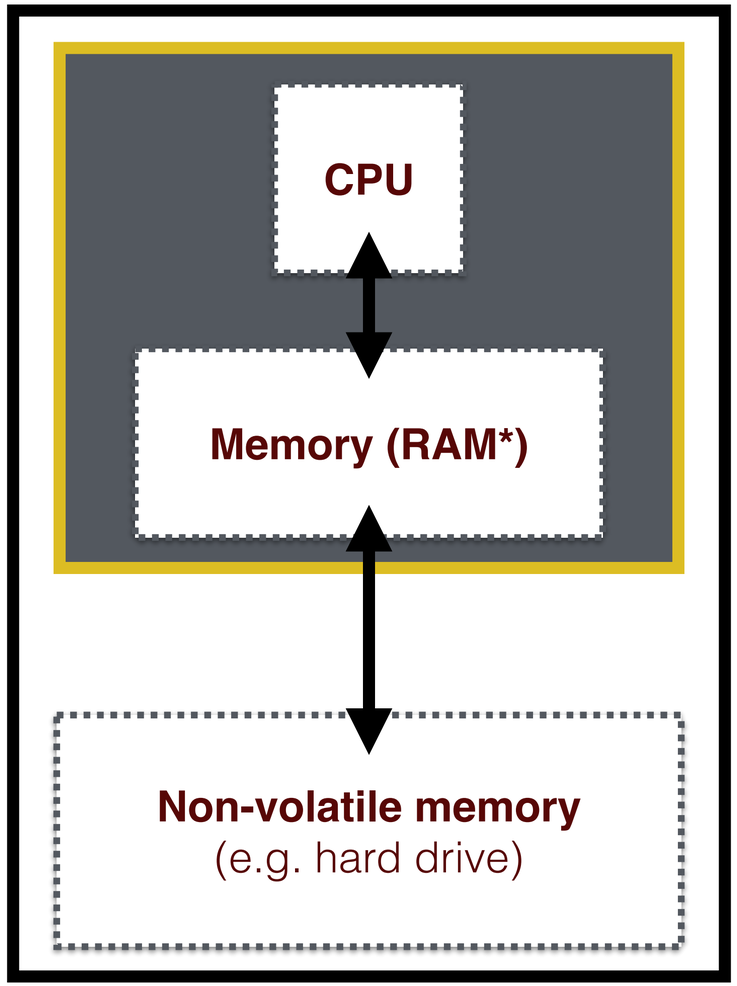


***Random Access Memory**

We abstract away details to an architecture

Machine code instructions copied to here when needed

Initial js and machine code stored here

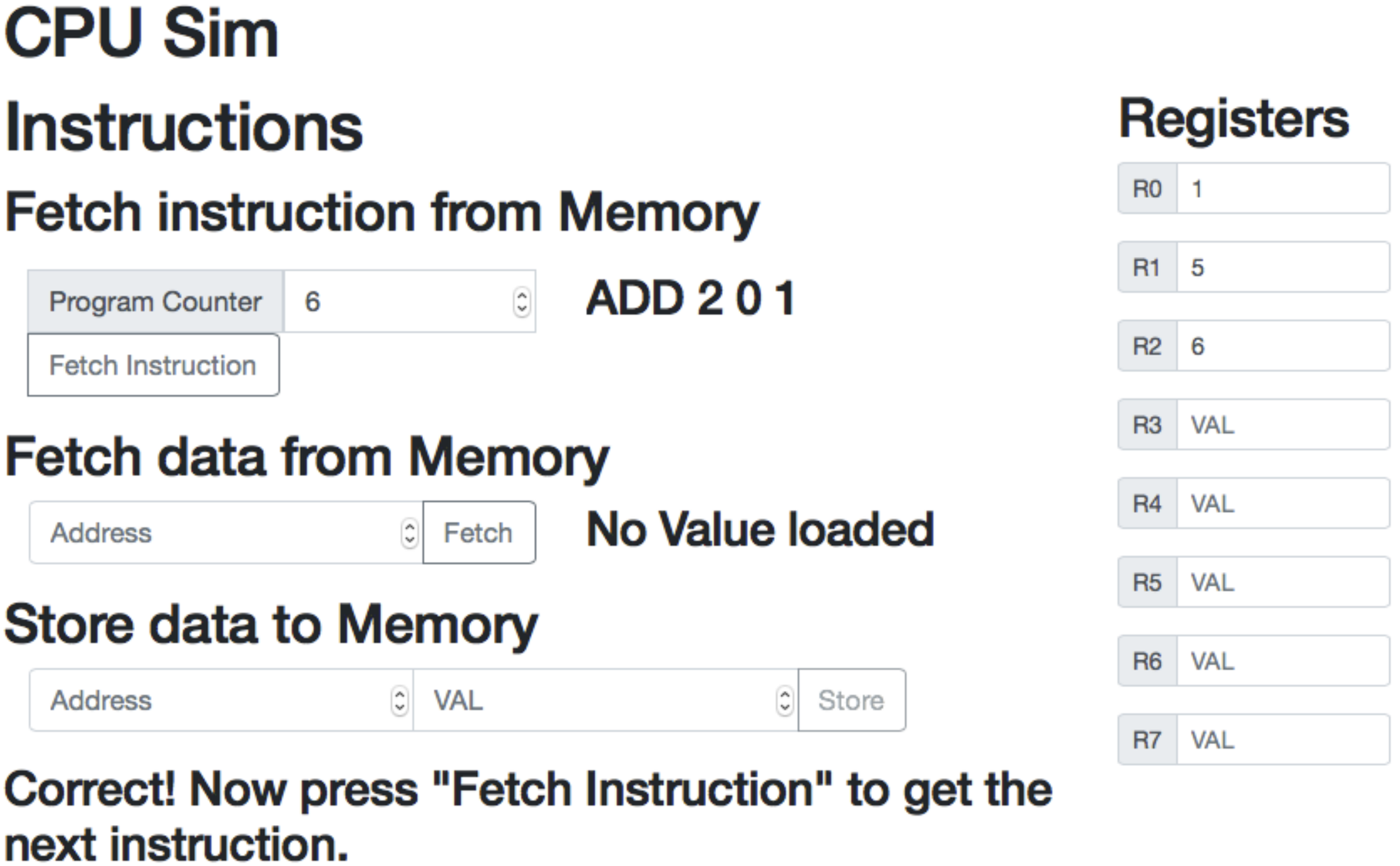


Machine code instructions implemented by this

***Random Access Memory**

CPU Simulator (How Computers Work)

http://igor.doc.gold.ac.uk/~afior002/cpu_simulator/index.html



What are the main components?

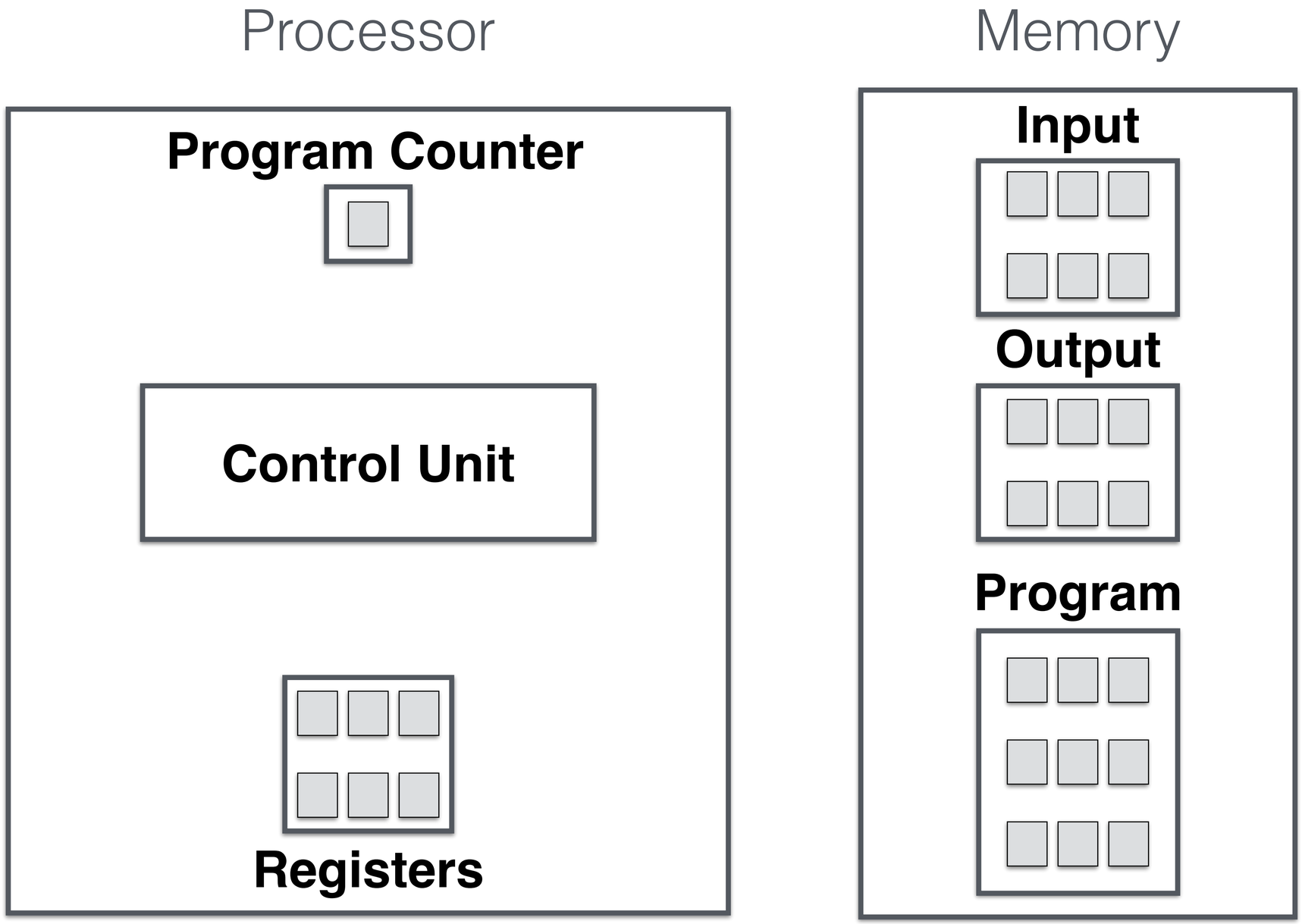
We can distill this architecture to two basic elements: processor and memory

Give an abstract model of computation to study the performance of algorithms

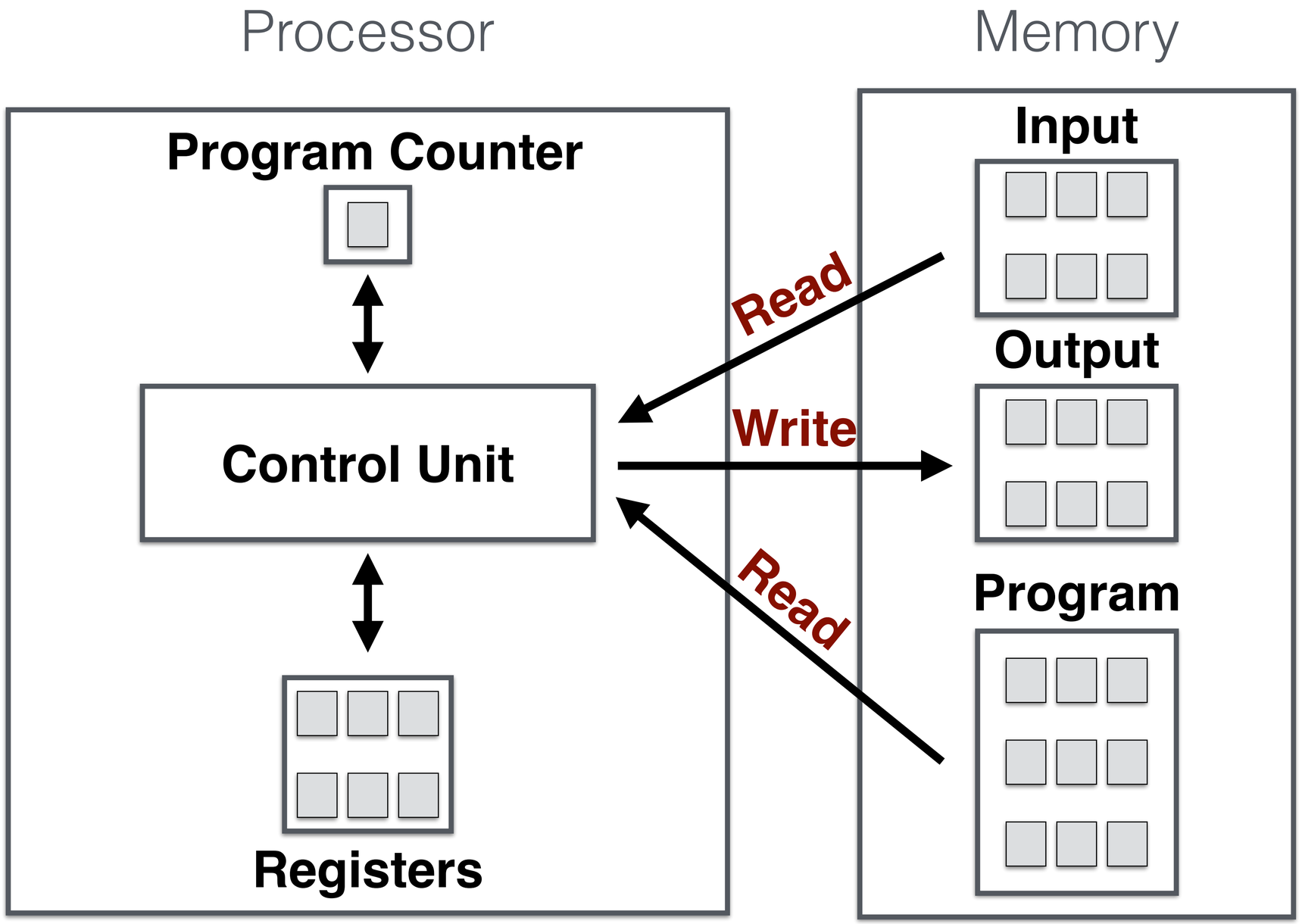
Abstract model is called Random-Access Machine model

Why do we need an abstract model?

Random Access Machine



Random Access Machine

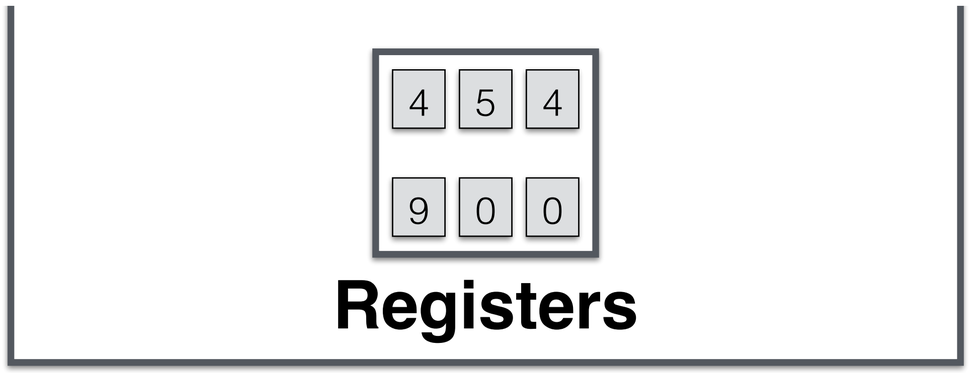


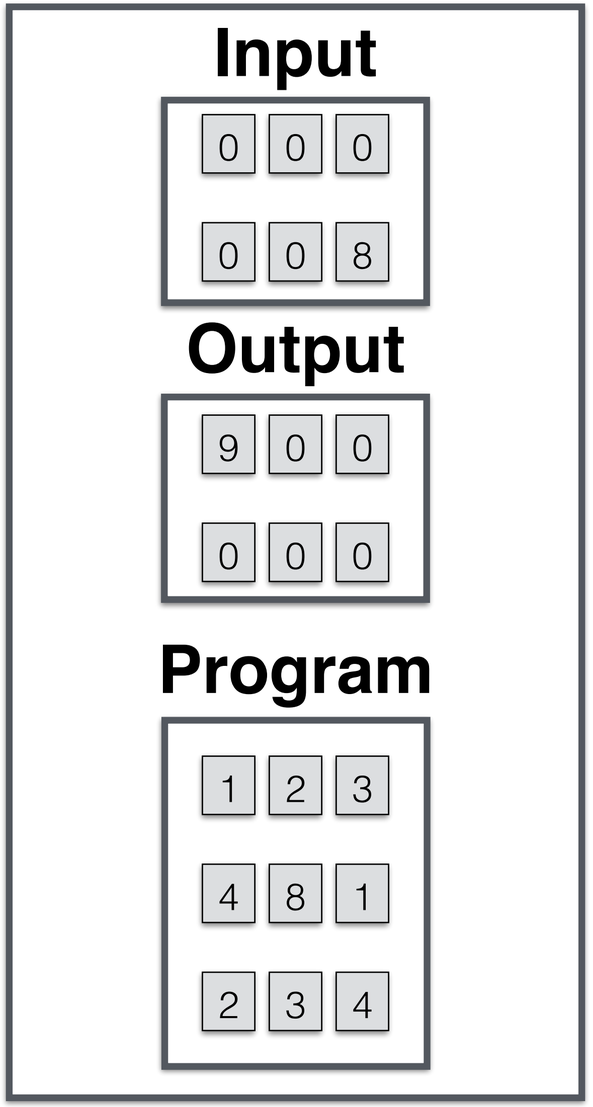
Random Access Machine

Each memory unit can store an arbitrary integer

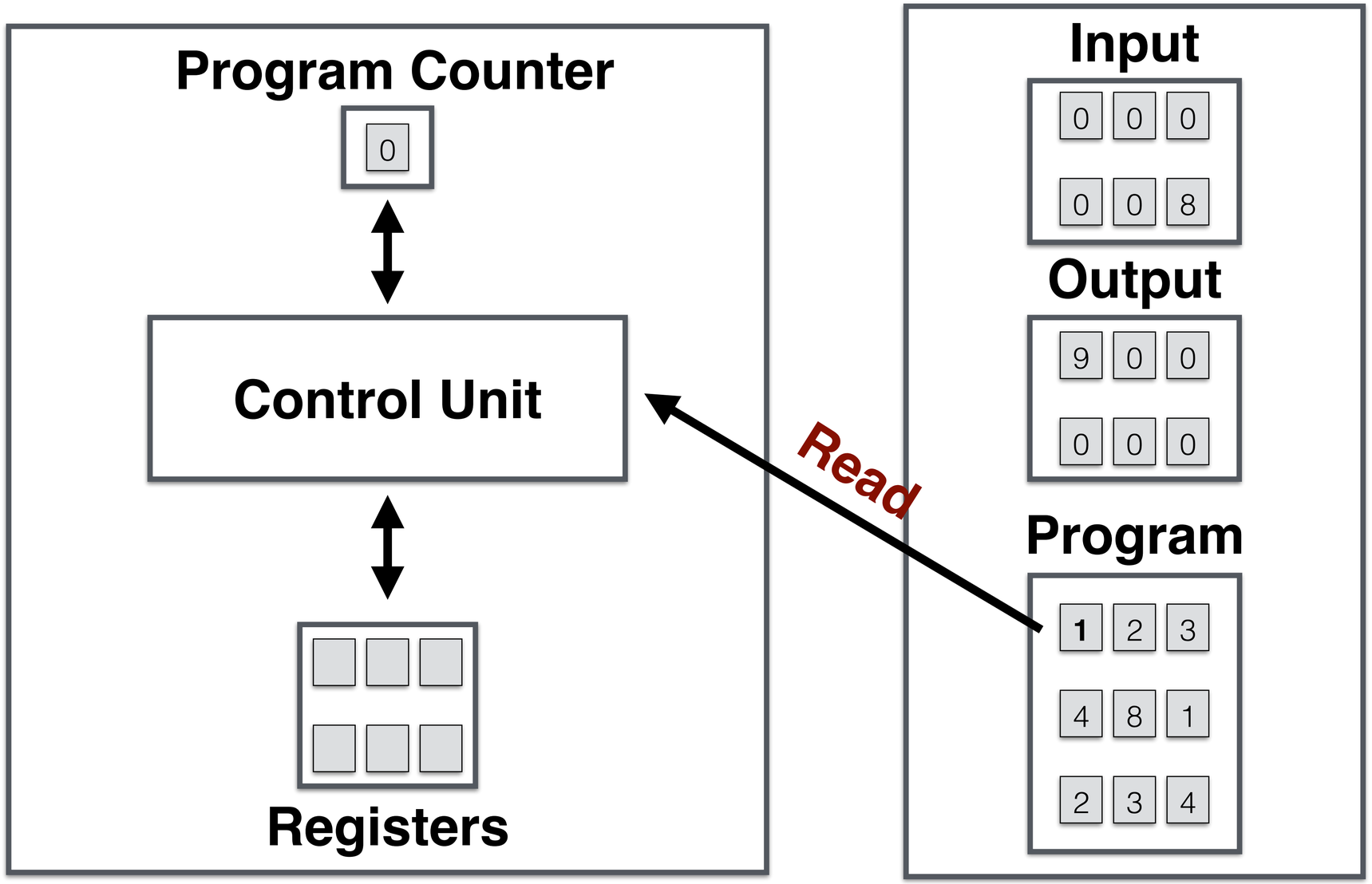
Must be non-negative for Program Counter

Depending on values, Control Unit does an operation

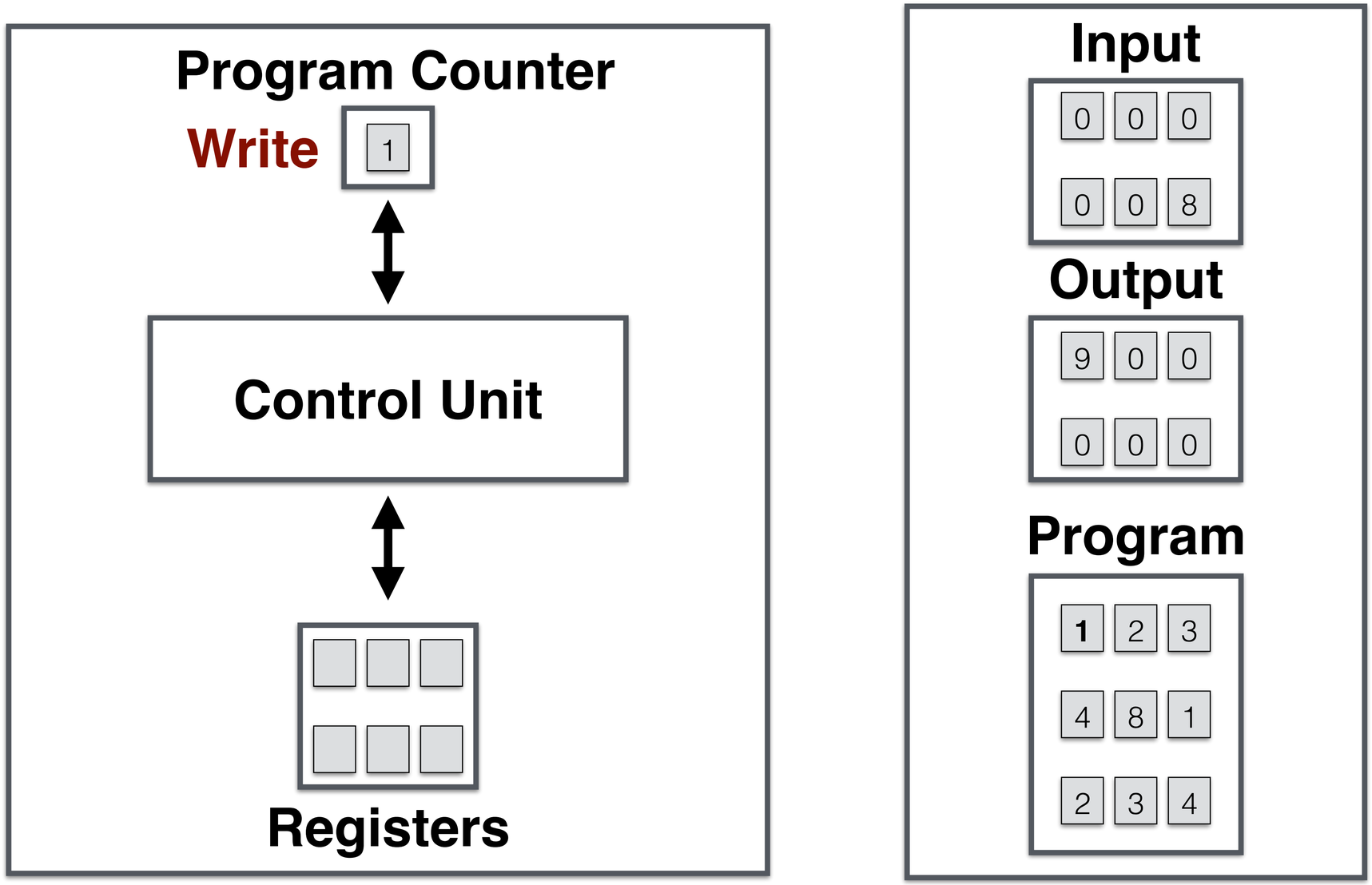




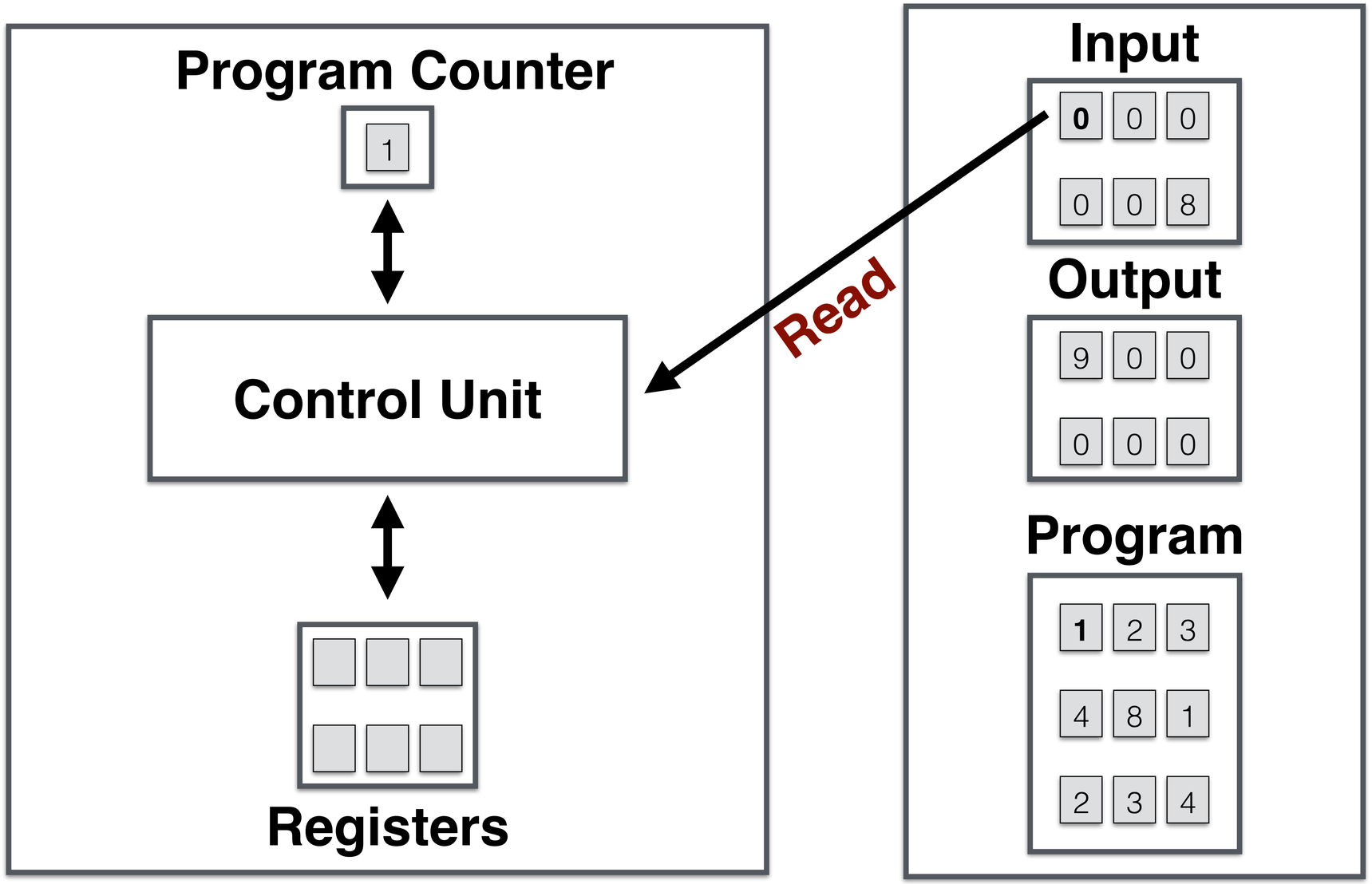
Random Access Machine



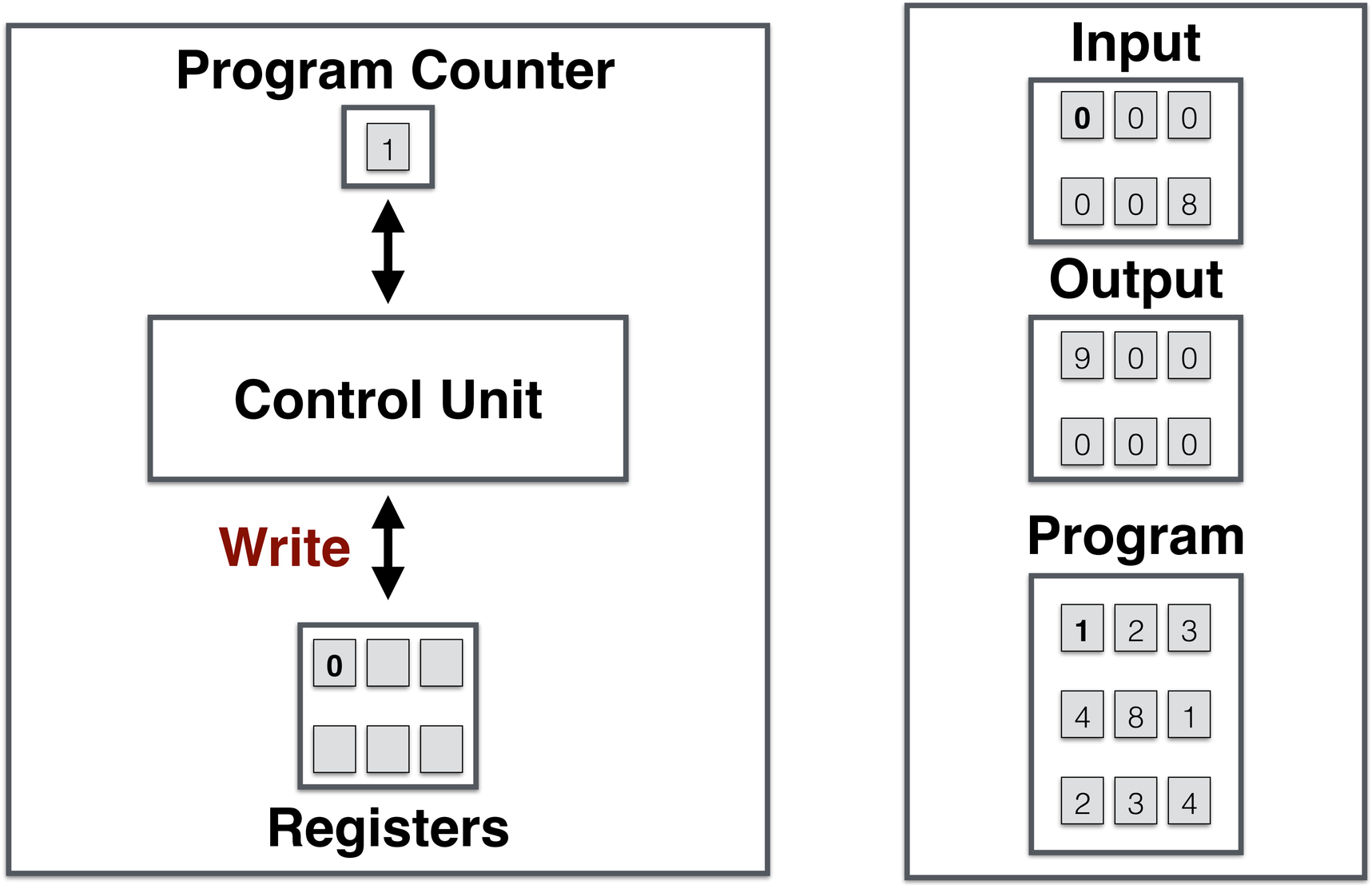
Random Access Machine



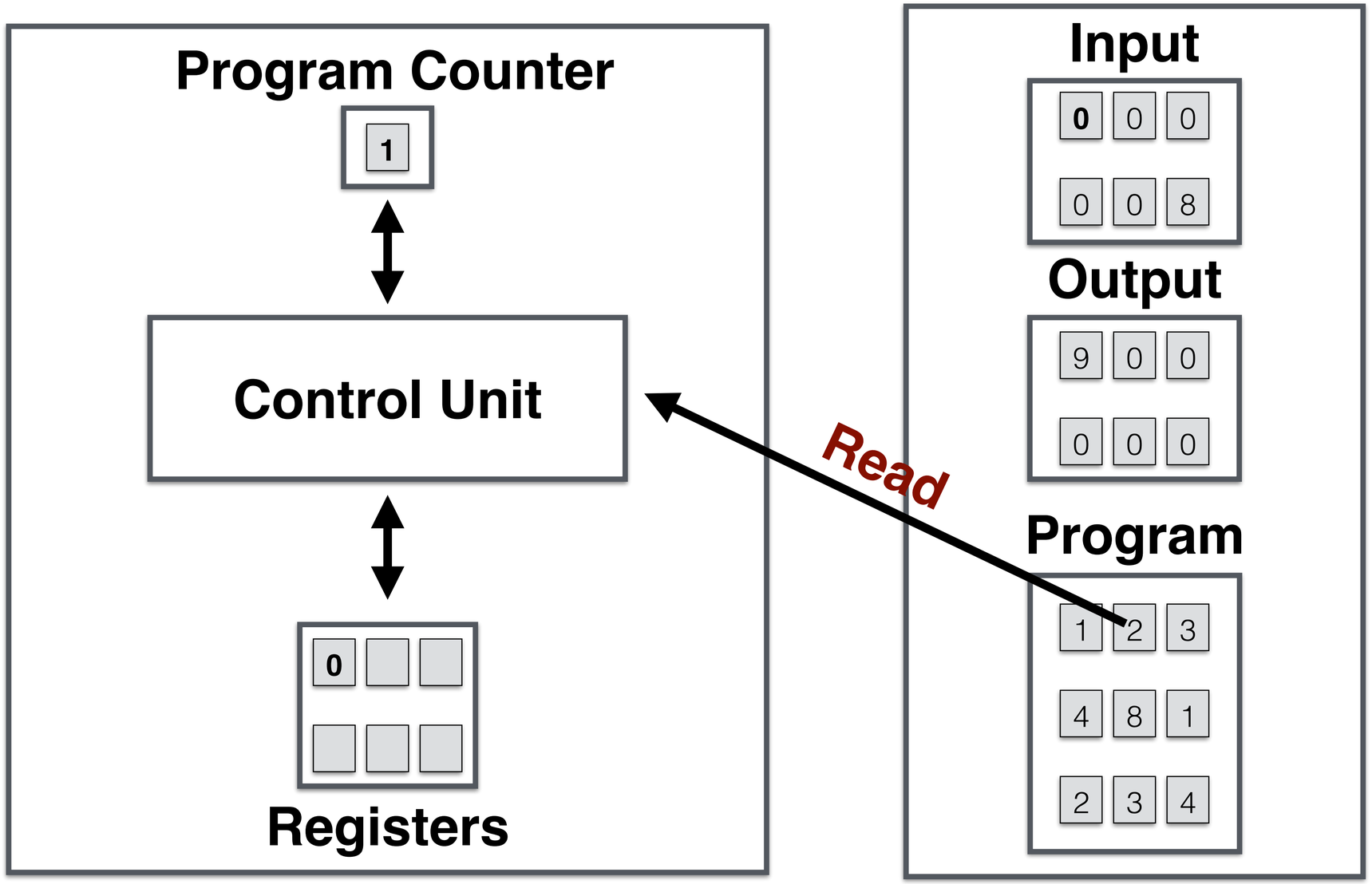
Random Access Machine



Random Access Machine



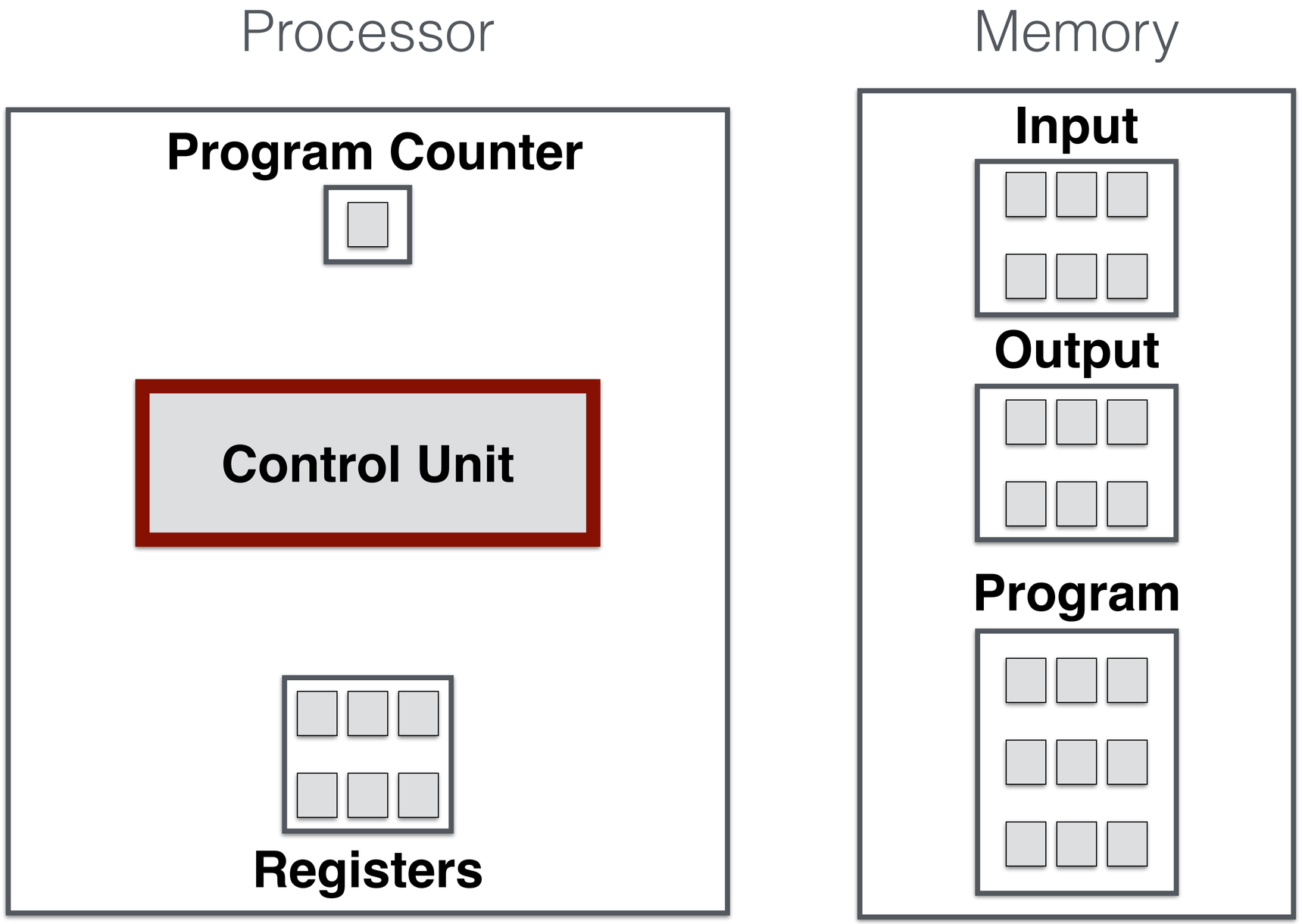
Random Access Machine



The Control Unit can read and write values in memory units if the index of that unit is specified in the program

The Program Counter keeps track of the index of the next instruction to be carried out

The Control Unit



Random Access Machine

Control Unit can:

Read and write values in single memory units

Do simple arithmetic (add, subtract, multiply, divide)

Random Access Machine

Control Unit can:

Read and write values in single memory units

Do simple arithmetic (add, subtract, multiply, divide)

Each individual operation done in one time-step

Control Unit can:

Read and write values in single memory units

Do simple arithmetic (add, subtract, multiply, divide)

Above operations done in one time-step

Perform conditional operations: if then conditionals

One time-step for comparison in if statement

e.g. *if (a*b == 0)* takes two time-steps to compute

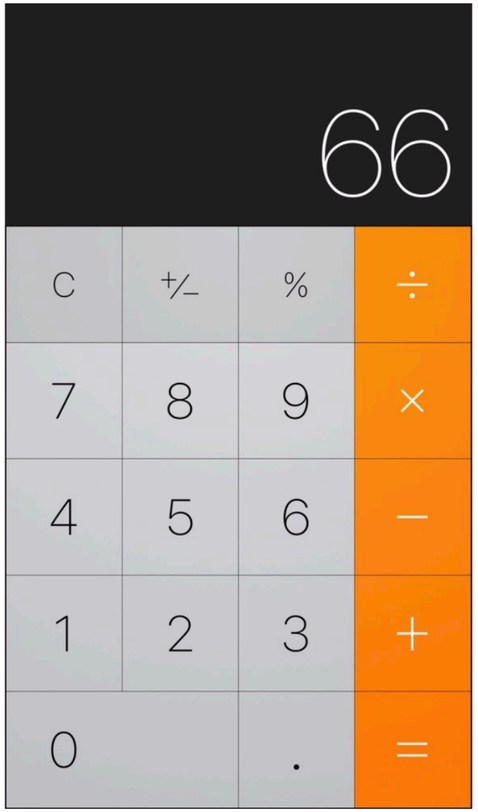
The “size” of the program stored in memory is not necessarily the number of time-steps in an implementation

Control Unit can:

Read, write, and copy values of memory units

Do simple arithmetic (add, subtract, multiply, divide)

Do conditional operations (if then)



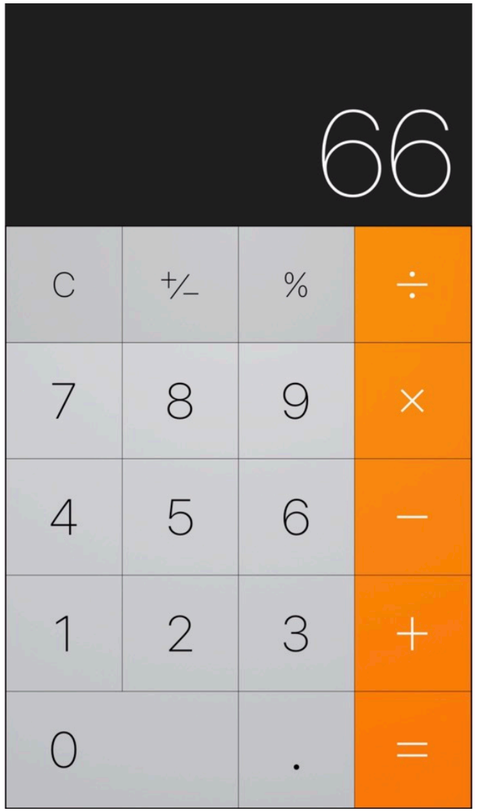
Control Unit can:

Read, write, and copy values of memory units

Do simple arithmetic (add, subtract, multiply, divide)

Do conditional operations (if then)

Basically a basic calculator



Control Unit can:

Read, write, and copy values of memory units

Do simple arithmetic (add, subtract, multiply, divide)

Do conditional operations (if then)

Give RAM implementation for:

If x AND y == TRUE output 1