CprE 381: Computer Organization and Assembly Level Programming

Henry Duwe
Electrical and Computer Engineering
Iowa State University

Administrative

- Lab 1 is this week!
- Textbook
  - Great reference
  - Support lecture
    - Won't include stuff not touched on in lecture
  - Homework problems
- HW0 posted (due next Wed Jan 23)
- HW policies:
  - Published ~1 week prior to due date
  - Due on Canvas as **typeset** pdf
  - Due just before Midnight (**11:59pm**) on due date
    - HW solutions automatically released after due date → no late assignments accepted
  - 12 HWs: HW0 + highest 6 of HW1-10 + HW11 counted
- OH: 321 Durham (3pm Wed, 1:30pm Friday) starting **today**

Review

- Processors everywhere
- Processors generally look the same at high-level
- Computer Architecture != Magic

- Relevant to all, even programmers

Concept Map In-Class Assessment

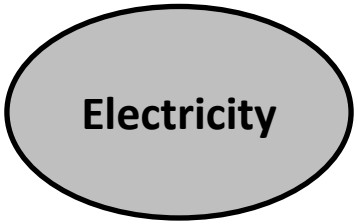
- Help you connect **new information** to your existing knowledge
- Deepen **understanding** and **comprehension**
- Important for this class:
  - Organize the relatively broad topics covered in lecture and homework
  - Help me understand how you are conceptualizing the content of the course
  - Will continue to use during exam reviews

Concept Map In-Class Assessment

1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arrows between related categories (and main concept)
   - Label the connections as well

Concept Map In-Class Assessment

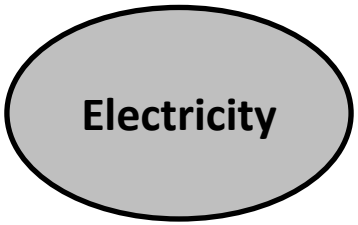
1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arrows between related categories (and main concept)
   - Label the connections as well



Concept Map In-Class Assessment

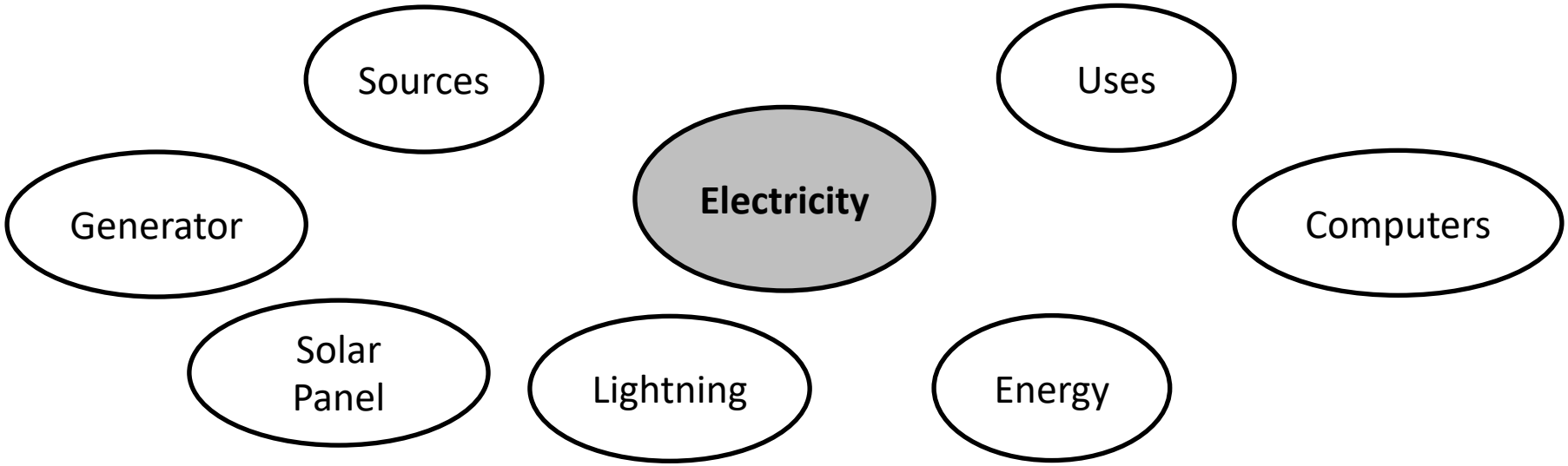
1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arows between related categories (and main concept)
   - Label the connections as well

- Energy
- Lightning
- Generator
- Solar panel
- Power
- Computers



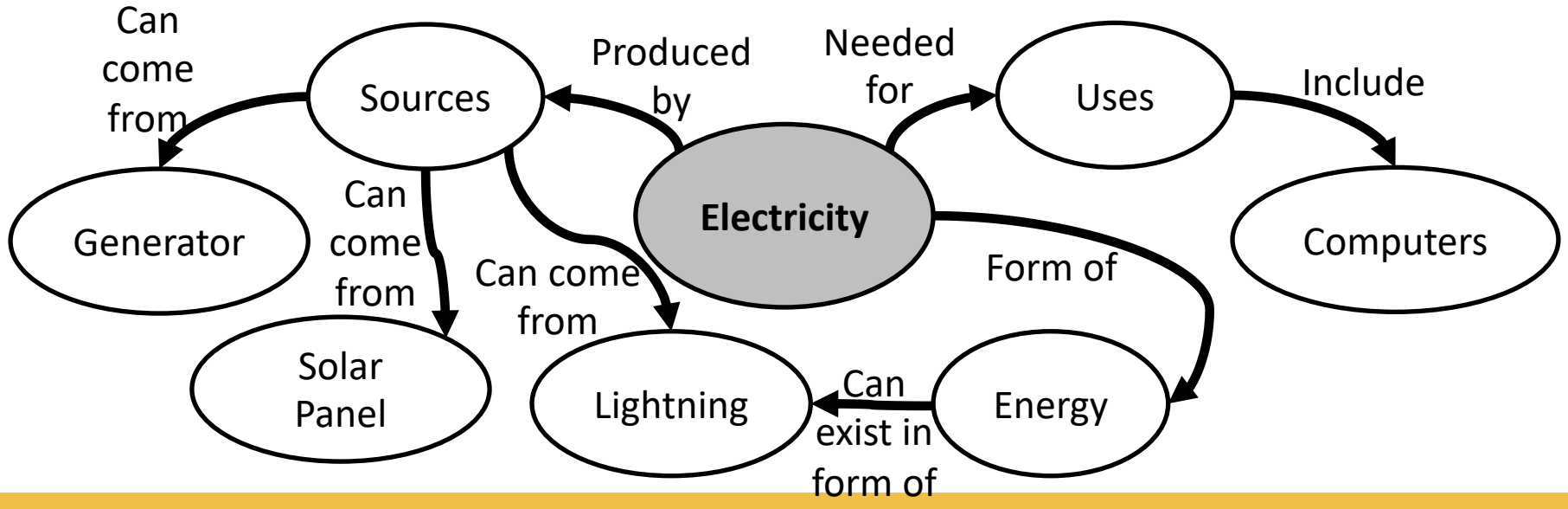
Concept Map In-Class Assessment

1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arrows between related categories (and main concept)
   - Label the connections as well



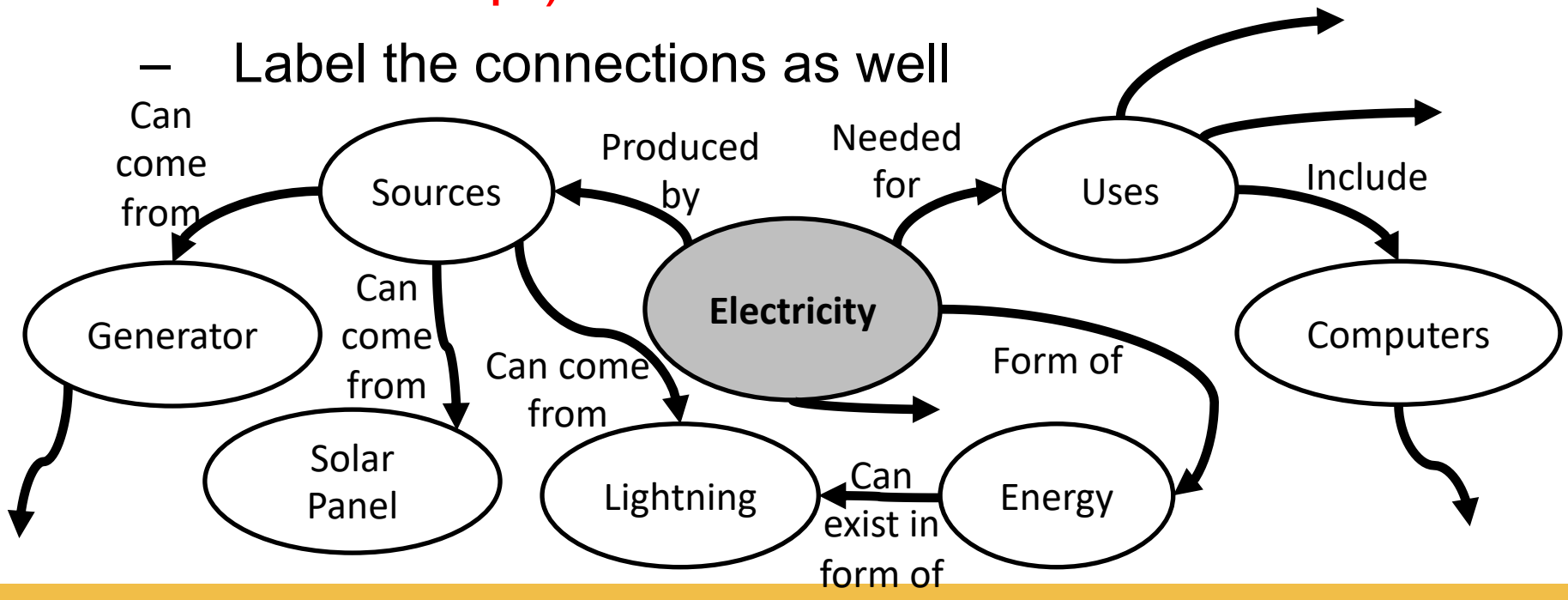
Concept Map In-Class Assessment

1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arrows between related categories (and main concept)
   - Label the connections as well



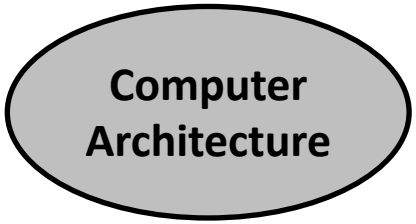
Concept Map In-Class Assessment

1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arrows between related categories (and main concept)
   - Label the connections as well



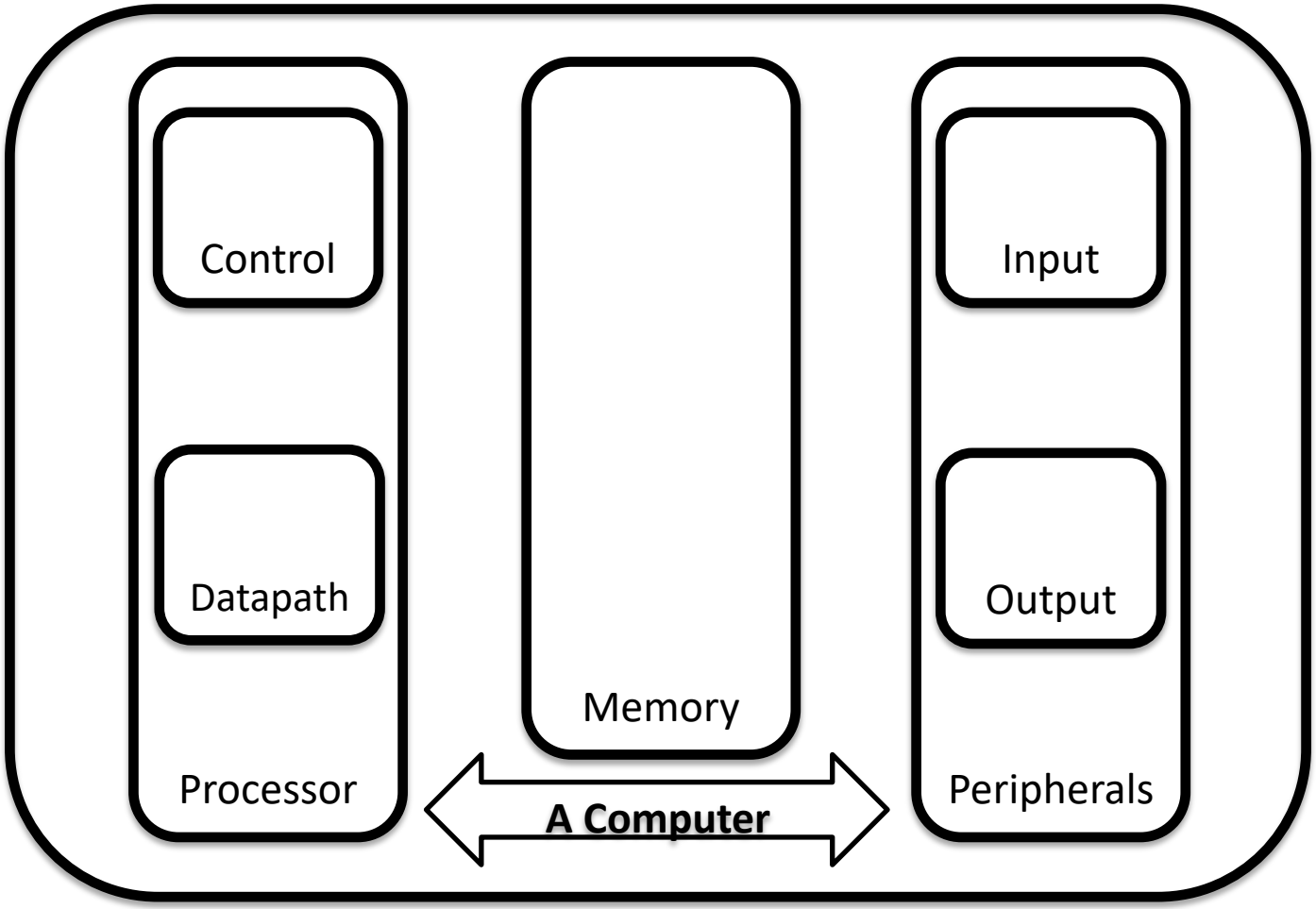
Concept Map In-Class Assessment

1. Brainstorm concepts relating to main concept
2. Organize into categories
3. Draw arrows between related categories (and main concept)
   - Label the connections as well

Your 

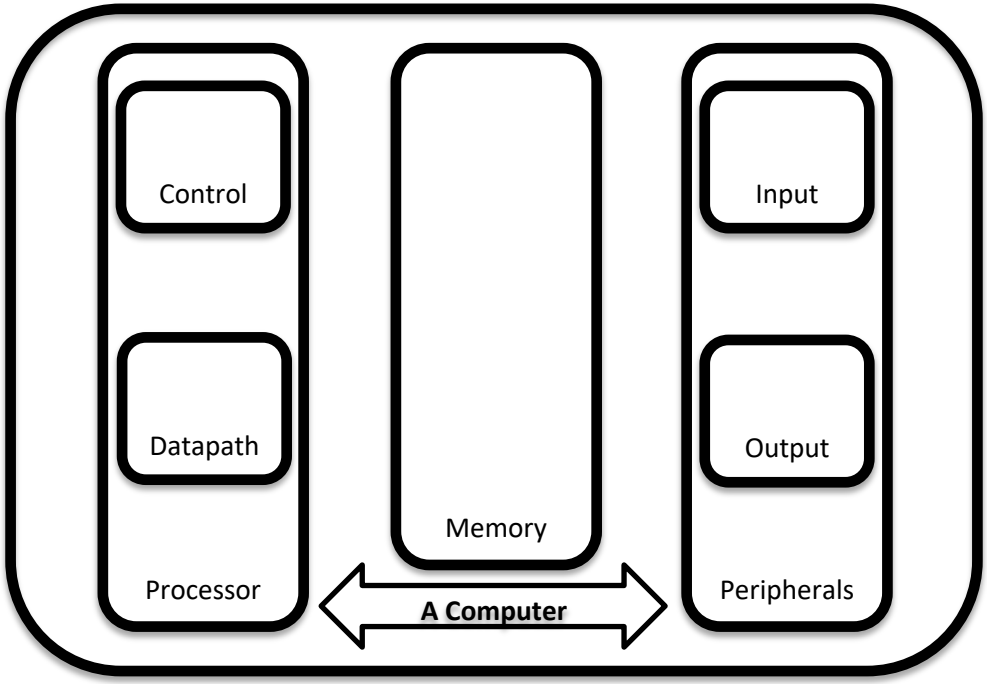
turn

Computers


Control
Datapath
Processor
Memory
Input
Output
Peripherals
A Computer

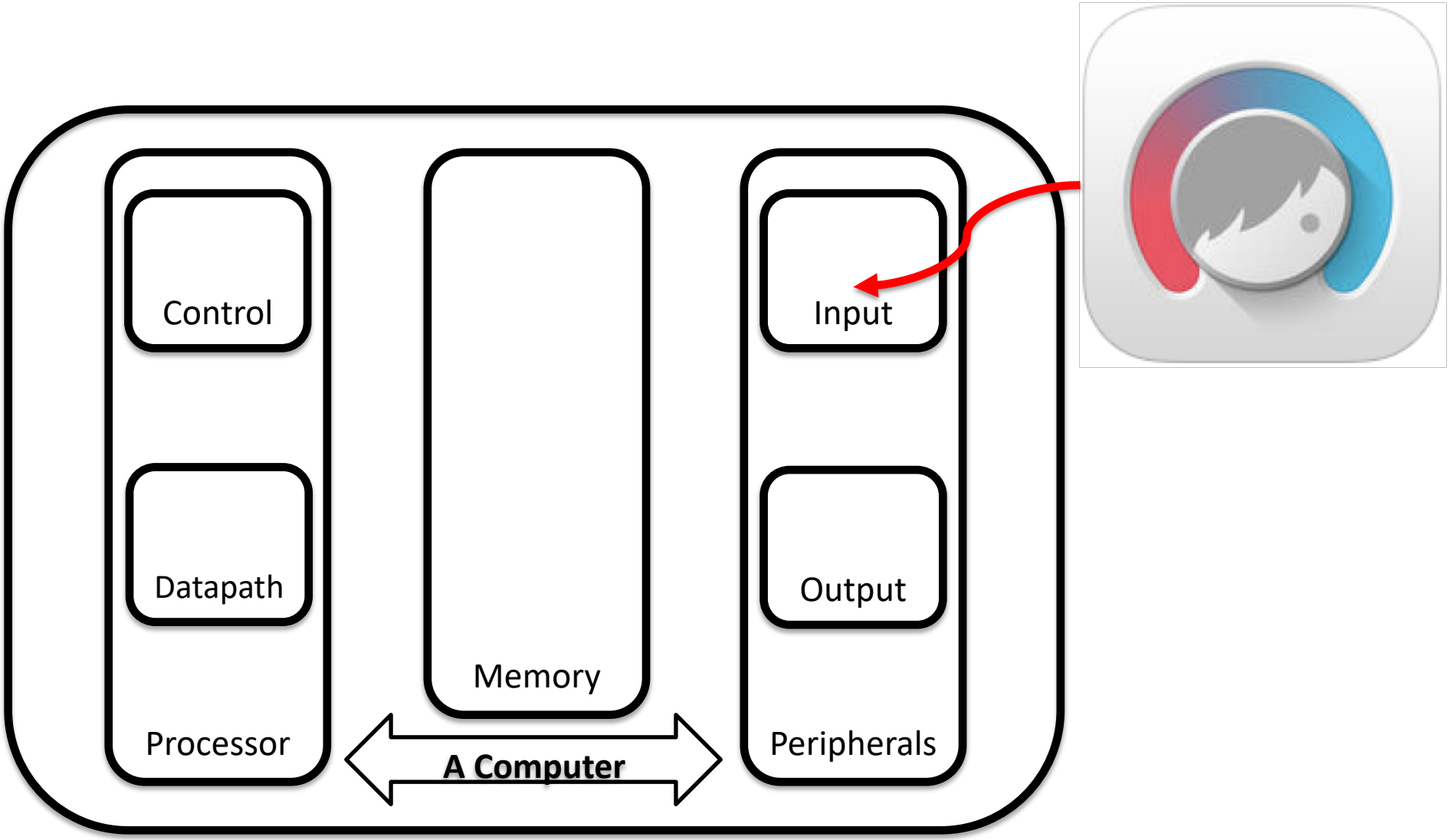
Stored Program Computers

- A la von Neumann Model / Architecture

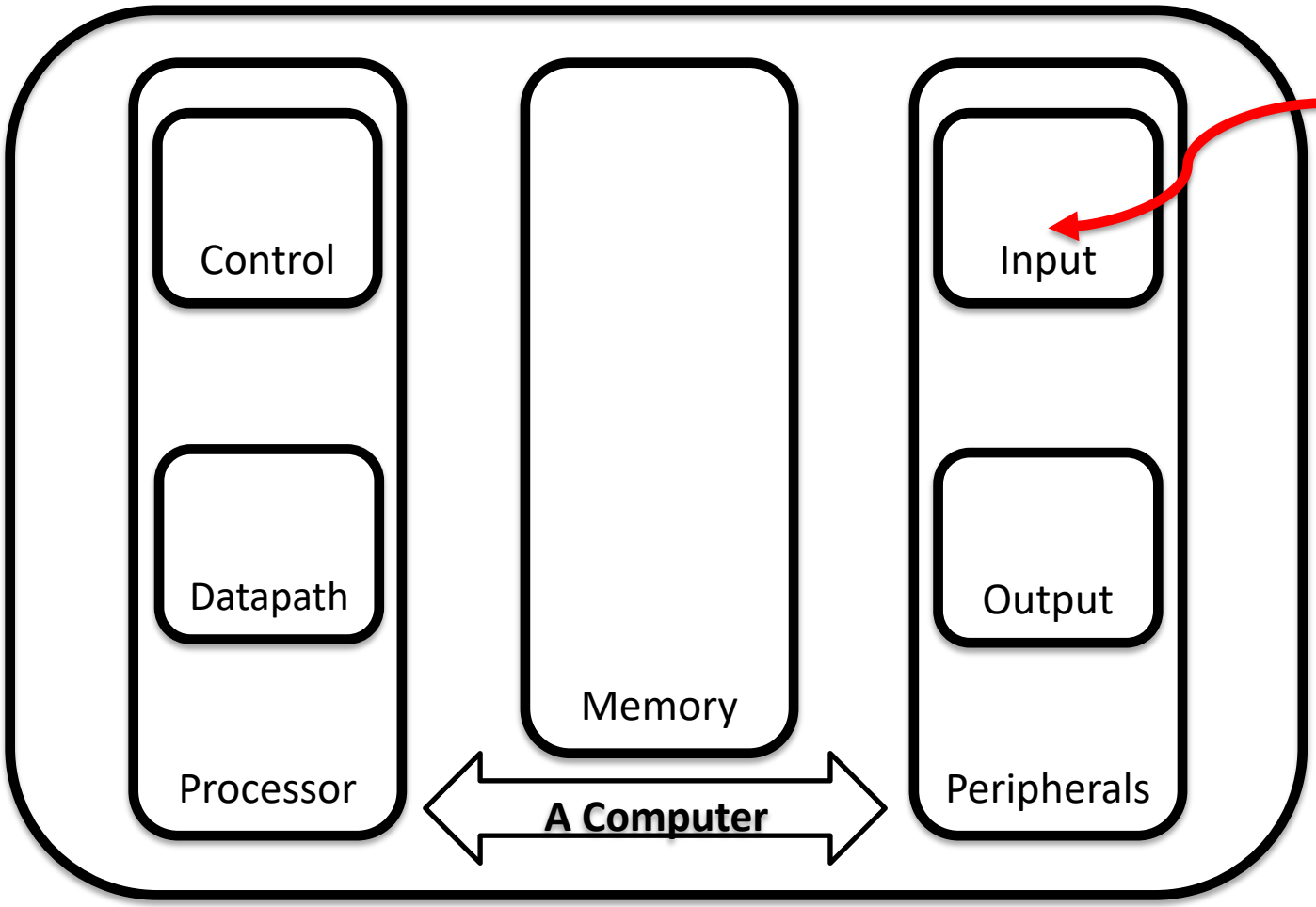


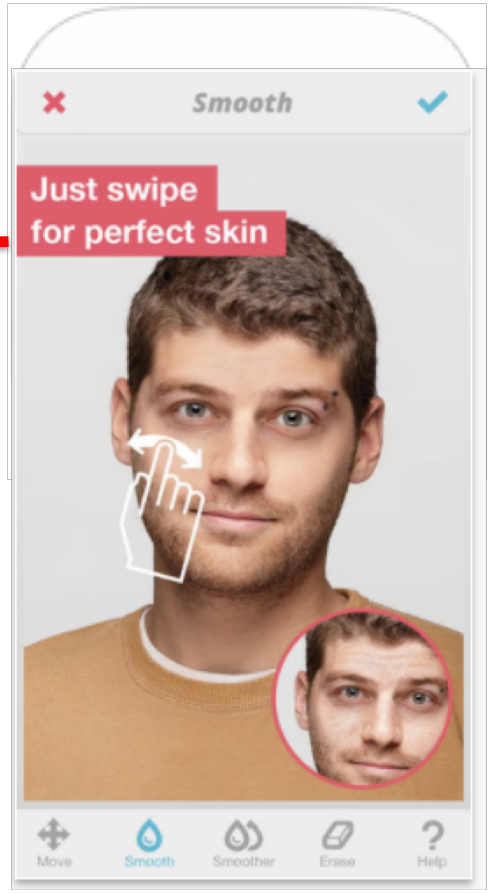
- Two key properties:
  - Instructions stored (as data) in a linear memory array
  - Sequential instruction processing

Input Device Inputs Object Code



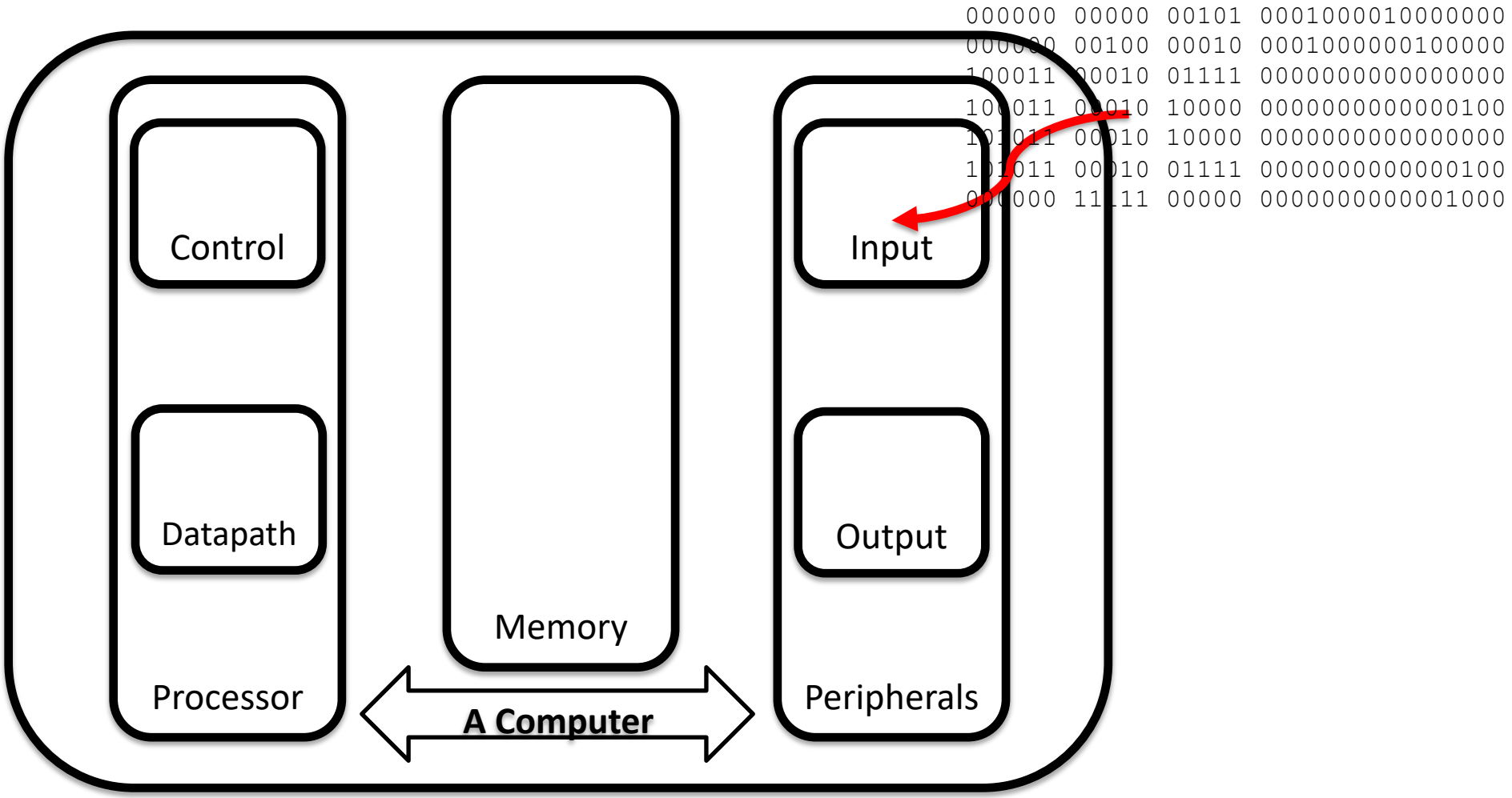
Input Device Inputs Object Code



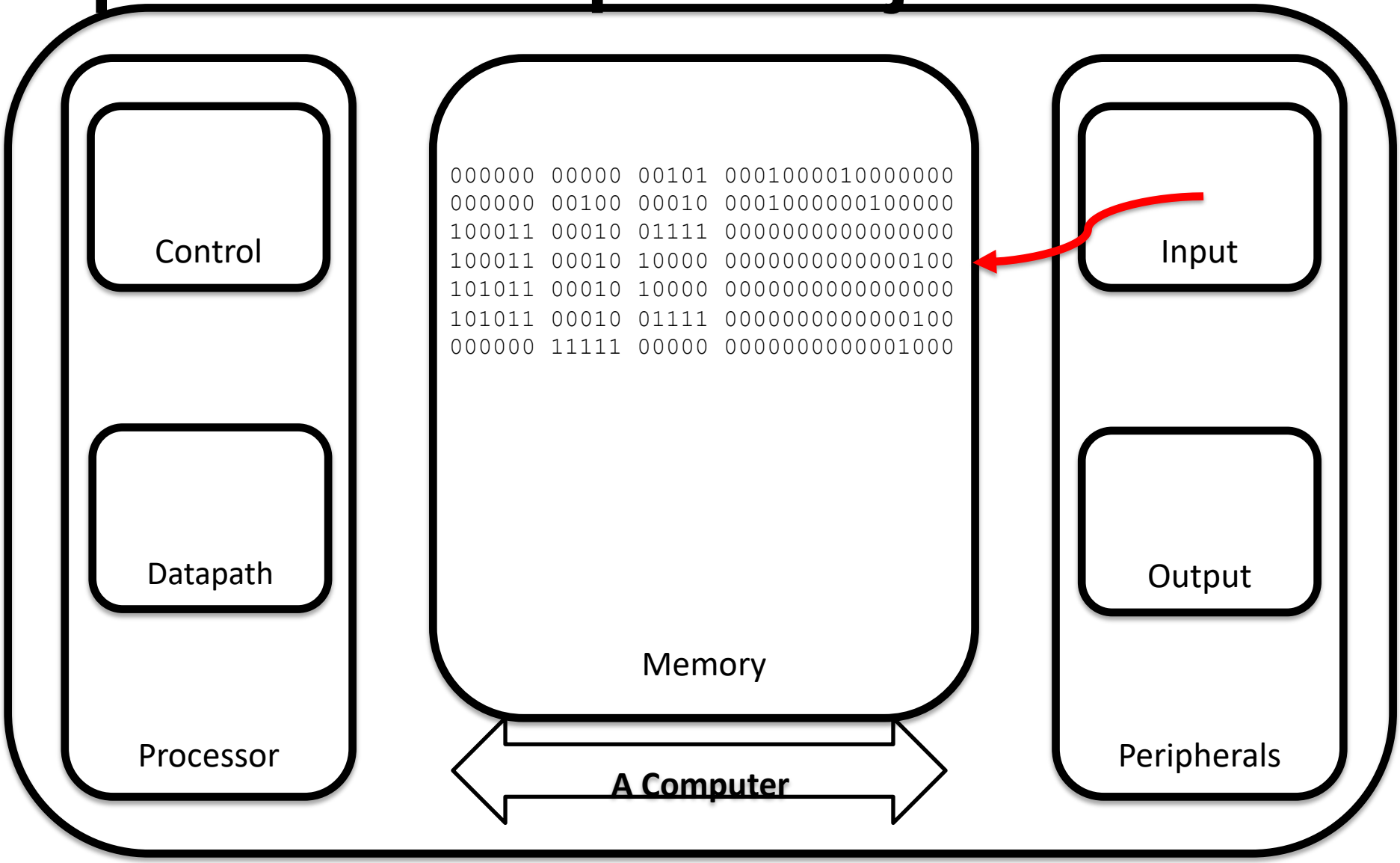


Most downloaded paid app on iOS 2017

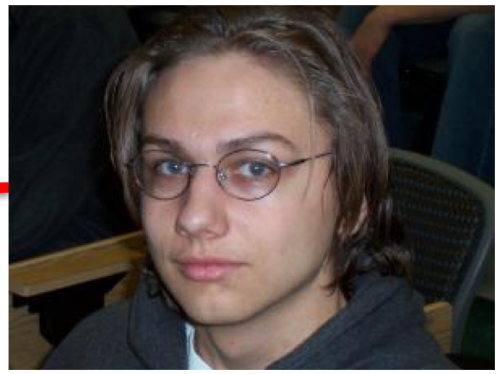
Input Device Inputs Object Code

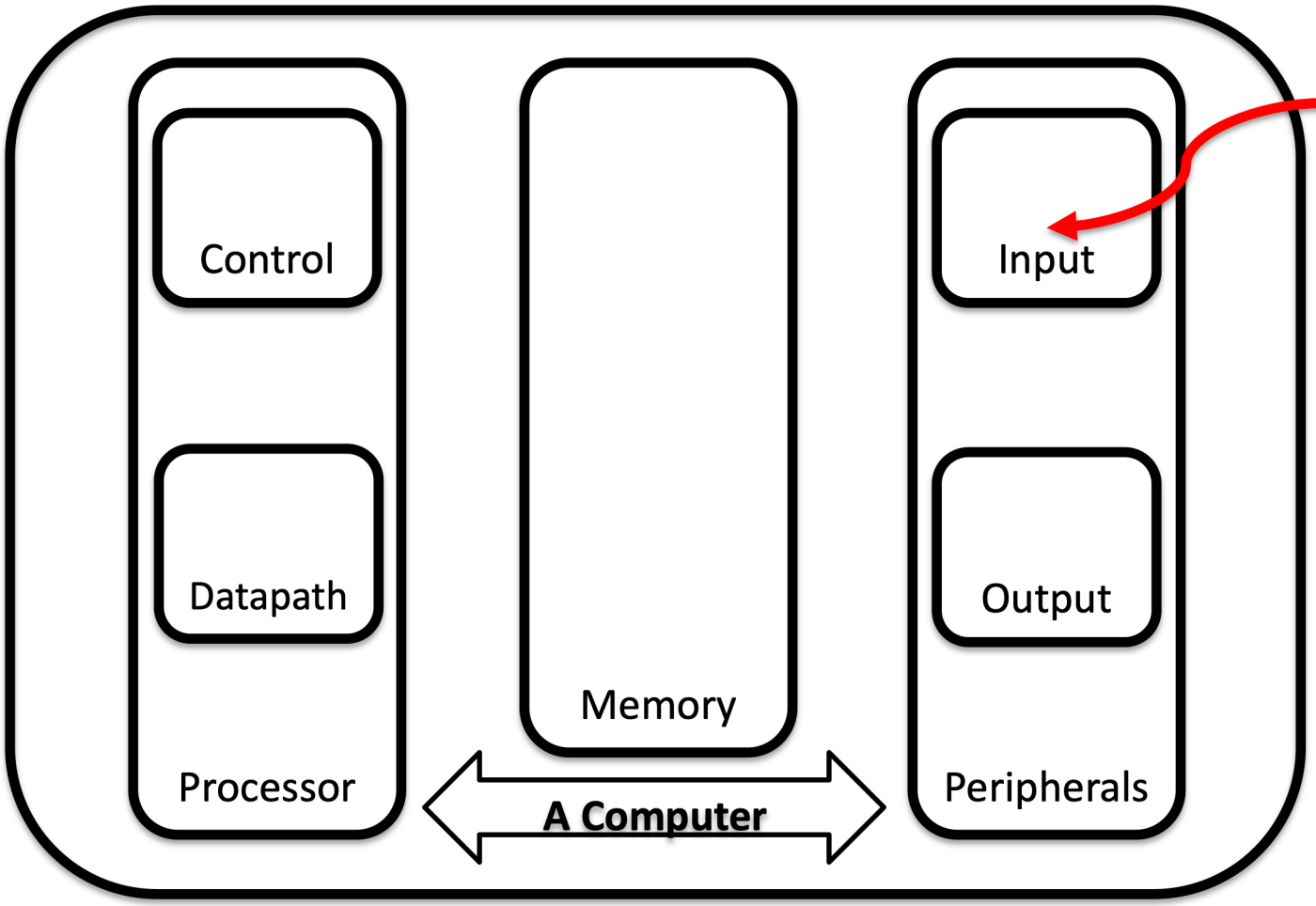


Input Device Inputs Object Code

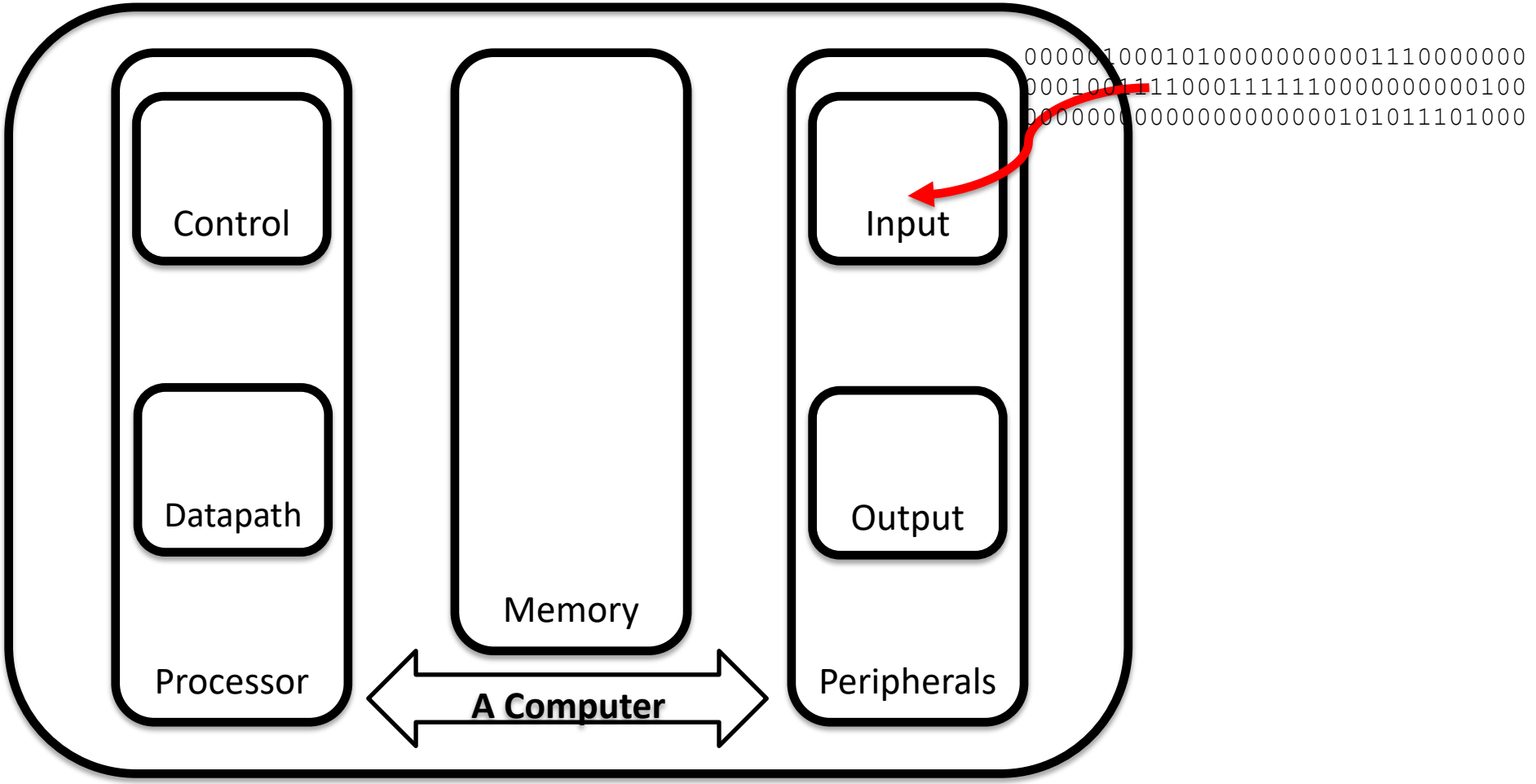


Input Device Inputs Object Code

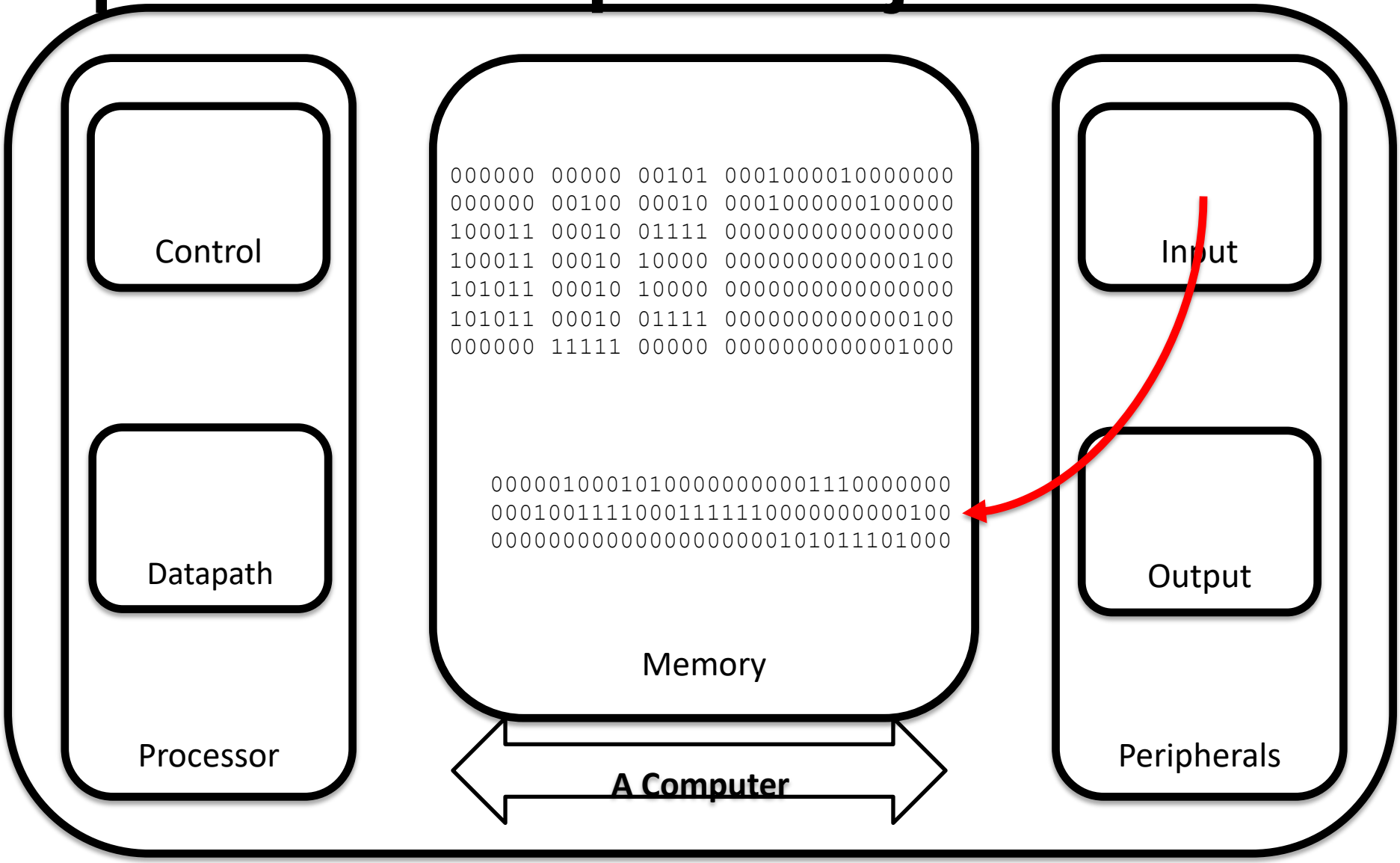




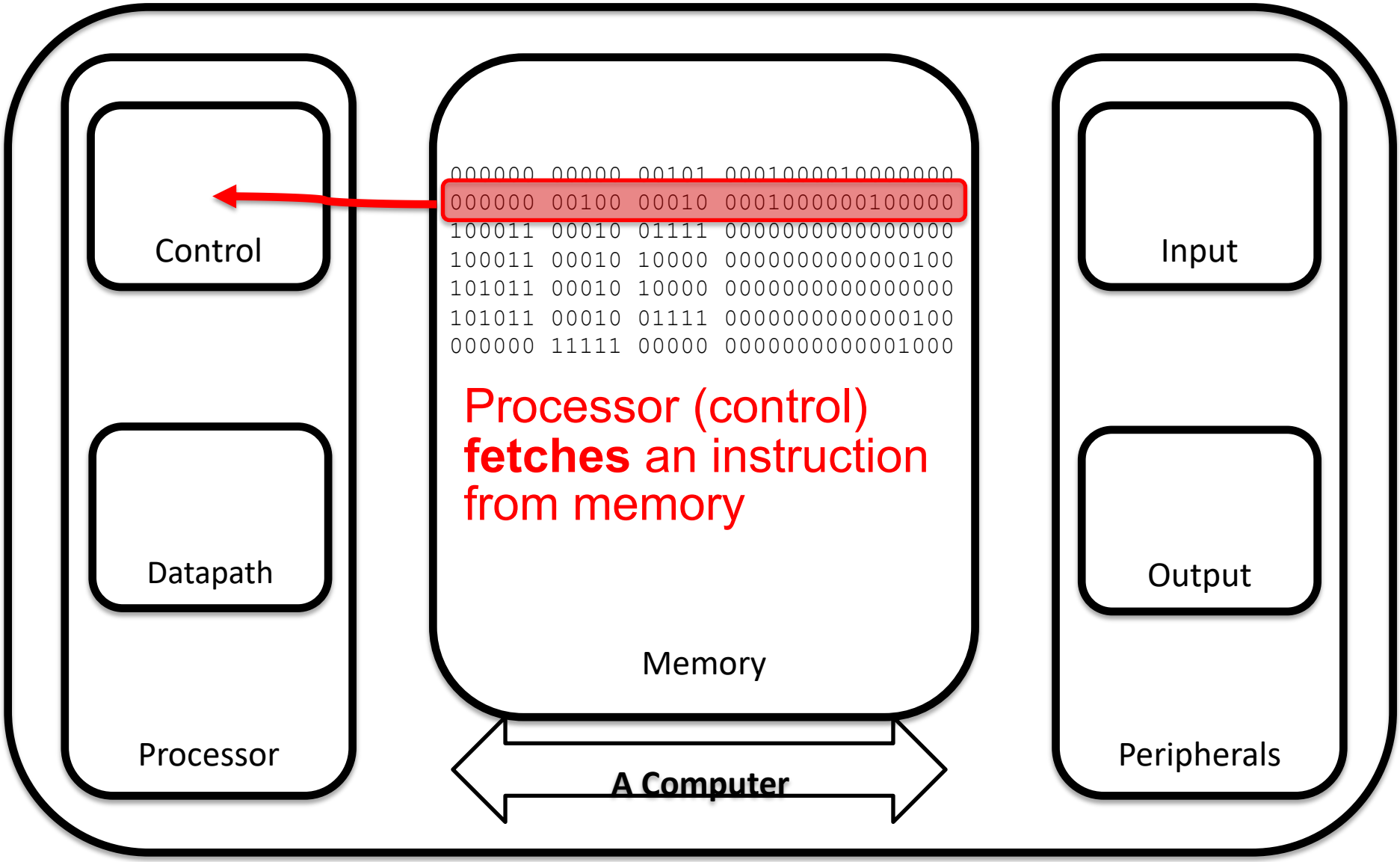
Input Device Inputs Object Code



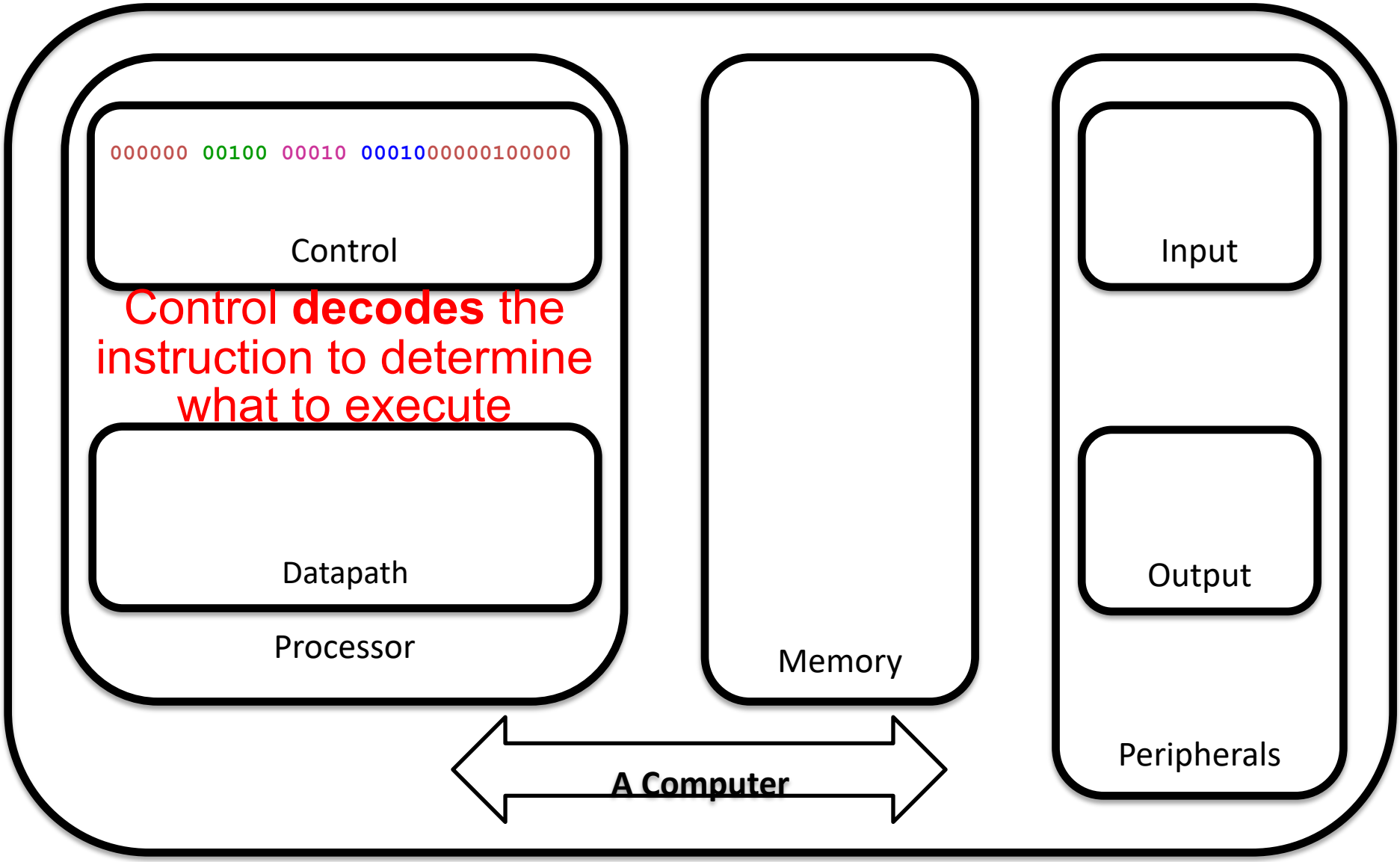
Input Device Inputs Object Code



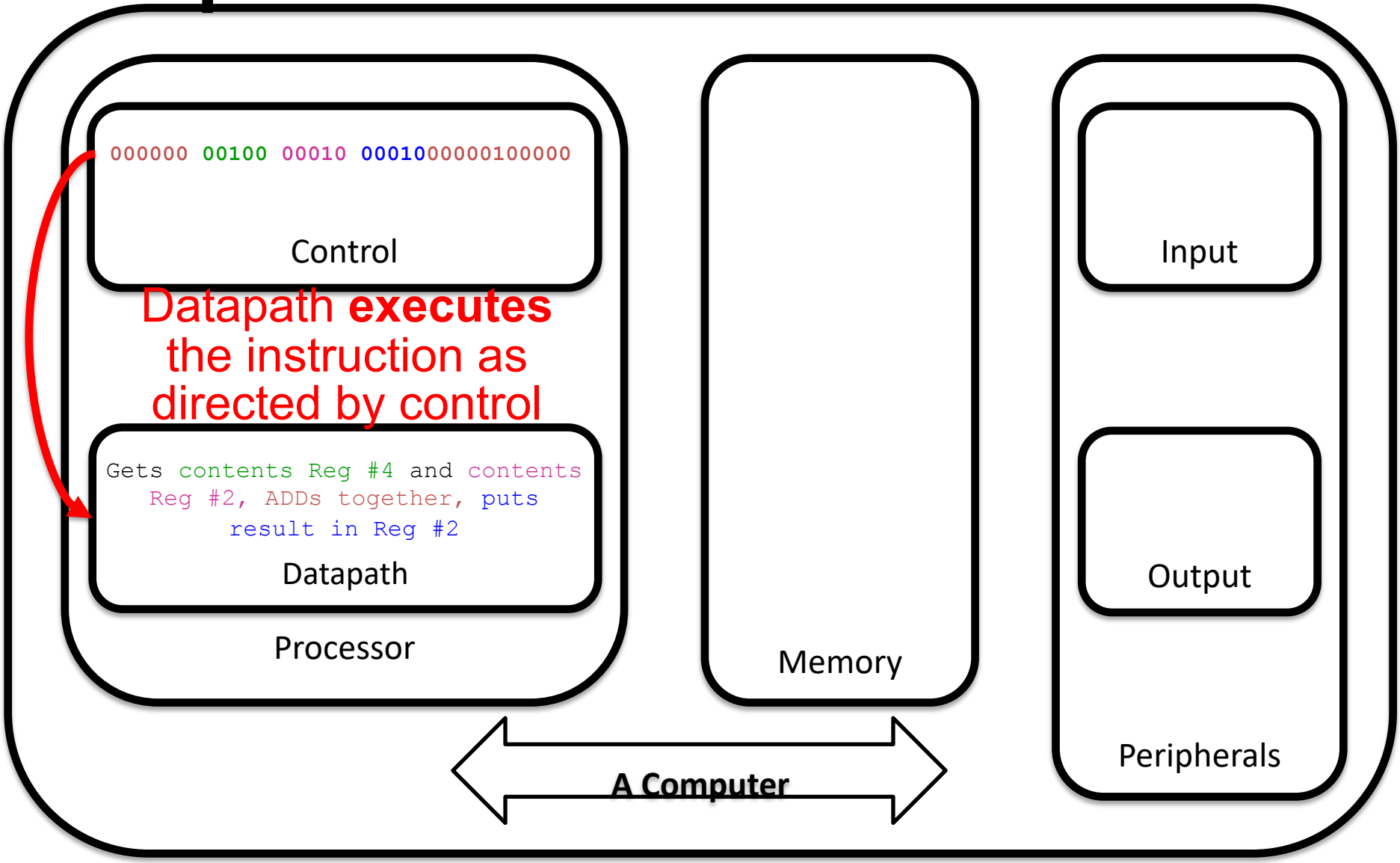
Instruction Fetch



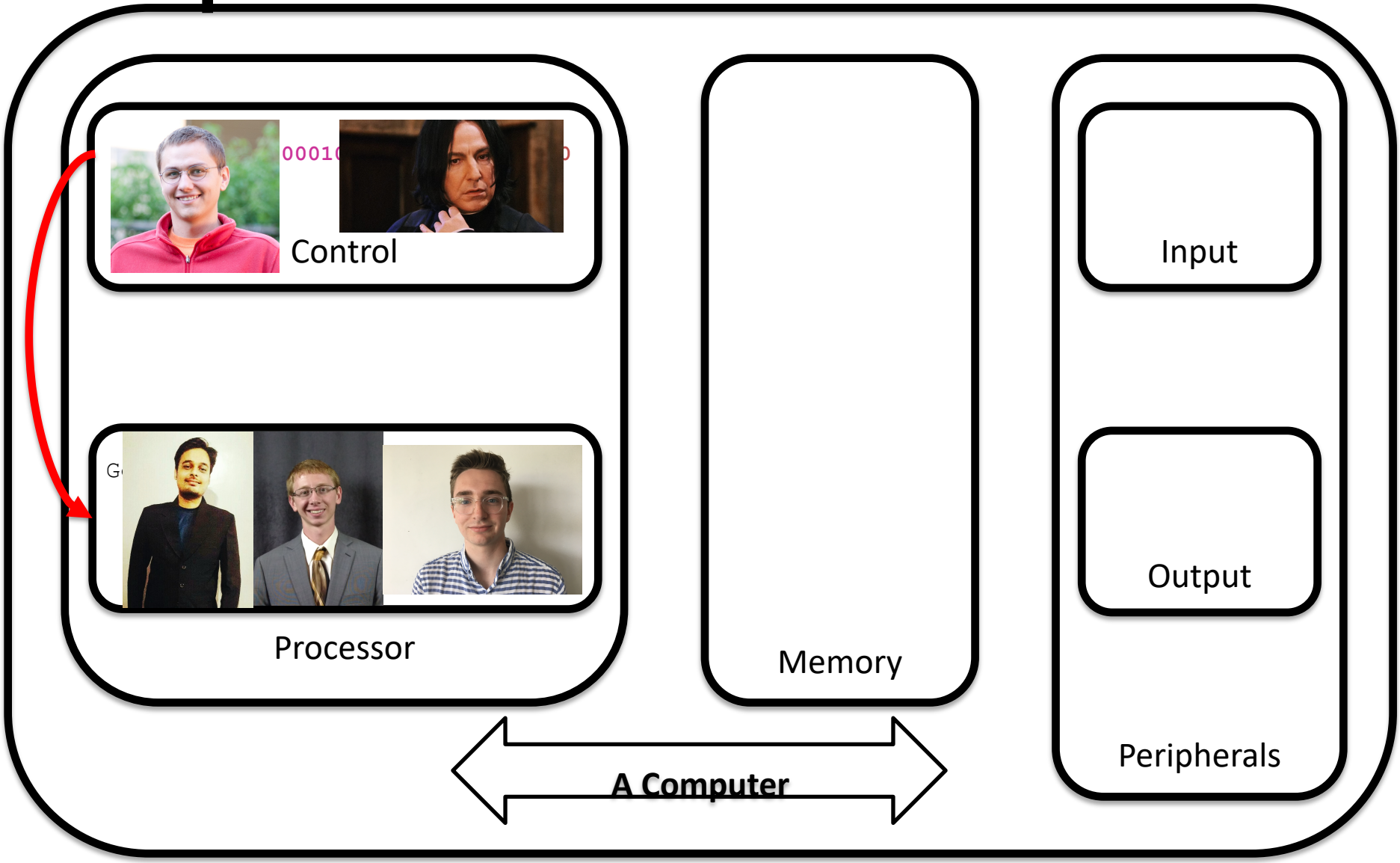
Control Decodes the Instruction



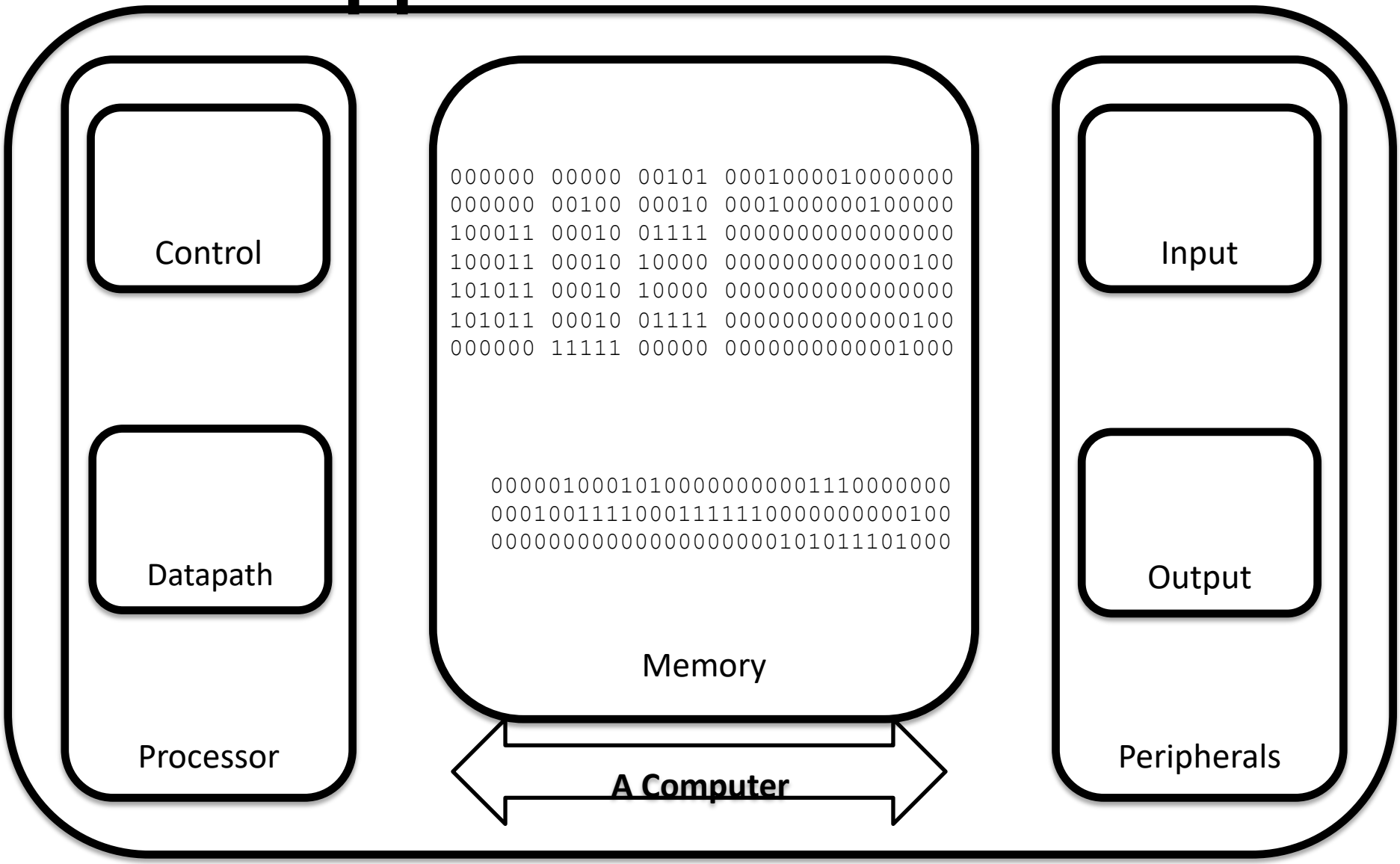
Datapath Executes the Instruction



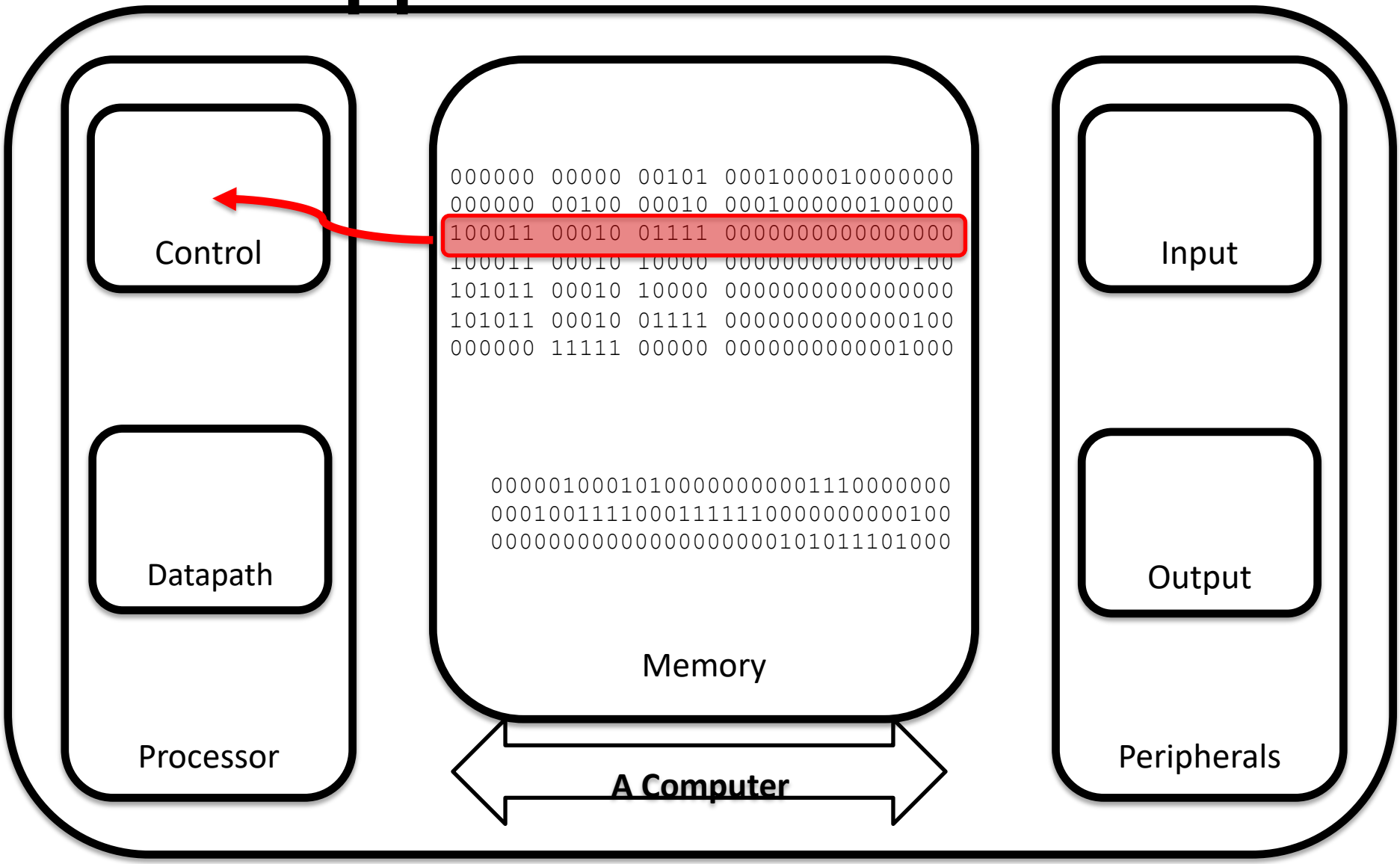
Datapath Executes the Instruction



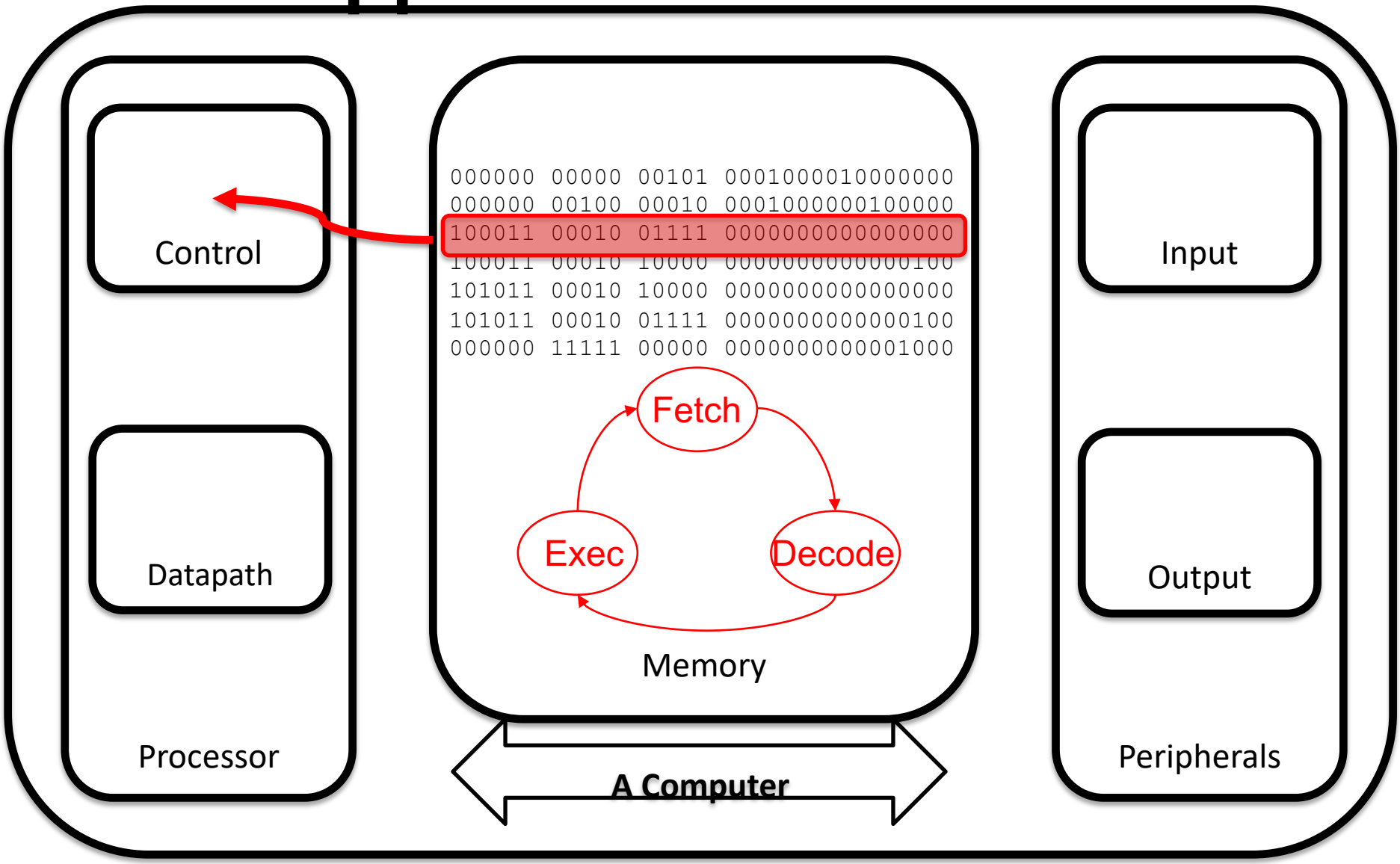
What Happens Next?



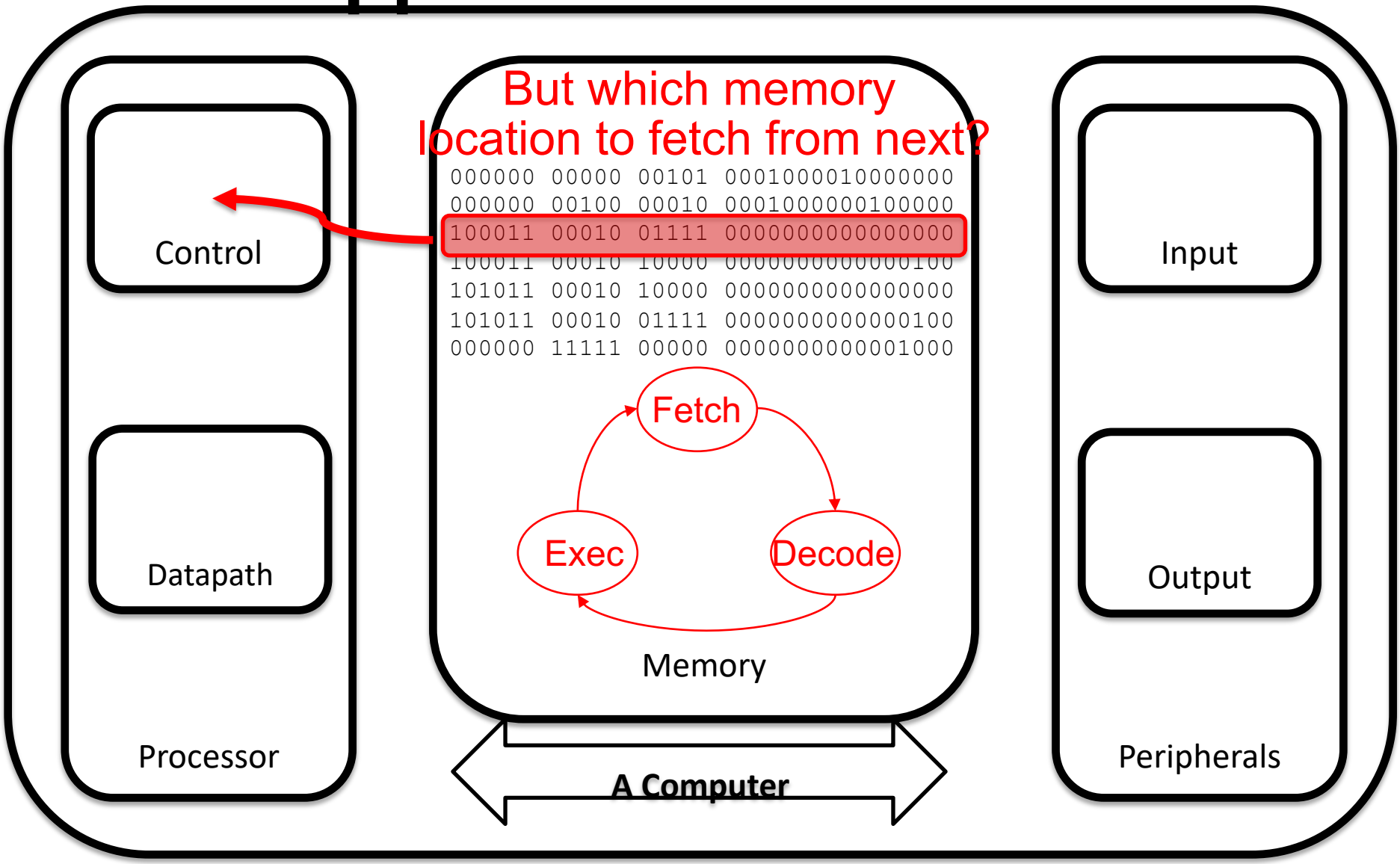
What Happens Next?



What Happens Next?



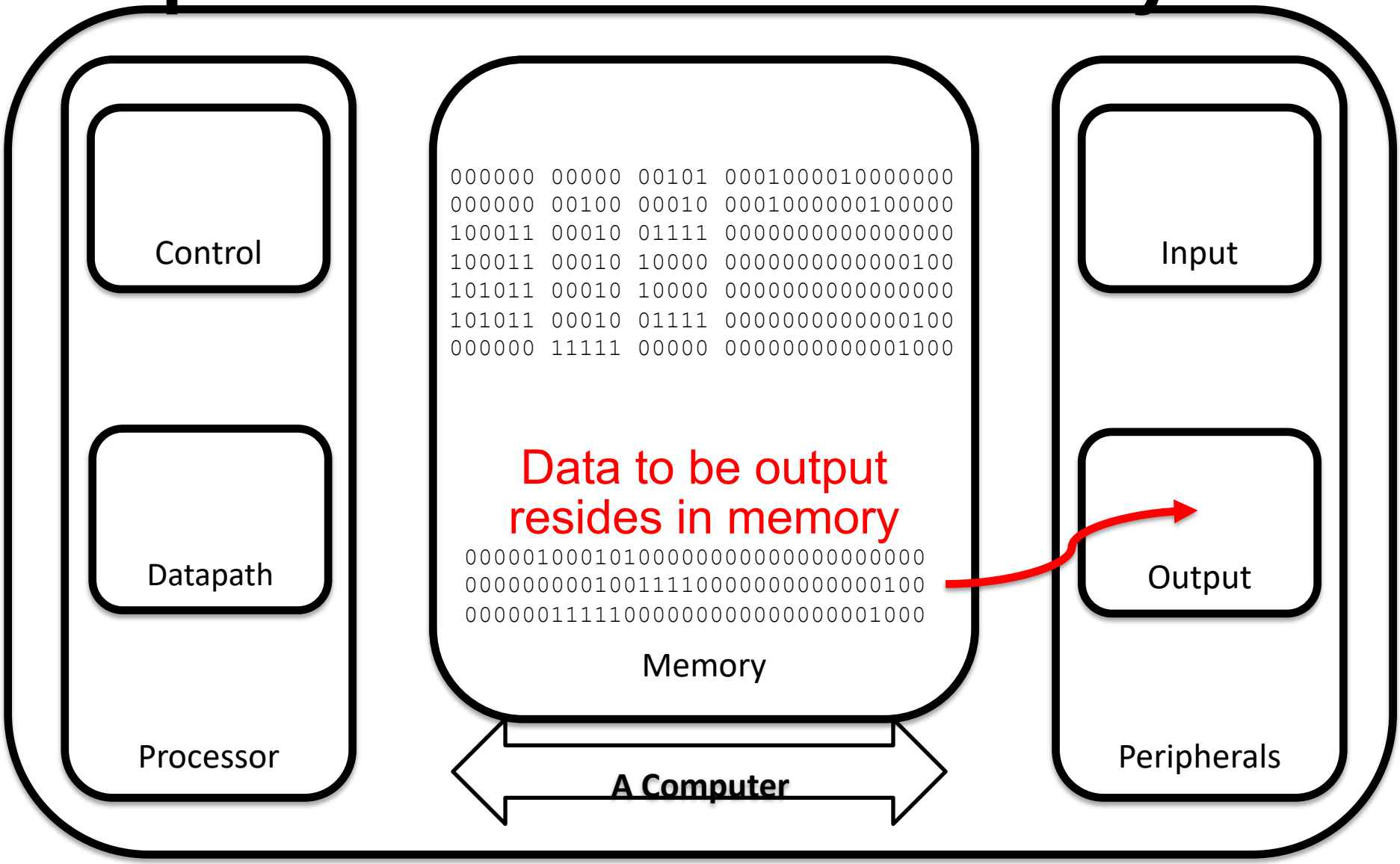
What Happens Next?



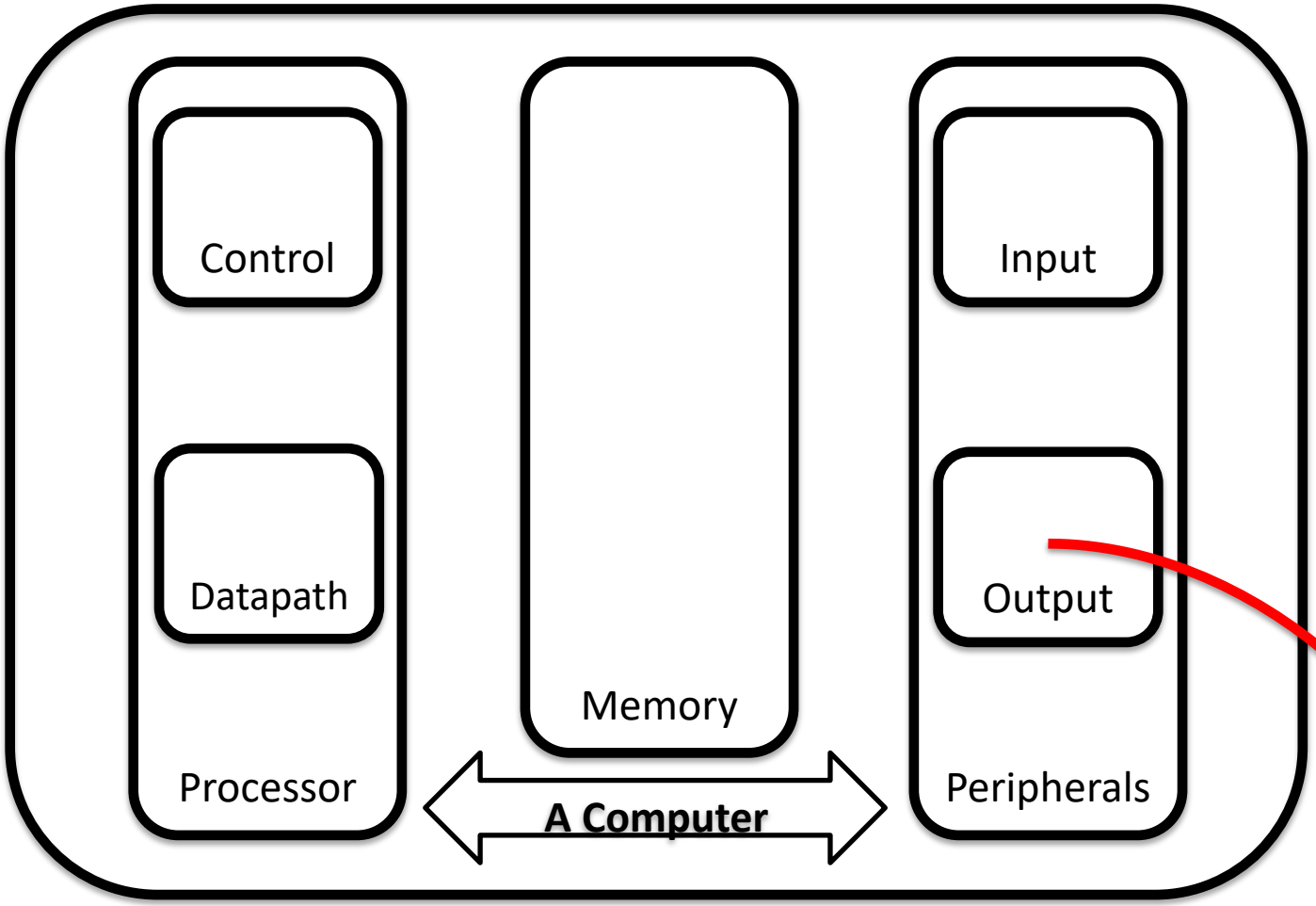
Processor Organization

- Control needs to have circuitry to
  - Decide which is the next instruction and input it from memory
  - Decode the instruction
  - Issue signals that control the way information flows between datapath components
  - Control what operations the datapath's functional units perform
- Datapath needs to have circuitry to
  - Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
  - Interconnect the functional units so that the instructions can be executed as required
  - Load data from and store data to memory

Output Data Stored in Memory

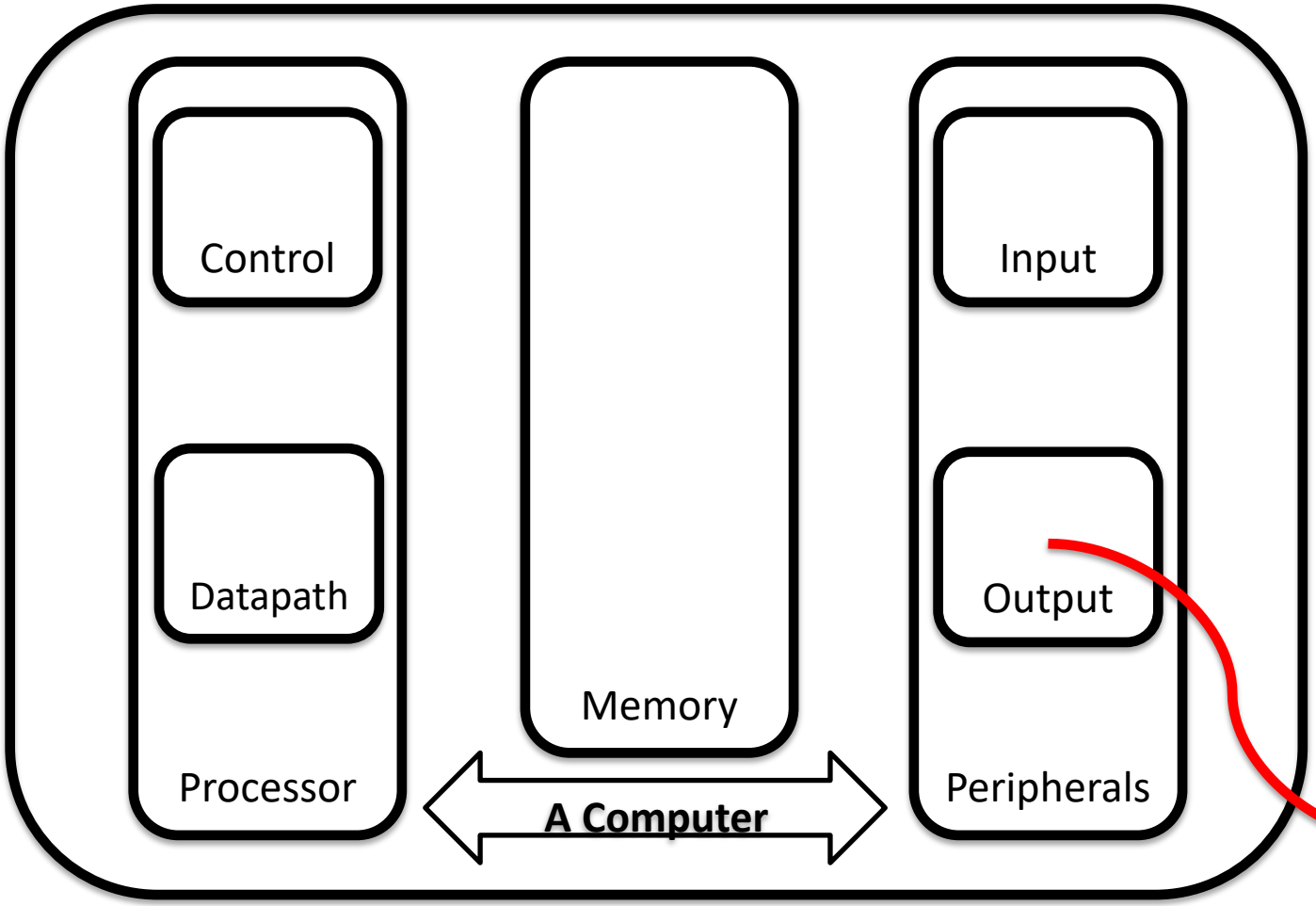


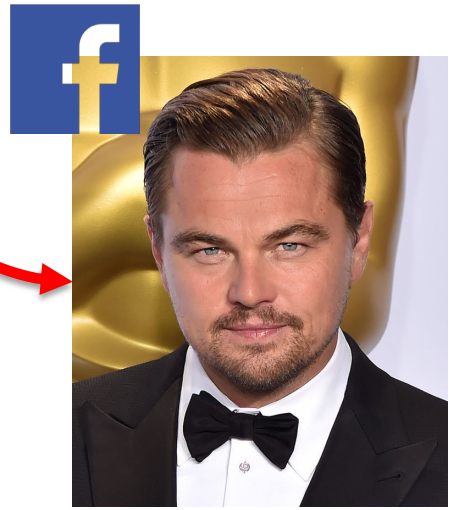
Output Device Outputs Data




```
00000100010100000000000000000000
00000000010011110000000000000100
00000011111000000000000000001000
```

Output Device Outputs Data





Machine Organization Summary

- Capabilities and performance characteristics of the principal Functional Units (FUs)
  - e.g., register file, ALU, multiplexors, memories, ...
- The ways those FUs are interconnected
  - e.g., buses
- Logic and means by which information flow between FUs is controlled
- **R**egister **T**ransfer **L**evel (**RTL**) machine description
- The machine's **I**nstruction **S**et **A**rchitecture (**ISA**)

Below the Program

- High-level language program (in C)

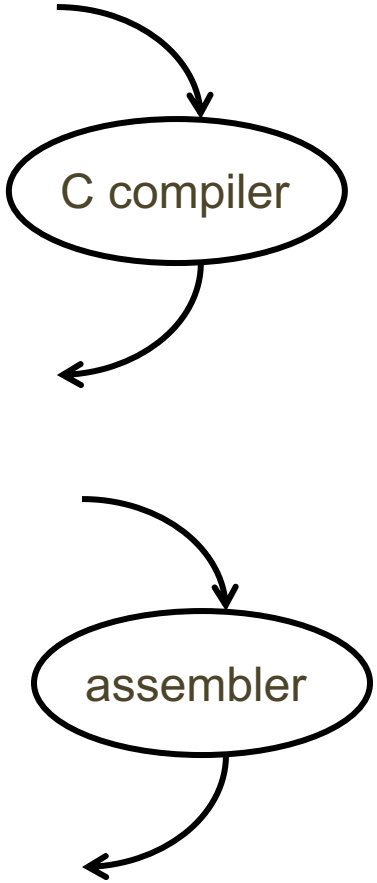
```
swap (int v[], int k)
. . .
```

- Assembly language program (for MIPS)

```
swap:   sll   $2, $5, 2
        add   $2, $4, $2
        lw    $15, 0($2)
        lw    $16, 4($2)
        sw    $16, 0($2)
        sw    $15, 4($2)
        jr    $31
```

- Machine (object) code (for MIPS)

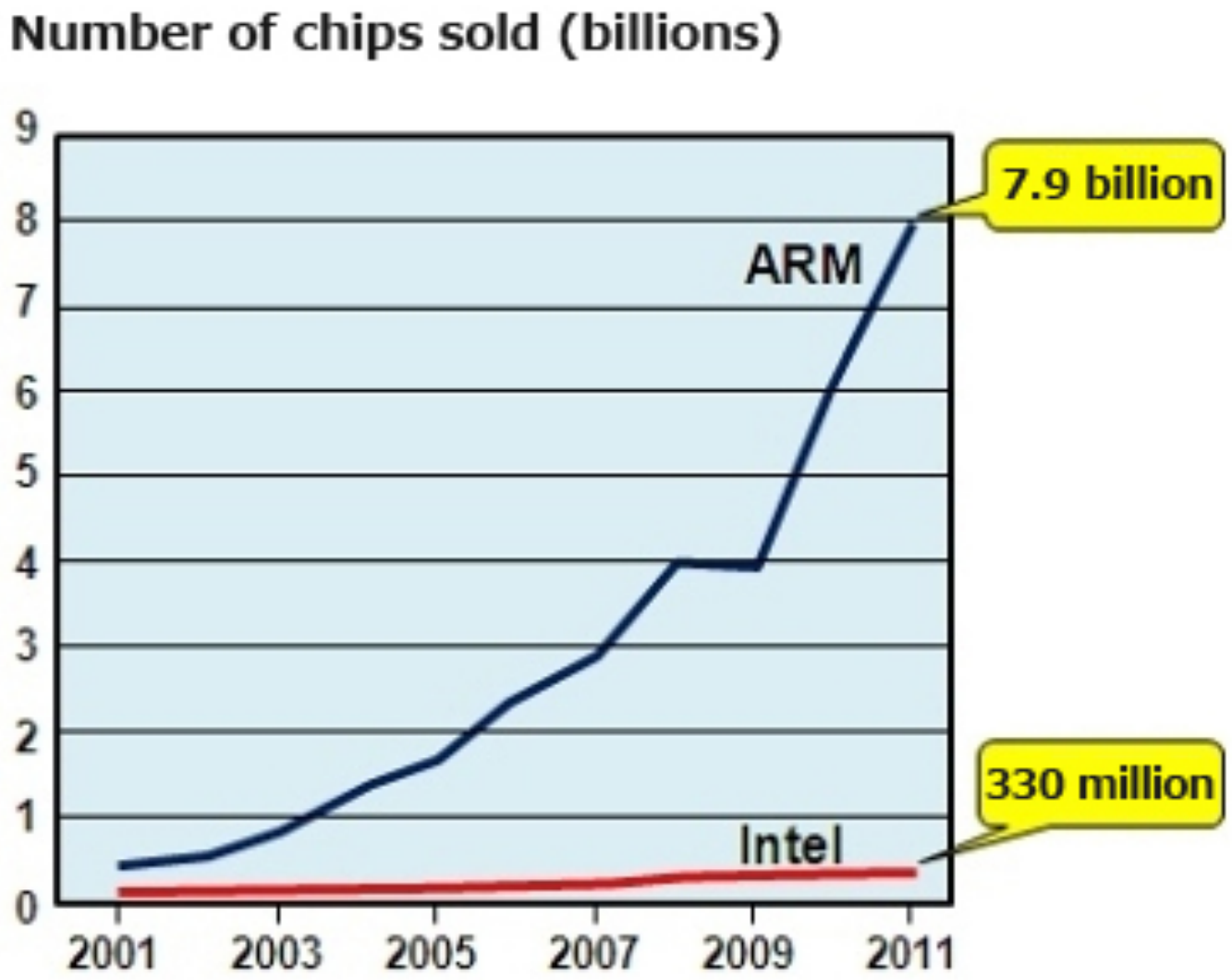
```
000000 00000 00101 00010000010000000
000000 00100 00010 00010000000100000
100011 00010 01111 00000000000000000
100011 00010 10000 00000000000000100
101011 00010 10000 00000000000000000
101011 00010 01111 00000000000000100
000000 11111 00000 00000000000001000
```



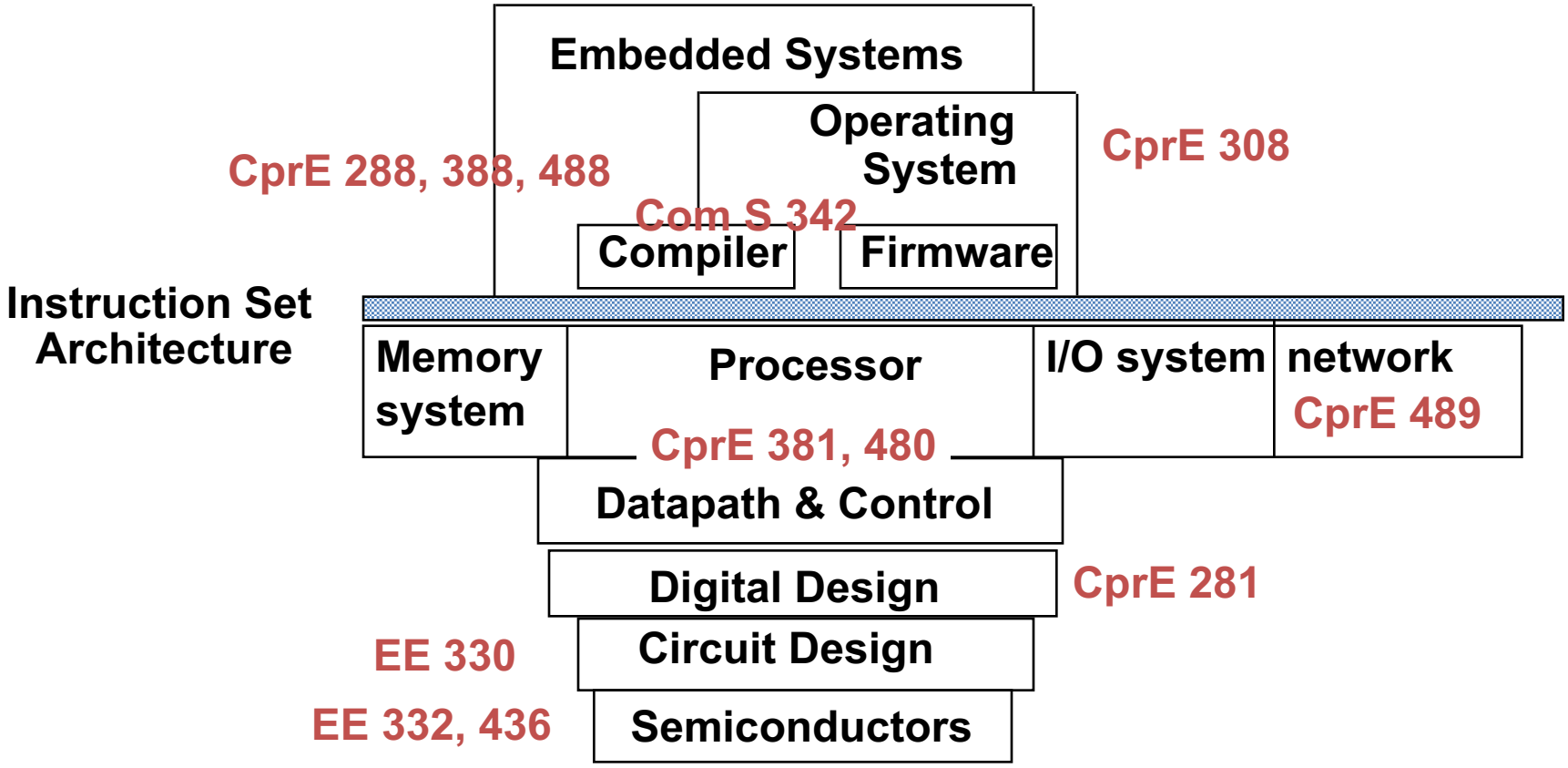
Advantages of Higher-Level Languages?

- Higher-level languages:
  - Allow the programmer to think in a **more natural language** and for their intended use (Fortran for scientific computation, C# for desktop GUI applications, Scheme for symbol manipulation, Javascript for web programming, …)
  - Improve **programmer productivity** – more understandable code that is easier to debug and validate
  - Improve **program maintainability**
  - Allow programs to be **independent of the computer** on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)
  - Emergence of **optimizing compilers** that produce very efficient assembly code optimized for the target machine
- As a result, very little programming is done today at the assembler level

ISA Sales Trends

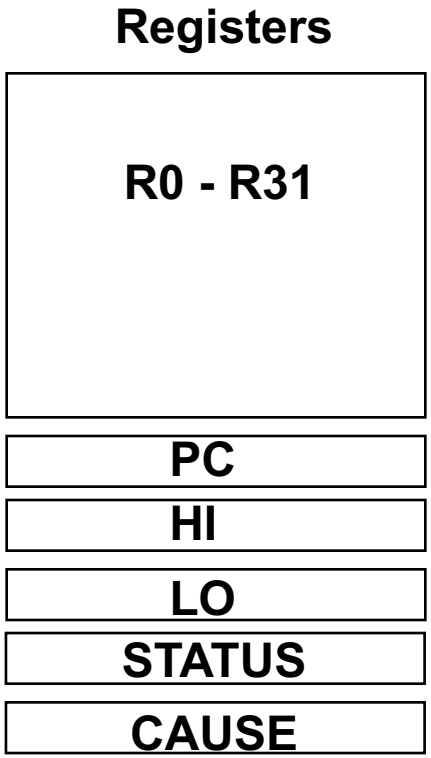


How do the Pieces Fit Together?

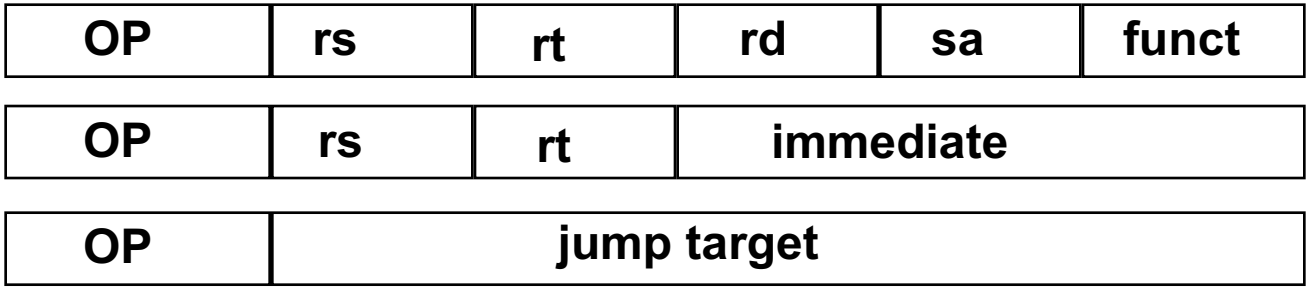


Preview: the MIPS ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



3 Instruction Formats: all 32 bits wide



Acknowledgments

- These slides contain material developed and copyright by:
  - Joe Zambreno (Iowa State)
  - David Patterson (UC Berkeley)
  - Mary Jane Irwin (Penn State)
  - Christos Kozyrakis (Stanford)
  - Onur Mutlu (Carnegie Mellon)
  - Krste Asanović (UC Berkeley)