CENG 3420
Computer Organization & Design
Lecture 02: ISA Introduction

Textbook: Chapter 1.3-1.4, 2.1-2.2

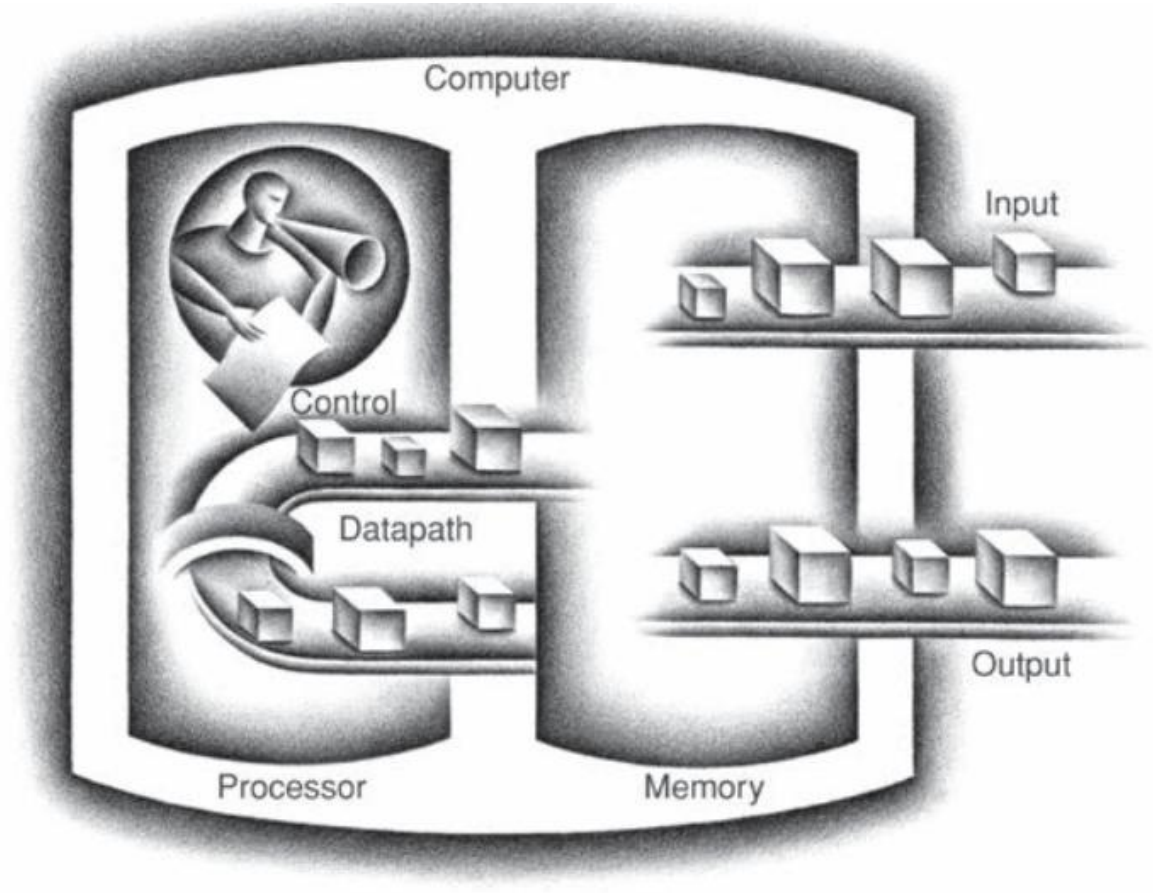
Zhengrong Wang

CSE Department, CUHK

zhengrongwang@cuhk.edu.hk

What is a Computer?

- Five components:
  - Input (keyboard, mouse, etc.).
  - Output (display, printer).
  - Memory (cache, main memory, disk, SSD).
  - Datapath.
  - Control.
- Our primary focus:
  - Datapath and control.
  - Interaction with memory.
  - Implemented with billions of transistors.



Opening the Box – iPhone XS Max

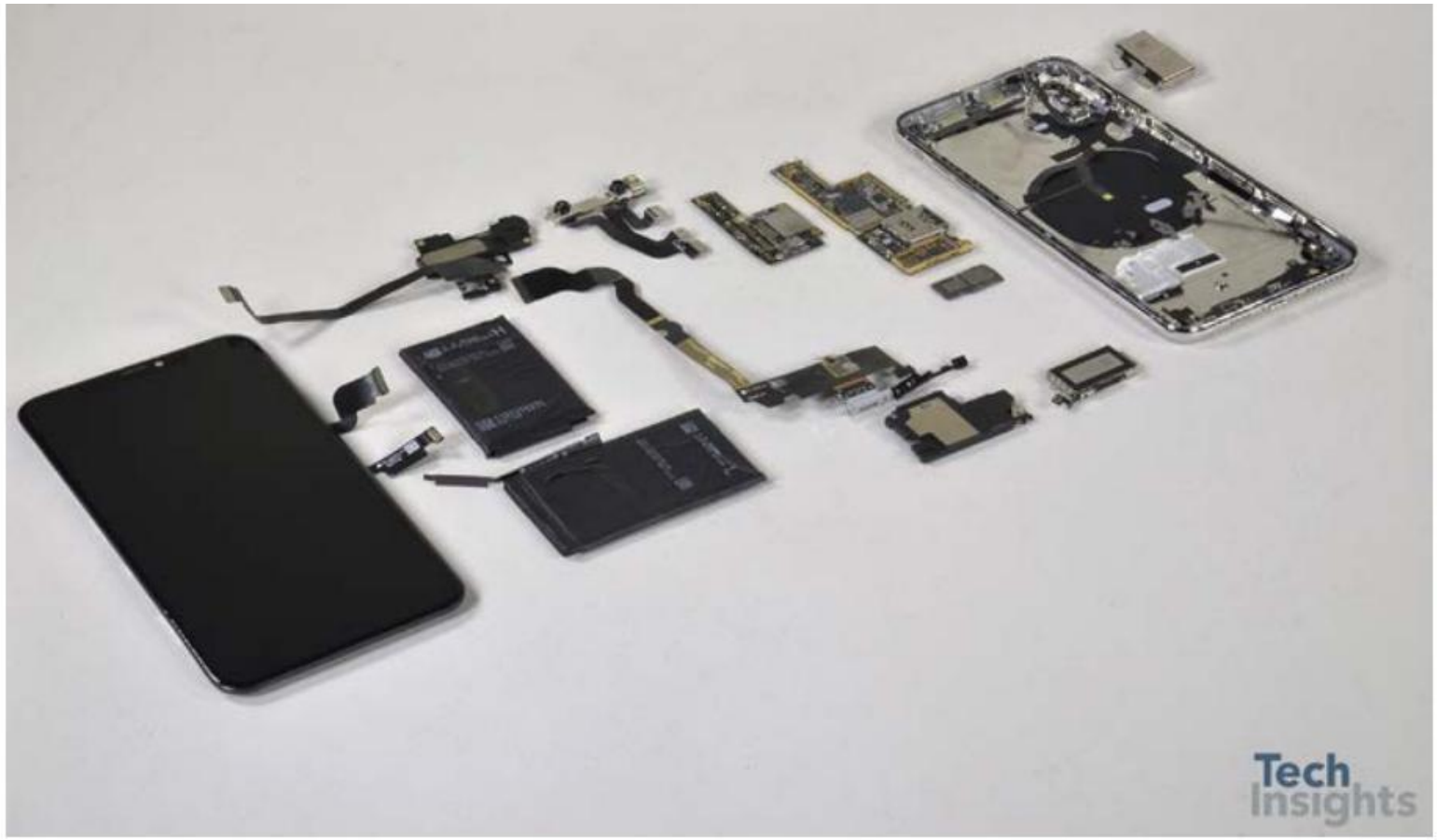


FIGURE 1.7 Components of the Apple iPhone XS Max cell phone. At the left is the capacitive multitouch screen and LCD display. Next to it is the battery. To the far right is the metal frame that attaches the LCD to the back of the iPhone. The small components in the center are what we think of as the computer; they are not simple rectangles to fit compactly inside the case next to the battery. Figure 1.8 shows a close-up of the board to the left of the metal case, which is the logic printed circuit board that contains the processor and memory. (Courtesy TechIngishts, www.techIngishts.com)

Opening the Box – Motherboard

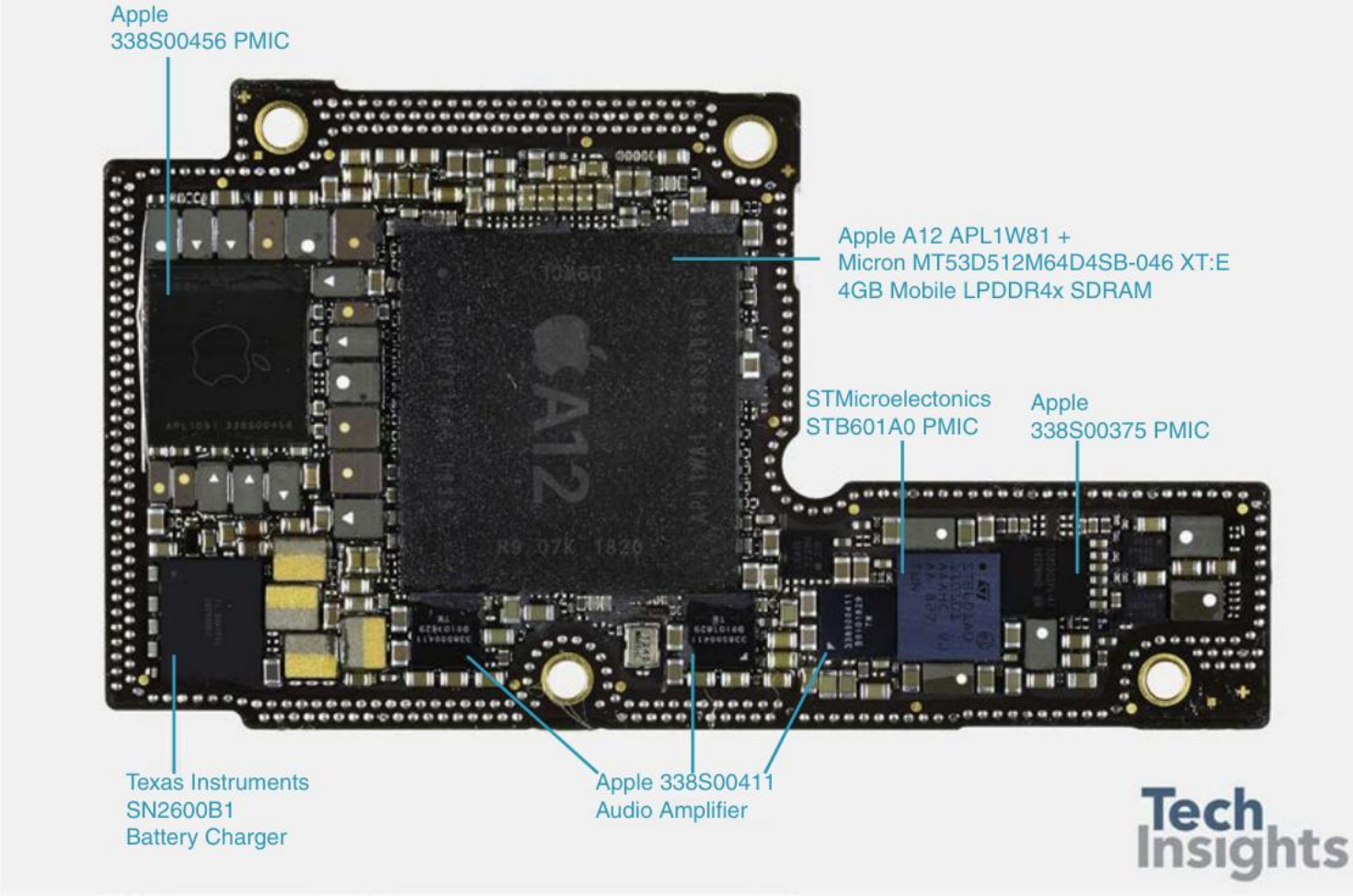


FIGURE 1.8 The logic board of Apple iPhone XS Max in Figure 1.7. The large integrated circuit in the middle is the Apple A12 chip, which contains two large and four small ARM processor cores that run at 2.5 GHz, as well as 2 GiB of main memory inside the package. Figure 1.9 shows a photograph of the processor chip inside the A12 package. A similar-sized chip on a symmetric board that attaches to the back is a 64 GiB flash memory chip for nonvolatile storage. The other chips on the board include the power management integrated controller and audio amplifier chips. (Courtesy TechIngishts, www.techIngishts.com)

Processor Organization

- Control needs to have circuitry to
  - Decide which is the next instruction.
  - Fetch instruction from memory.
  - Decode the instruction.
  - Issue and control it through the datapath.
  - Control what operations the datapath performs.
- Datapath needs to have circuitry to
  - Execute instructions – function units (e.g., adder) and storage locations (e.g., registers).
  - Interconnect components so that instructions can be executed as required.
  - Load and store data from memory.

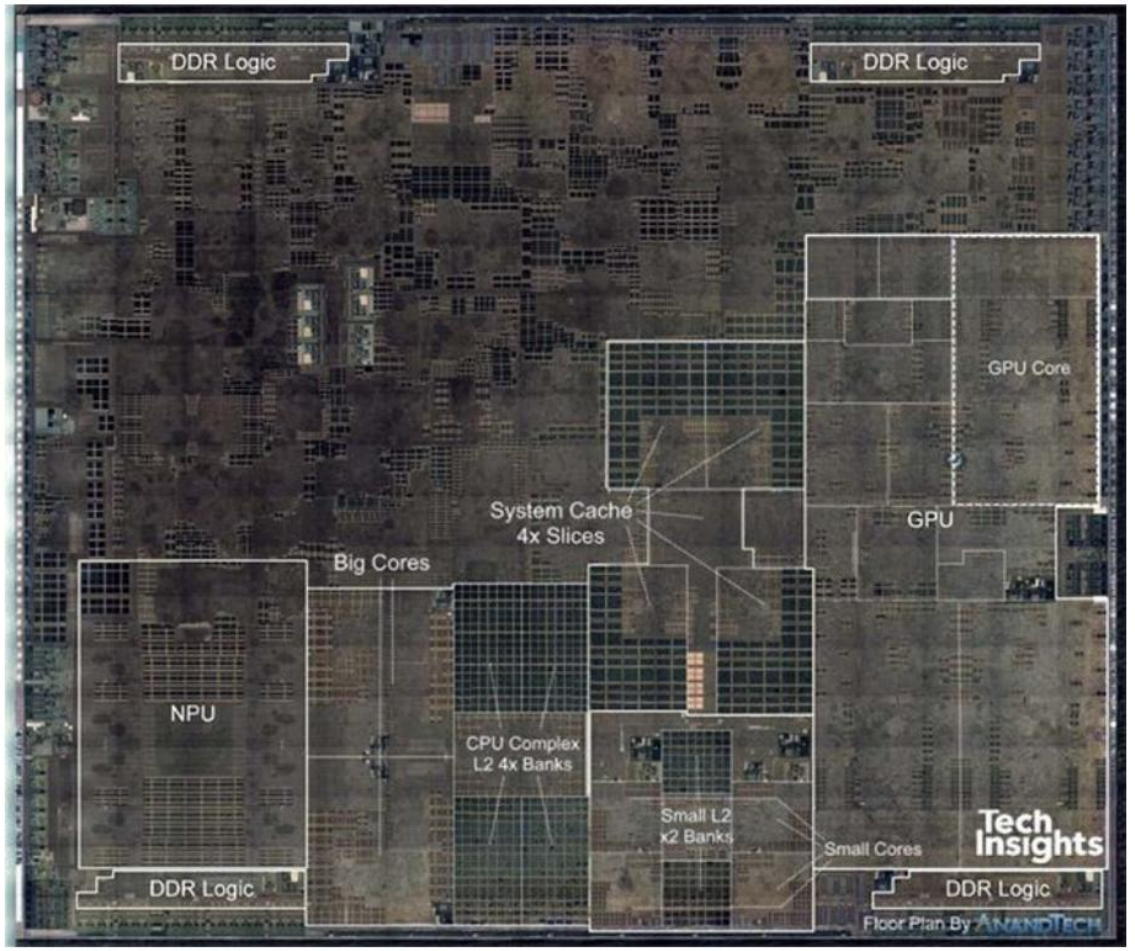
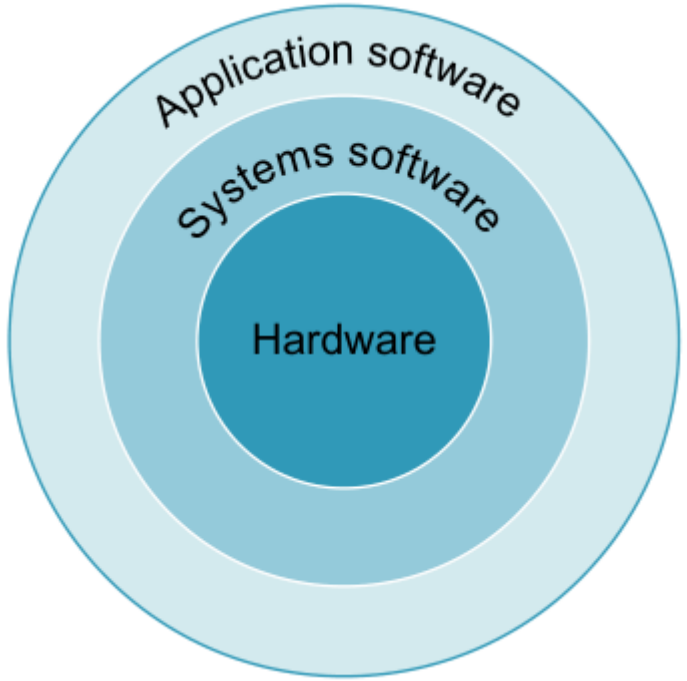


FIGURE 1.9 The processor integrated circuit inside the A12 package. The size of chip is 8.4 by 9.91 mm, and it was manufactured originally in a 7-nm process (see Section 1.5). It has two identical ARM processors or cores in the lower middle of the chip, four small cores on the lower right of the chip, a graphics processing unit (GPU) on the far right (see Section 6.6), and a domain-specific accelerator for neural networks (see Section 6.7) called the NPU on the far left. In the middle are second-level cache memory (L2) banks for the big and small cores (see Chapter 5). At the top and bottom of the chip are interfaces to the main memory (DDR DRAM). (Courtesy TechInsights, www.techinsights.com)

System Software

- Operating System
  - Supervising program that interfaces the user's program with the hardware (e.g., Linux, iOS, Windows)
  - Handles basic input and output operations
  - Allocates storage and memory
  - Provides for protected sharing among multiple applications
- Compiler
  - Translate programs written in a high-level language(e.g., C, Java) into instructions that the hardware can execute



System Software

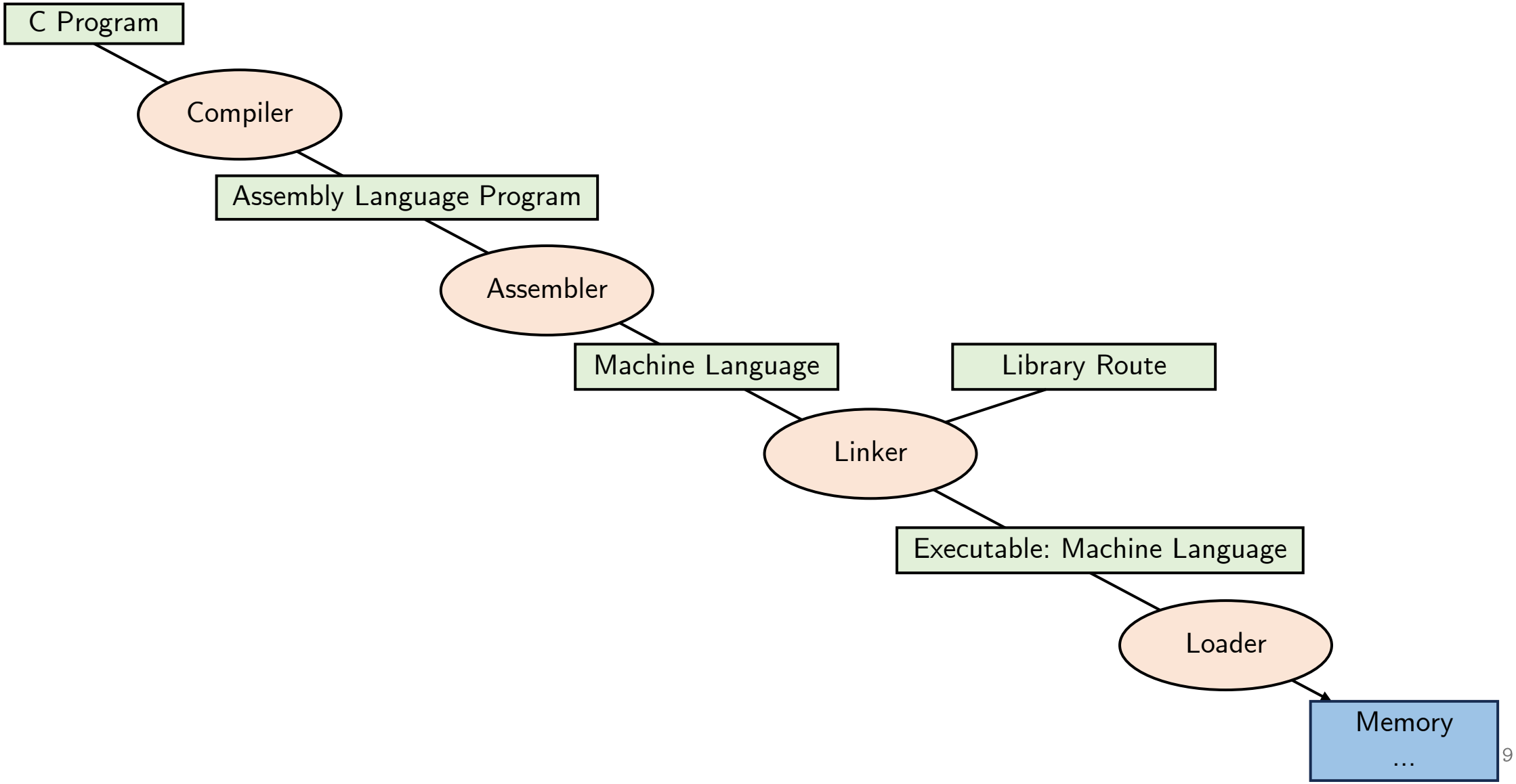
- Which one is not a task for an operating system?
  - A: output images to the screen.
  - B: create files.
  - C: translate C program to assembly language.
  - D: switch processes that run on the processor.
- Answer: C

Advantages of High-Level Languages?

- Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java 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

Very little programming is done today at the assembler lever.

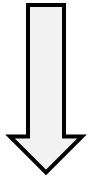
Traditional Compilation Flow



Below the Program

- High-level language program (in C)

```
void swap(int v[], int k) {
    int tmp;
    tmp = v[k];
    v[k] = v[k + 1];
    v[k + 1] = tmp;
}
```

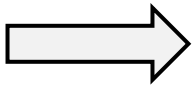


Compiler

- Assembly language (in RISC-V)

```
swap(int*, int):
    slli    a1,a1,2
    add     a0,a0,a1
    lw      a4,0(a0)
    lw      a5,4(a0)
    sw      a4,4(a0)
    sw      a5,0(a0)
    ret
```

Assembler



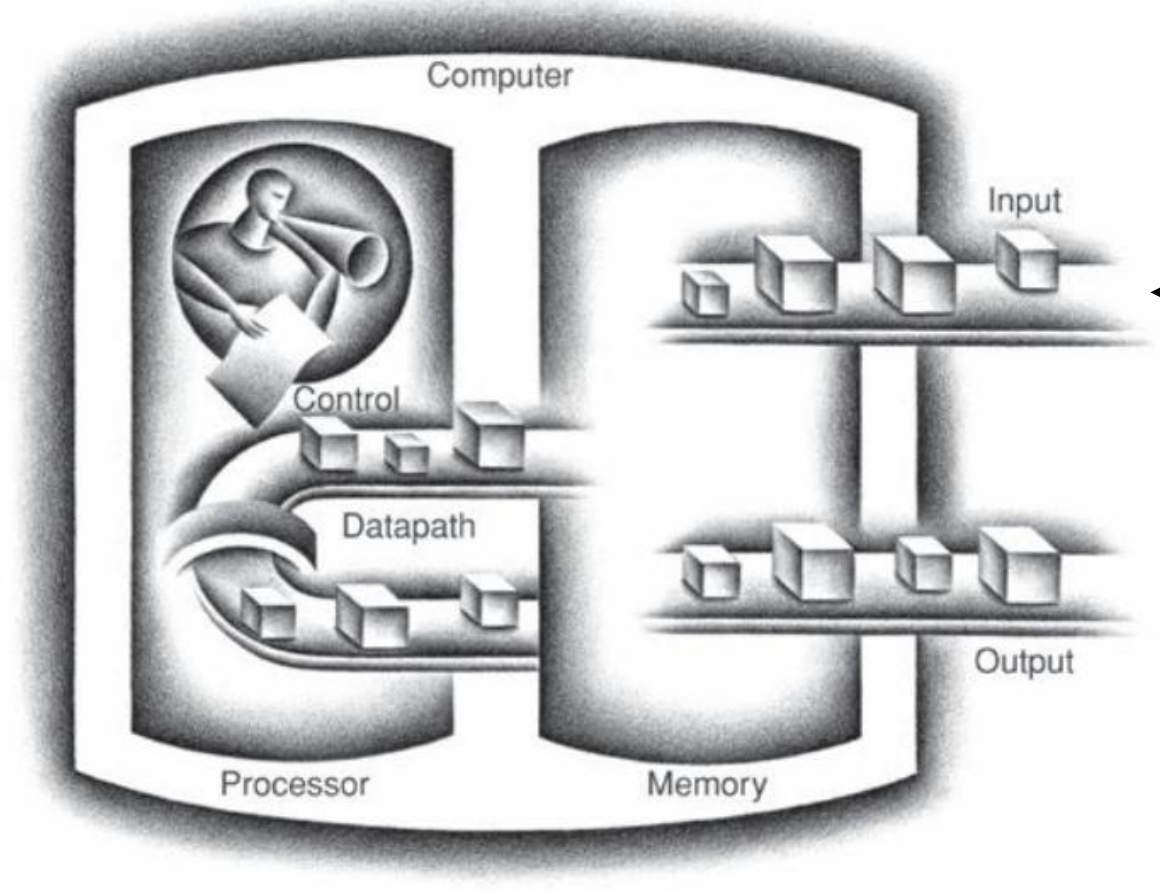
- Online RISC-V Compiler:
  - https://godbolt.org/
- Online RISC-V Assembler:
  - https://riscvasm.lucasteske.dev/#

- Machine language (in RISC-V)

```
 0:   00259593          slli a1,a1,0x2
 4:   00b50533          add  a0,a0,a1
 8:   00052703          lw   a4,0(a0)
 c:   00452783          lw   a5,4(a0)
10:   00e52223          sw   a4,4(a0)
14:   00f52023          sw   a5,0(a0)
```

There is an error in FIGURE 1.4. Can you find it?

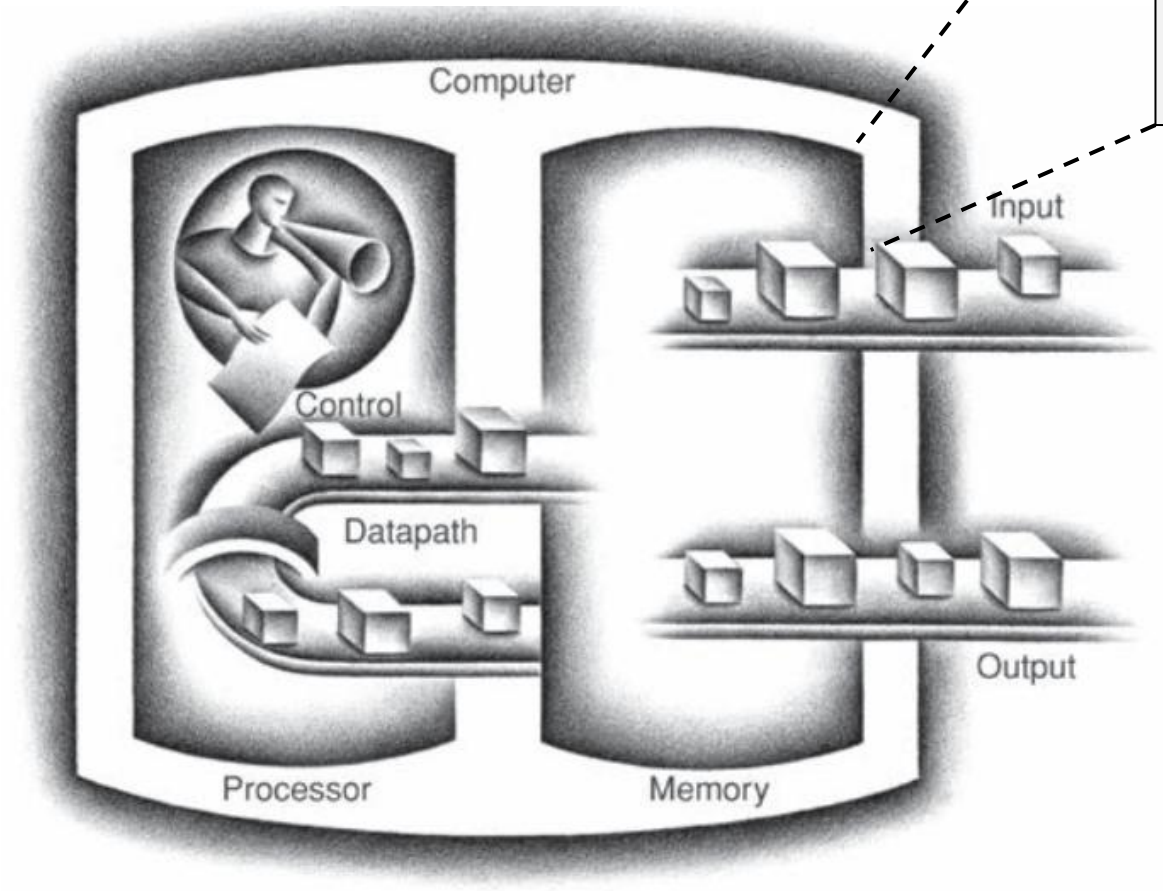
Input Device Inputs Machine Code




```
   0:   00259593          slli a1,a1,0x2
   4:   00b50533          add  a0,a0,a1
   8:   00052703          lw   a4,0(a0)
   c:   00452783          lw   a5,4(a0)
  10:   00e52223          sw   a4,4(a0)
  14:   00f52023          sw   a5,0(a0)
```

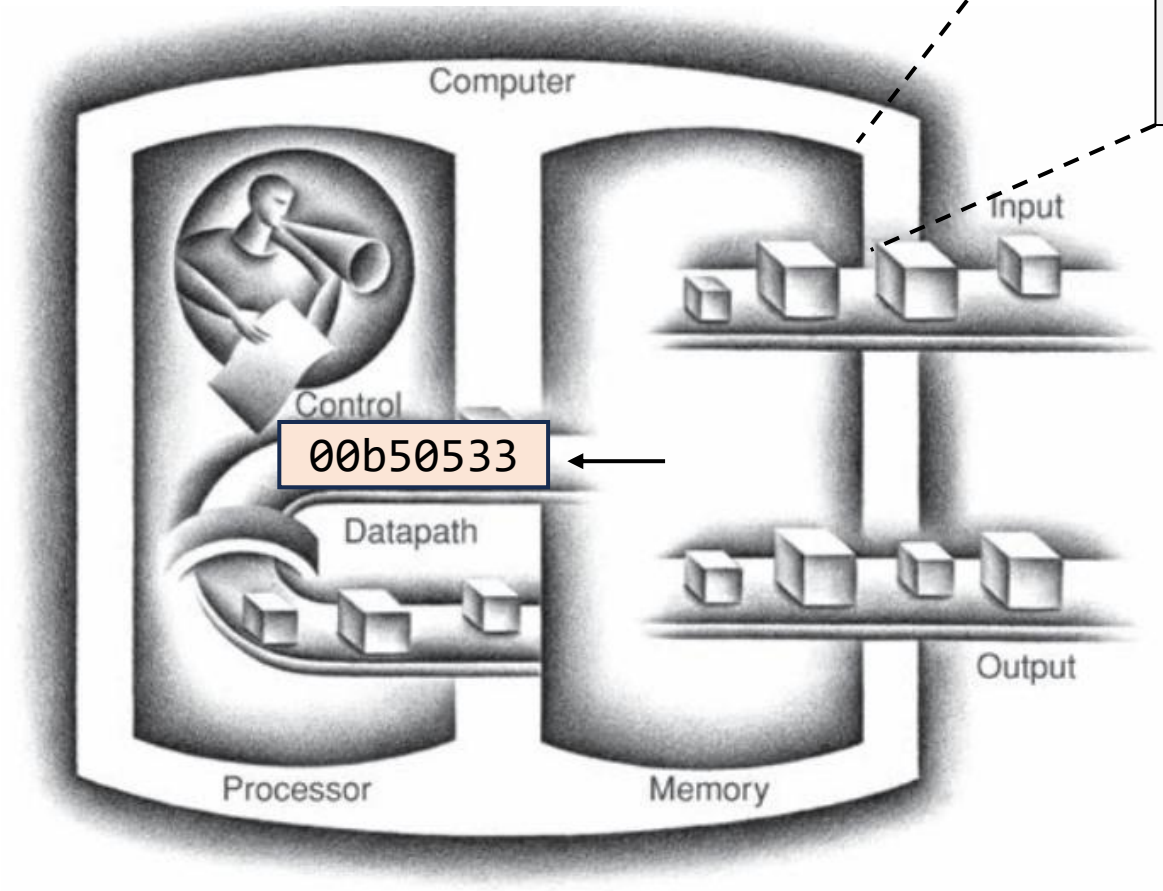
Machine Code in Memory

```
  0:   00259593          slli a1,a1,0x2
  4:   00b50533          add  a0,a0,a1
  8:   00052703          lw   a4,0(a0)
  c:   00452783          lw   a5,4(a0)
 10:   00e52223          sw   a4,4(a0)
 14:   00f52023          sw   a5,0(a0)
```



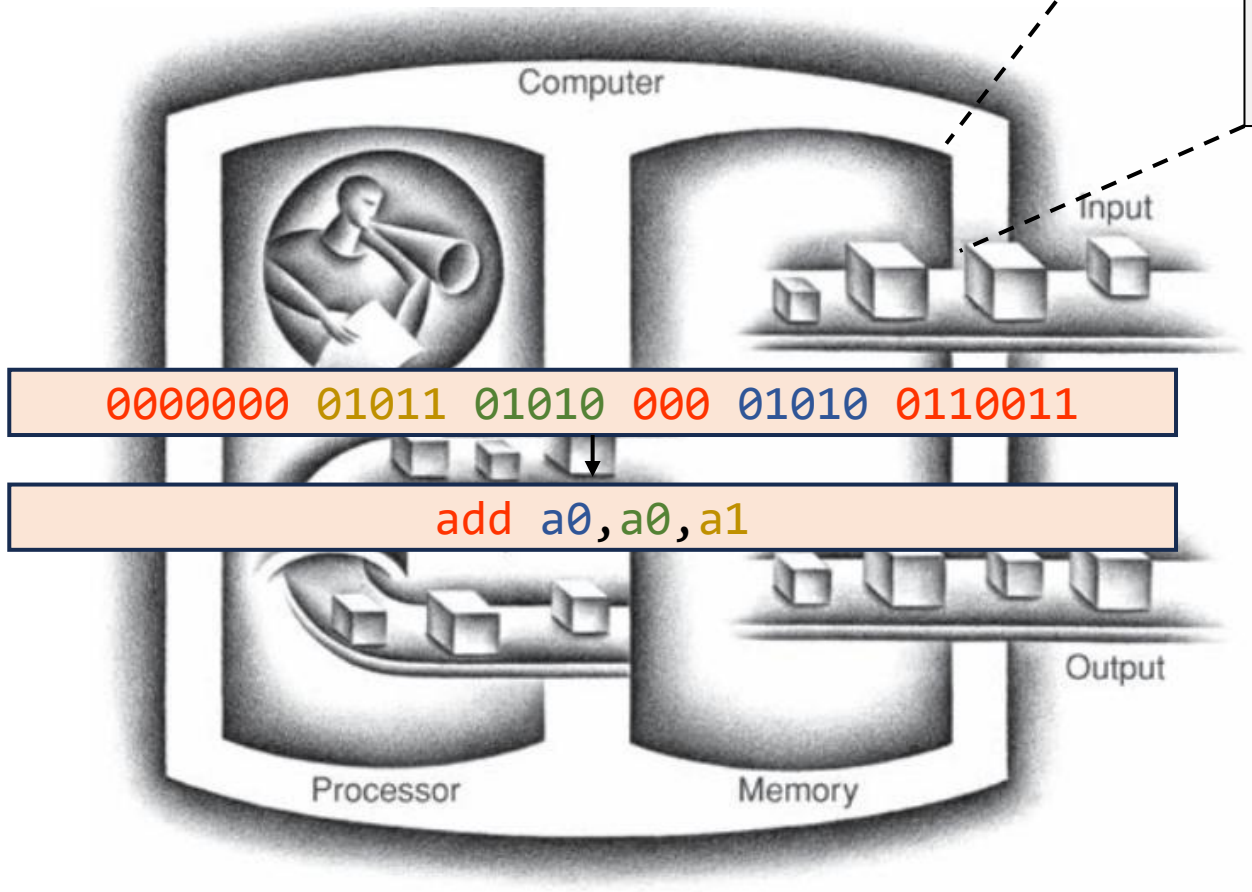
Control Fetches One Instruction

```
 0:   00259593          slli a1,a1,0x2
 4:   00b50533          add  a0,a0,a1
 8:   00052703          lw   a4,0(a0)
 c:   00452783          lw   a5,4(a0)
10:   00e52223          sw   a4,4(a0)
14:   00f52023          sw   a5,0(a0)
```



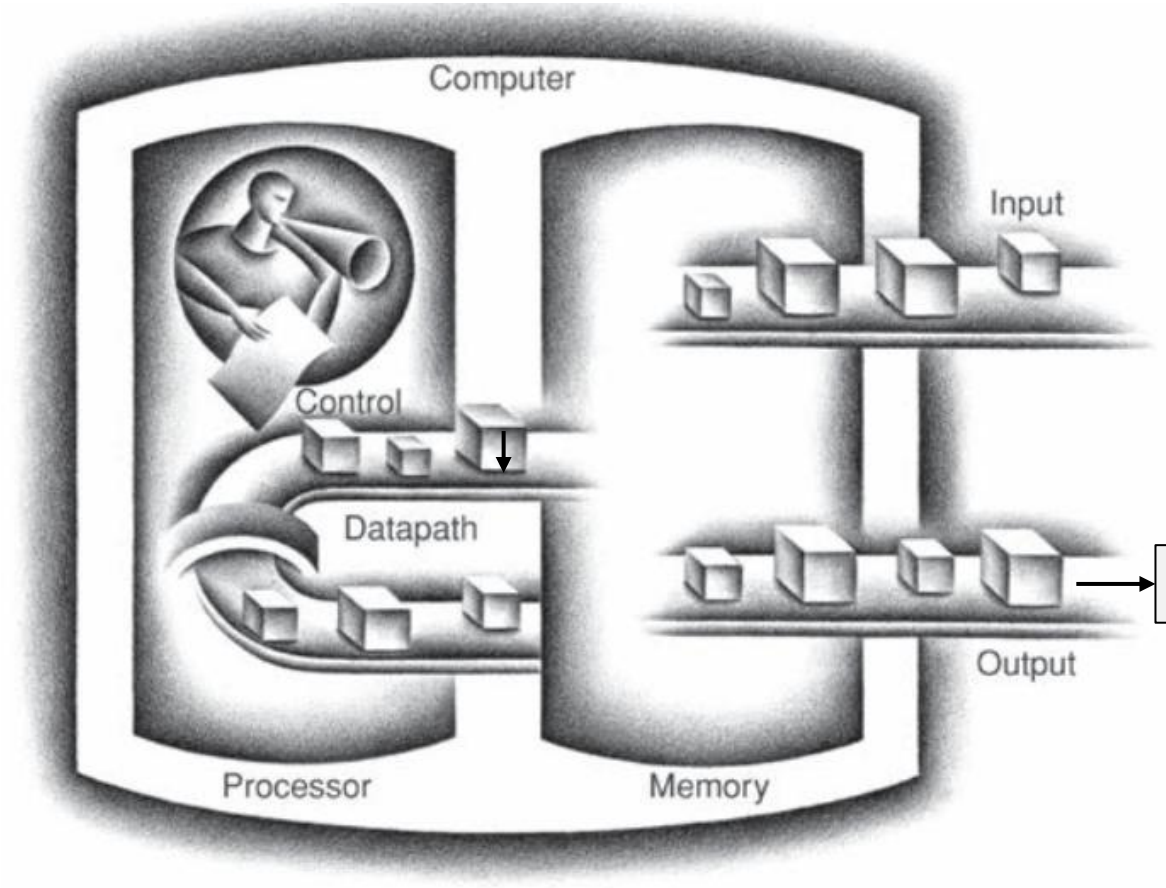
Decodes and Execute, and Fetch Again...

```
 0:   00259593          slli a1,a1,0x2
 4:   00b50533          add  a0,a0,a1
 8:   00052703          lw   a4,0(a0)
 c:   00452783          lw   a5,4(a0)
10:   00e52223          sw   a4,4(a0)
14:   00f52023          sw   a5,0(a0)
```



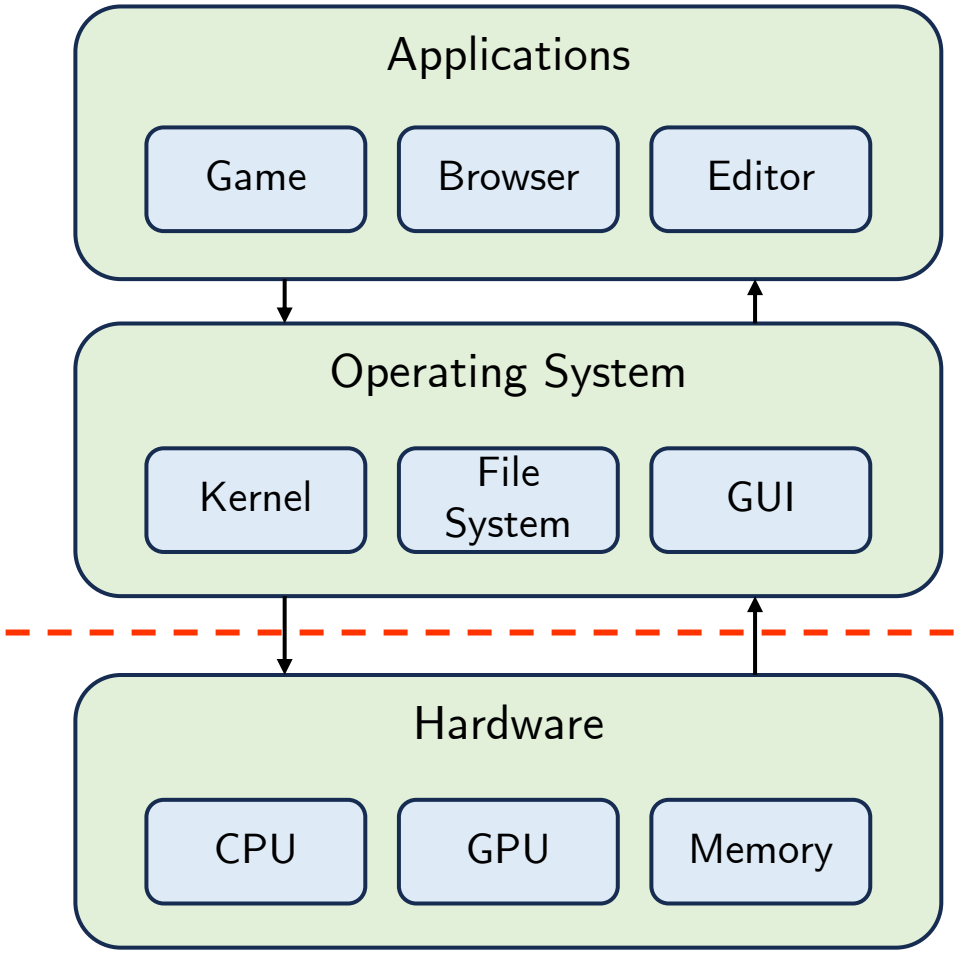
- Processor fetches the next instruction from memory
- How does it know which location in memory to fetch from next?

Write Outputs

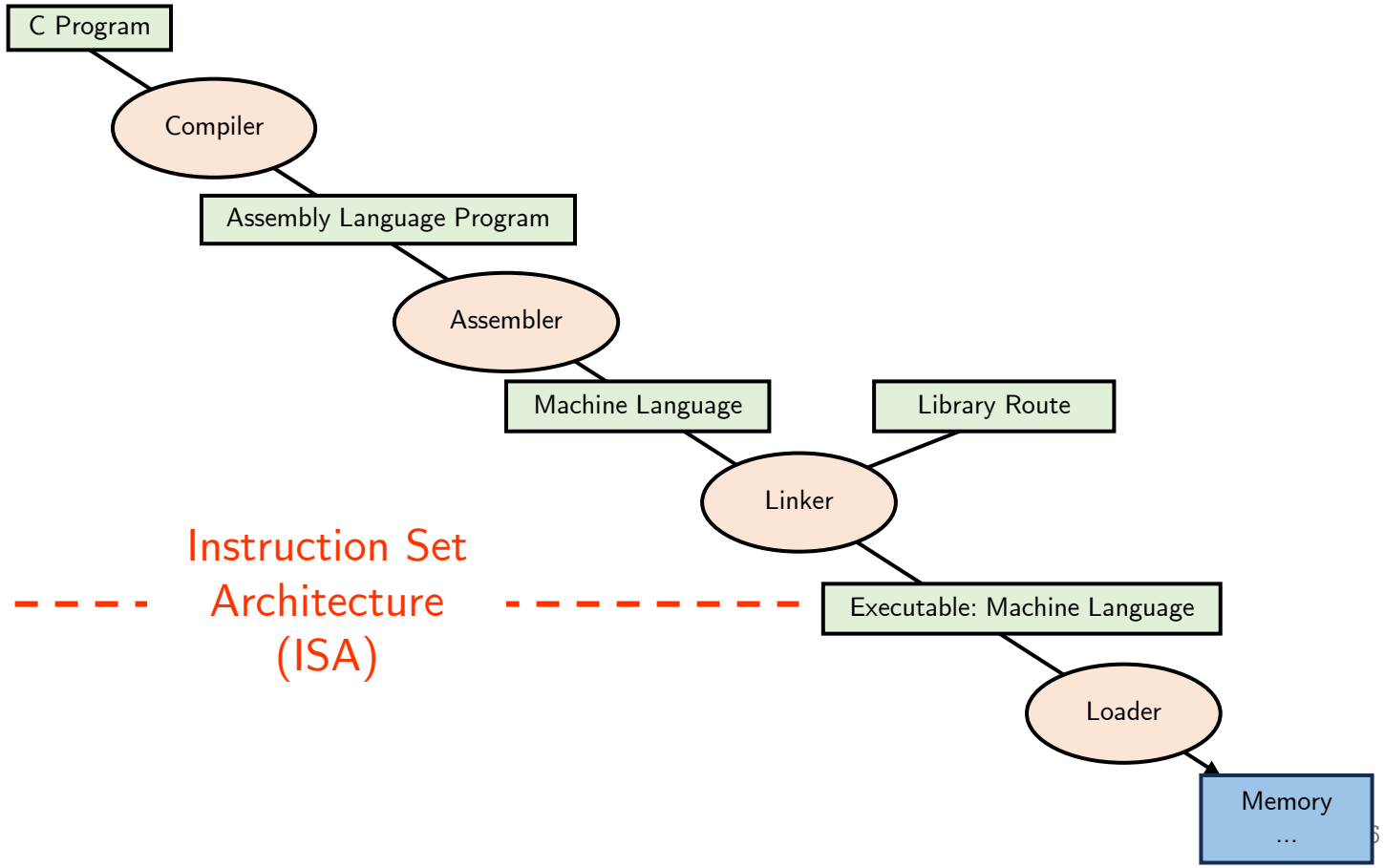



```
0:  deadbeef
```

ISA – Bridge between HW and SW



Instruction Set Architecture (ISA)



ISA and ABI

- ISA, or simply architecture – the abstract interface between the hardware and the lowest level software that includes all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, ..
- Enables implementations of varying cost and performance to run identical software
- The combination of the basic instruction set (the ISA) and the operating system interface is called the application binary interface (ABI)
- ABI: The user portion of the instruction set plus the operating system interfaces used by application programmers. Defines a standard for binary portability across computers.

Stored-Program Concept

- **Everything** is data, including the program.
  - Programs are compiled to machine binary.
  - Stored in memory and loaded into the processor for execution.
  - Can be modified even during execution.

- This enables:
  - Programs can be shipped as binary files – binary compatibility.
  - Computers can inherit ready-made software provided they are compatible with existing ISA – leads the industry to align around a small number of ISAs.

Assembly Language

- The language of the machine
  - Want an ISA that makes it easy to build the hardware and the compiler while
  - maximizing performance and minimizing cost
- Our targets: **RISC-V** ISA
  - Like other ISAs developed since the 1980's
  - RISC-V is originated from MIPS, the latter of which is used by Broadcom, Cisco, NEC, Nintendo, Sony, ...
- Design Goals:
  - Maximize performance, minimize cost, reduce design time (time-to-market), minimize memory space (embedded systems), minimize power consumption (mobile systems).

CISC vs. RISC

- Complex Instruction Set Computer (CISC)
  - Lots of complex instructions with variable length.
  - Very memory optimal (when memory capacity was very limited).
  - E.g., x86.
- Reduced Instruction Set Computer (RISC)
  - Simple instructions of a fixed size. Usually called a load/store architecture.
  - Simplify hardware design at cost of more instructions.
  - E.g., Arm, MIPS, RISC-V, IBM PowerPC, ...

- **Today there is no clear boundary**:
  - x86 is decoded into micro-ops (similar to RISC instructions) before execution.
  - RISC ISA introduces more complex instructions (e.g., matrix multiplication) for efficiency.

History of MIPS

- Used in many embedded systems.
  - E.g., Nintendo-64, PlayStation 1, PlayStation 2

Welcome to RISC-V

- RISC-V
  - An open standard instruction set architecture (ISA)
  - A clean break from the earlier MIPS-inspired designs
  - Modular ISA organization
  - Open standards, numerous proprietary and open-source cores
  - Managed by RISC-V Foundation

RISC-V ISA I

- Instruction Categories:
  - Arithmetic
  - Data transfer
  - Bitwise operation
  - Control transfer
  - Pseudo instruction
- 6 base instruction formats: all **32-bit** wide:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Registers

- Operands of arithmetic op must be a register (small but fast).
- RISC-V uses 32-bit registers.
  - 32-bit is also called a word in RISC-V.
  - Similarly, 64-bit is a doubleword.
  - 32 registers (5-bit to encode).

Register	ABI Name	Description	Saver
x0	zero	Zero constant	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	—
x3	gp	Global pointer	—
x4	tp	Thread pointer	Callee
x5	t0-t2	Temporaries	Caller
x8	s0 / fp	Saved / frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Fn args/return values	Caller
x12-x17	a2-a7	Fn args	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller
f0-7	ft0-7	FP temporaries	Caller
f8-9	fs0-1	FP saved registers	Callee
f10-11	fa0-1	FP args/return values	Caller
f12-17	fa2-7	FP args	Caller
f18-27	fs2-11	FP saved registers	Callee
f28-31	ft8-11	FP temporaries	Caller

Registers

- Which register cannot we store an operand when we call “max(a, b)”?
  - A: x0 (zero)
  - B: x3 (gp)
  - C: x9 (s1)
  - D: x11 (a1)
- Answer: A