M1 – Computers and Data

Outline

- Components of the System
- Instructions
- Architecture vs. Organization
- The Compute Stack

Computer

- Programs
  - Instructions and Data

Computer

- Programs
  - Instructions and Data
- Computer: An electronic device which is capable of receiving information (**data**) and performing a sequence of operations defined by **instructions** (**program**) to produce a result in the form of information

Instructions

- Tasks that a processor can execute
  - eg. add, mul, beq, load, store, call, ...

Instructions

- Tasks that a processor can execute
  - eg. add, mul, beq, load, store, call, ...
- Defined in the “**Instruction Set Architecture (ISA)**”

Instructions

- Tasks that a processor can execute
  - eg. add, mul, beq, load, store, call, ...
- Defined in the “**Instruction Set Architecture (ISA)**”
- Several ISAs exist
  - ARM, x86, POWER, MIPS, ...

Instructions

- Which instructions to implement?

Instructions

- Which instructions to implement?
- Desktop publishing vs. Gaming
  - Integer operations vs. Floating point operations

Instructions

- Which instructions to implement?
- Desktop publishing vs. Gaming
  - Integer operations vs. Floating point operations
- Database program vs. Video processing
  - I/O processing vs. Floating point operations

Instructions

- Which instructions to implement?
- Desktop publishing vs. Gaming
  - Integer operations vs. Floating point operations
- Database program vs. Video processing
  - I/O processing vs. Floating point operations
- Some fundamental instructions exist
  - Arithmetic, Logic, Memory transfer, Control statements, Priviliged instructions, ...

Instruction Tasks

```
a = b + c;
```

- **Compiler** converts a=b+c into a ***list of tasks*** the processor can accomplish

Instruction Tasks

a = b + c;

- Compiler converts a=b+c into a ***list of tasks*** the processor can accomplish
- Inputs: **b** and **c**. Output: **a**. Operation: addition.

Instruction Tasks

```
a = b + c;
```

- Compiler converts a=b+c into a ***list of tasks*** the processor can accomplish
- Inputs: **b** and **c**. Output: **a**. Operation: addition.
- Addition operations is performed in the Processor
  - Arithmetic and Logic Unit

Instruction Tasks

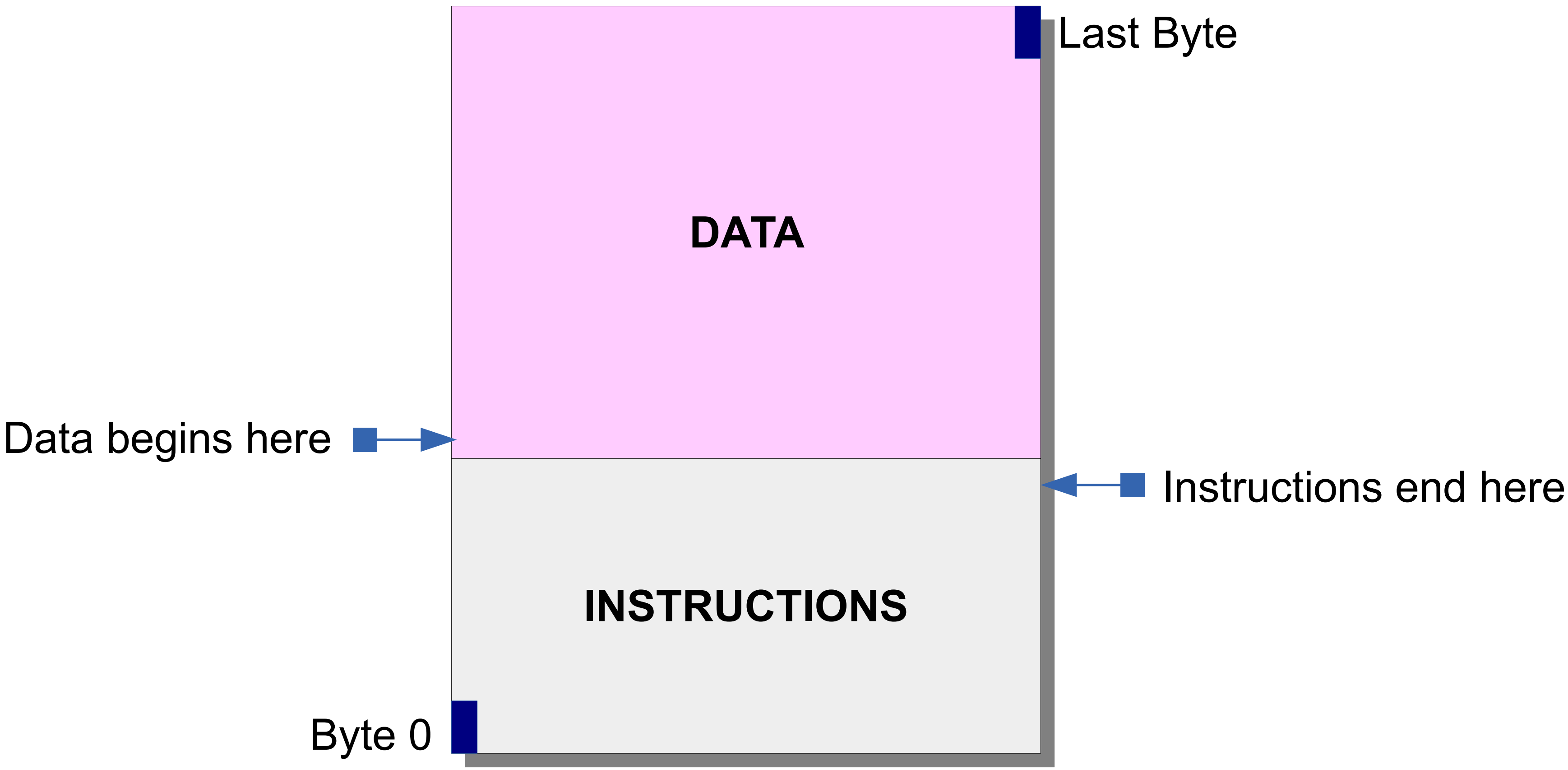
```
a = b + c;
```

- Compiler converts a=b+c into a ***list of tasks*** the processor can accomplish
- Inputs: **b** and **c**. Output: **a**. Operation: addition.
- Addition operations is performed in the Processor
  - Arithmetic and Logic Unit
- a, b, and c are in the Memory
  - Assume: they have to be inside the proces addition to begin

Minor Detour - Memory

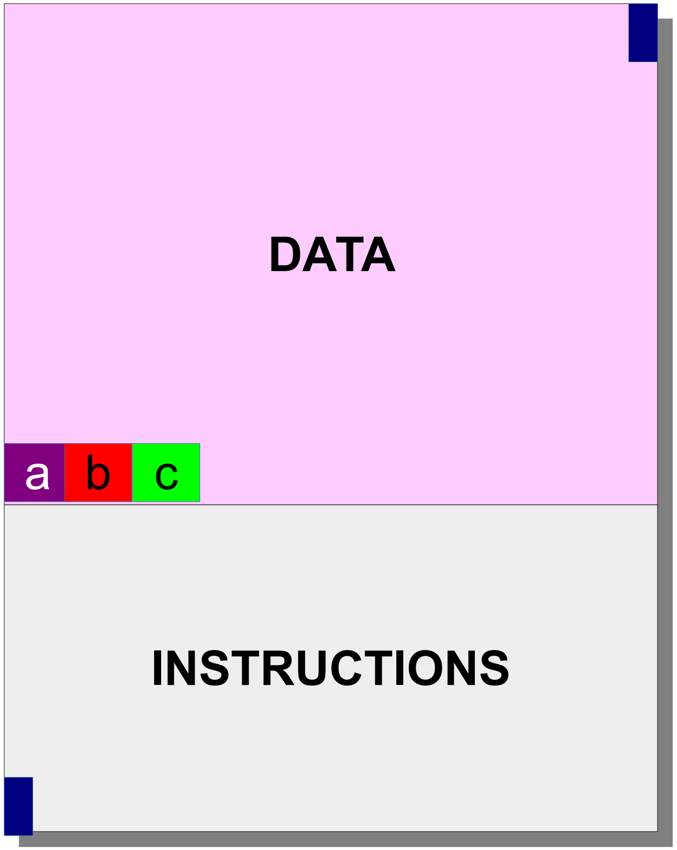
The Program in Memory

Simplified View



The Program in Memory

Simplified View

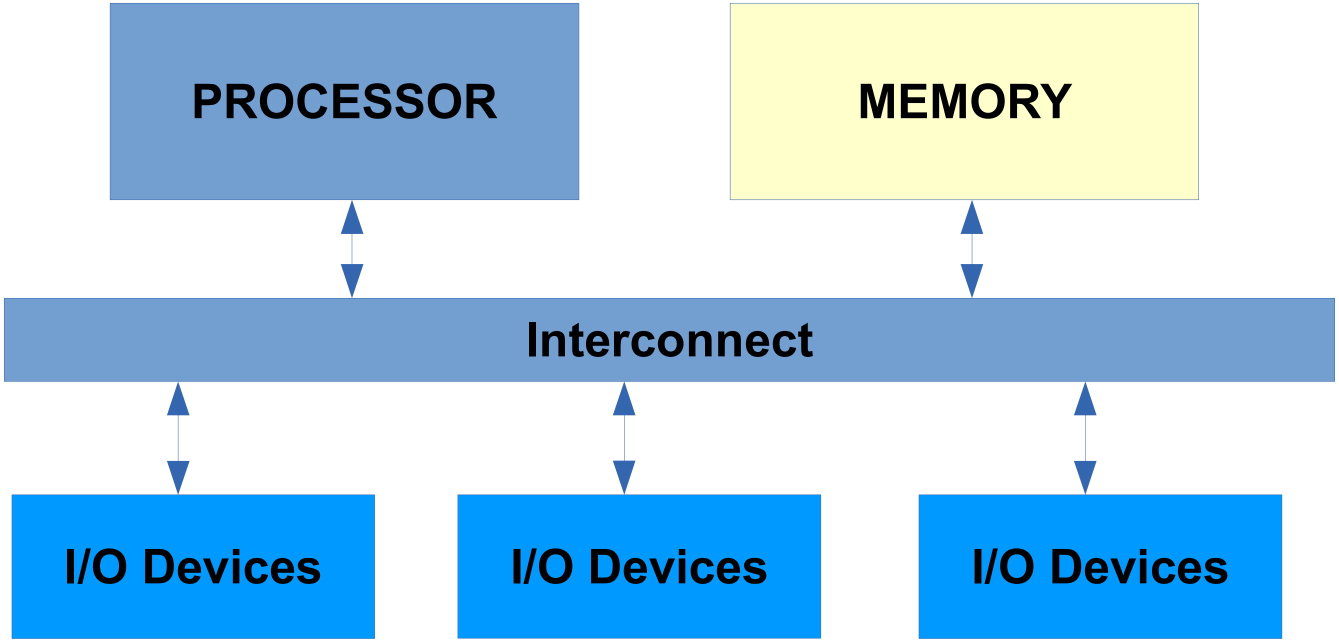


a, b, and c identify unique locations in program memory!

More details later !

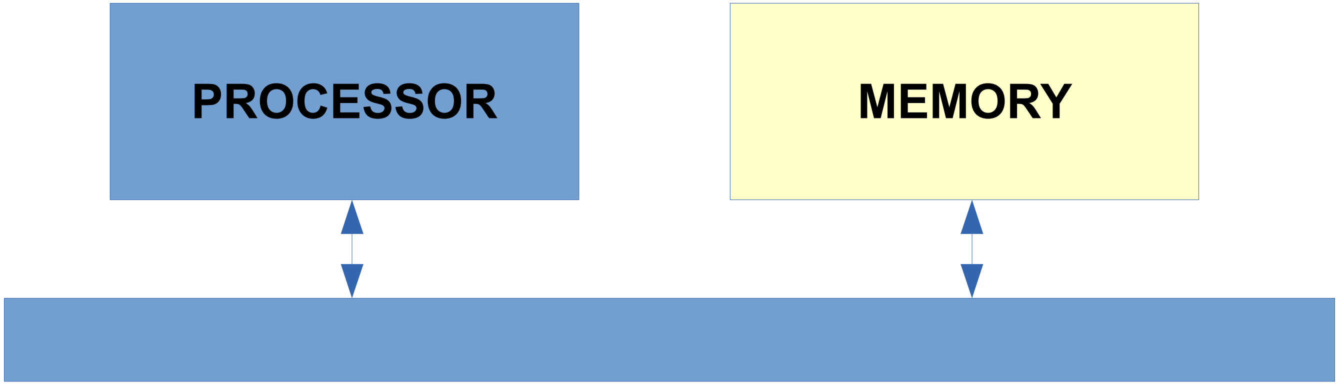
Instruction Tasks

a = b + c;



Instruction Tasks

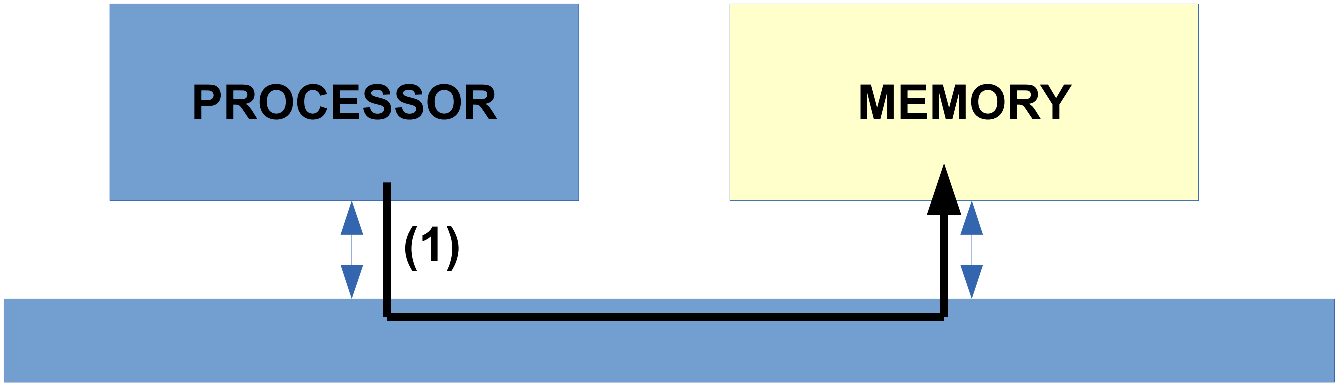
```
a = b + c;
```



1

Instruction Tasks

```
a = b + c;
```

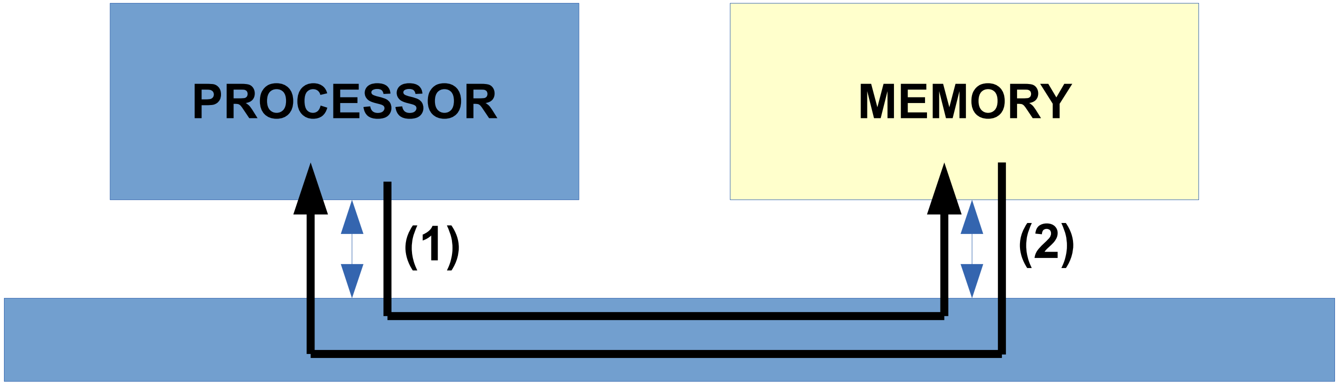


(1) Processor requests for value at location ‘b’ from memory

1

Instruction Tasks

a = b + c;



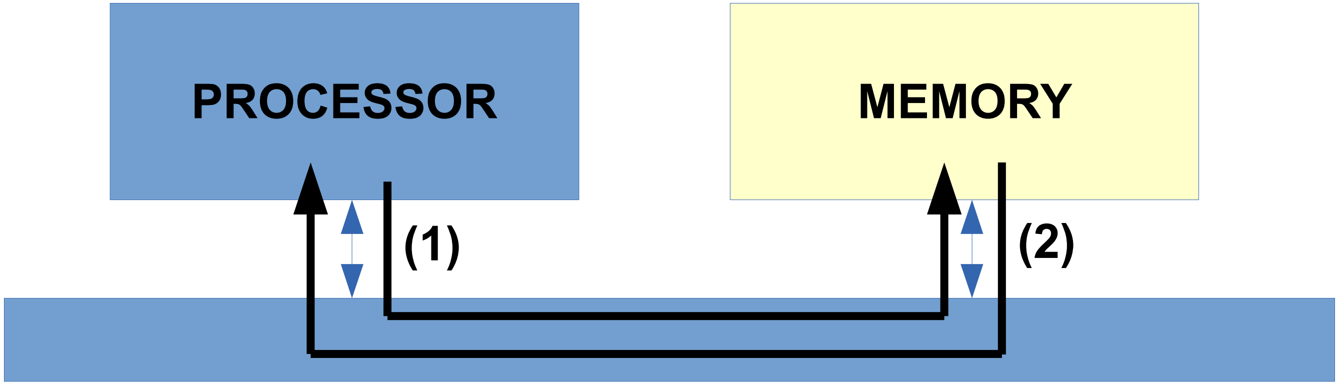
(1) Processor requests for value at location ‘b’ from memory
(2) Memory sends value at location ‘b’ to the processor

1

Instruction Tasks

a = b + c;

LOAD b



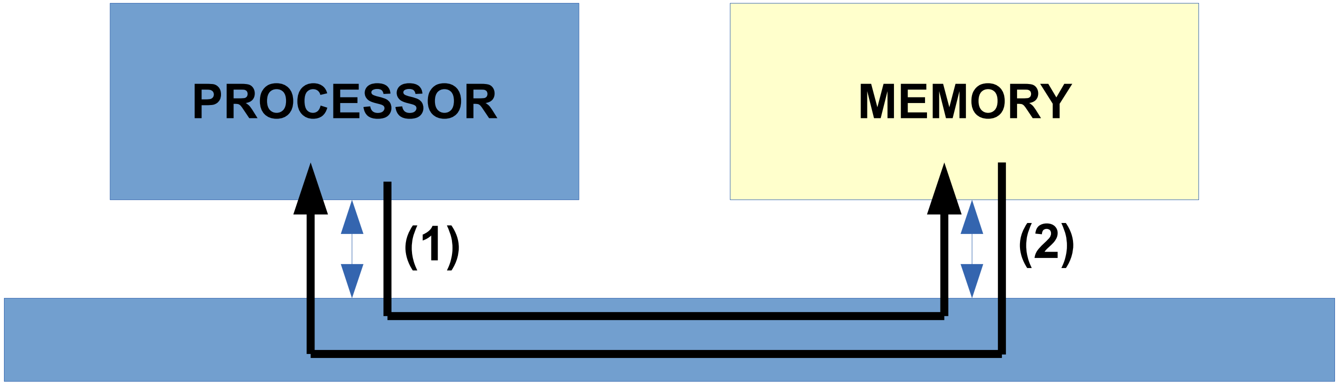
(1) Processor requests for value at location ‘b’ from memory
(2) Memory sends value at location ‘b’ to the processor

2

Instruction Tasks

a = b + c;

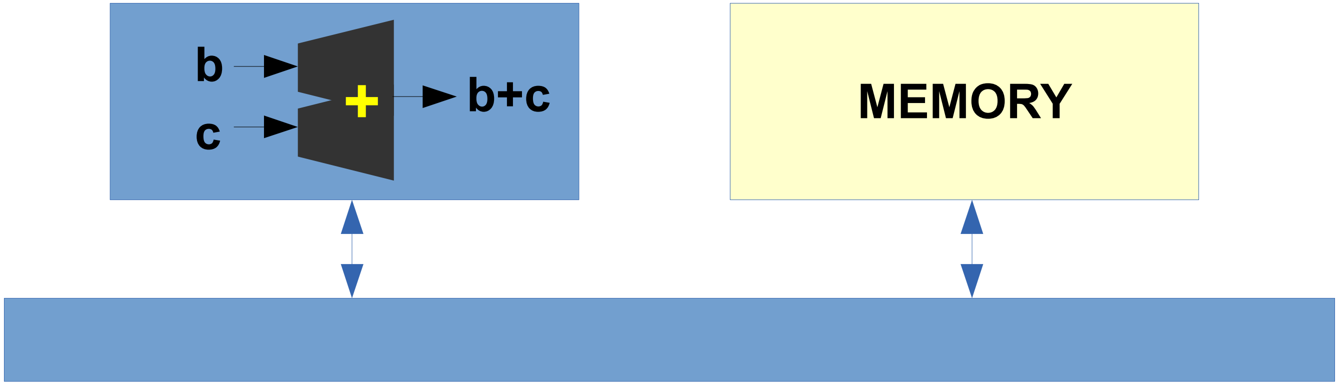
LOAD c



(1) Processor requests for value at location ‘b’ from memory
(2) Memory sends value at location ‘b’ to the processor

Instruction Tasks

a = b + c;



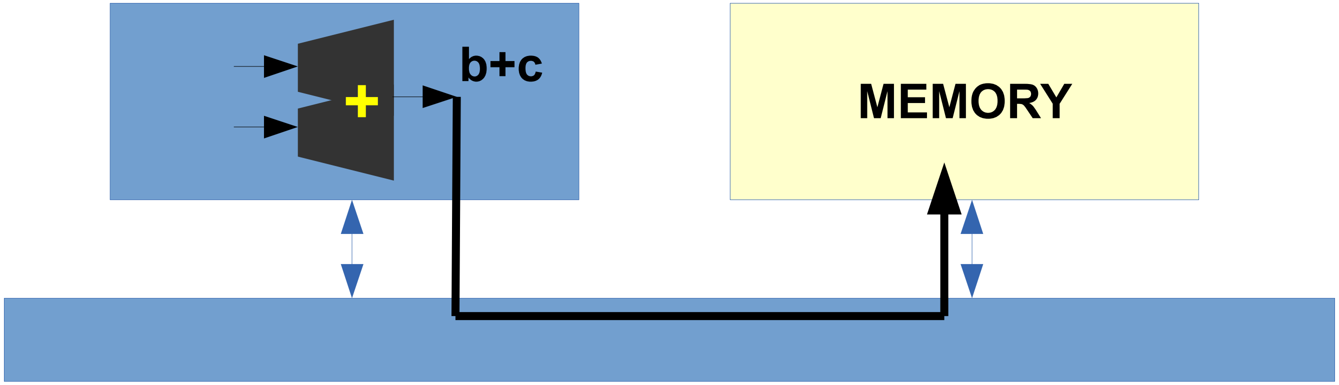
Add values from ‘b’ and ‘c’

4

Instruction Tasks

a = b + c;

STORE



Processor sends the sum to memory to put in location ‘a’

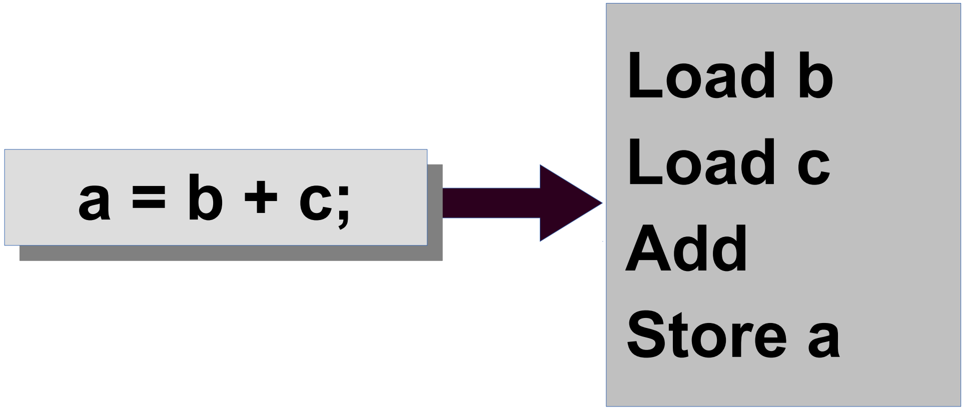
Instruction Tasks

```
a = b + c;
```

- Load value from memory location b
- Load value from memory location c
- Add these values
- Store sum into memory location ‘a’

Load and Store are memory transfer operations.
Add is an ALU operation

Instruction Tasks



a = b + c;

Load b
Load c
Add
Store a

- Load value from memory location b
- Load value from memory location c
- Add these values
- Store sum into memory location ‘a’

Instruction Tasks

```
a = b + c;
d = b + c + a;
f = d + a + e;
```

- List the fundamental tasks.
- Observations?
- Can you improve on the design?

Instruction Tasks

```
a = b + c;
d = b + c + a;
f = d + a + e;
```

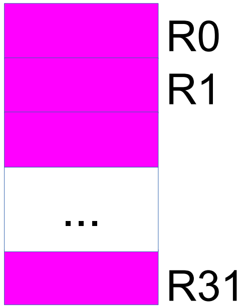
- Loads are repeated – b, c, a.
- Store and Load pairs – a, d.

Instruction Tasks

```
a = b + c;
d = b + c + a;
f = d + a + e;
```

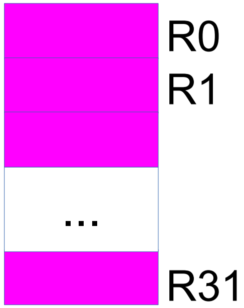
- Loads are repeated – b, c, a.
- Store and Load pairs – a, d.
- Eliminate repetitive/redundant operations
  - Store intermediate results
  - Register File (RF)

Register File



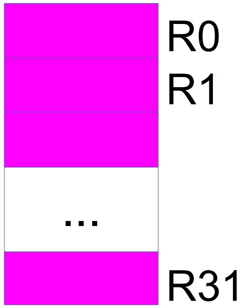
- Fast storage for quick access by the processor

Register File



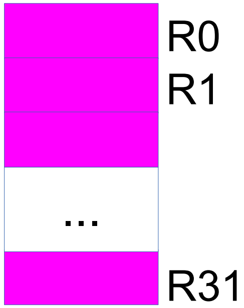
- Fast storage for quick access by the processor
- Small number of data elements Eg. 32.
- Accessed as registers R0, R1, R2, … R31.

Register File



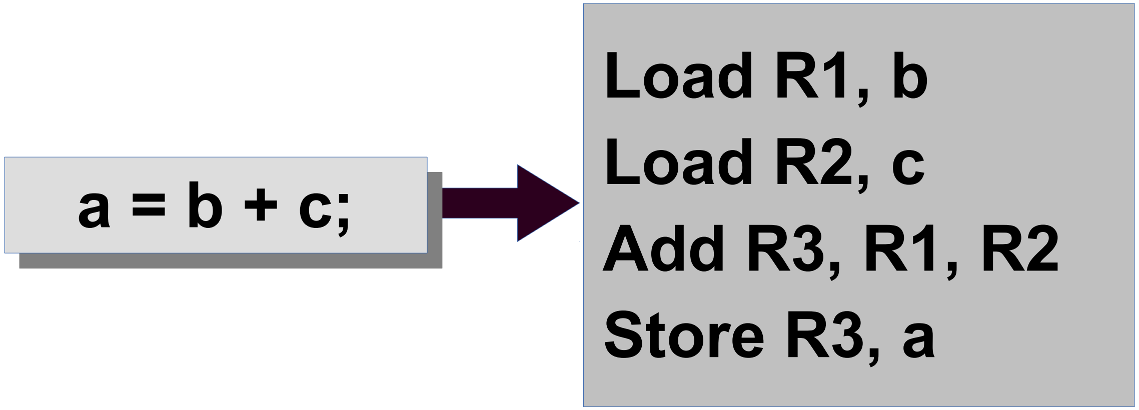
- Fast storage for quick access by the processor
- Small number of data elements Eg. 32.
- Accessed as registers R0, R1, R2, … R31.
- “load from memory into the processor” = “fetch data from memory and write into Register File”

Register File



- Fast storage for quick access by the processor
- Small number of data elements Eg. 32.
- Accessed as registers R0, R1, R2, … R31.
- “load from memory into the processor” = “fetch data from memory and write into Register File”
- Loaded values go in the RF; Results of computations go into the RF; Values from RF can be stored into memory

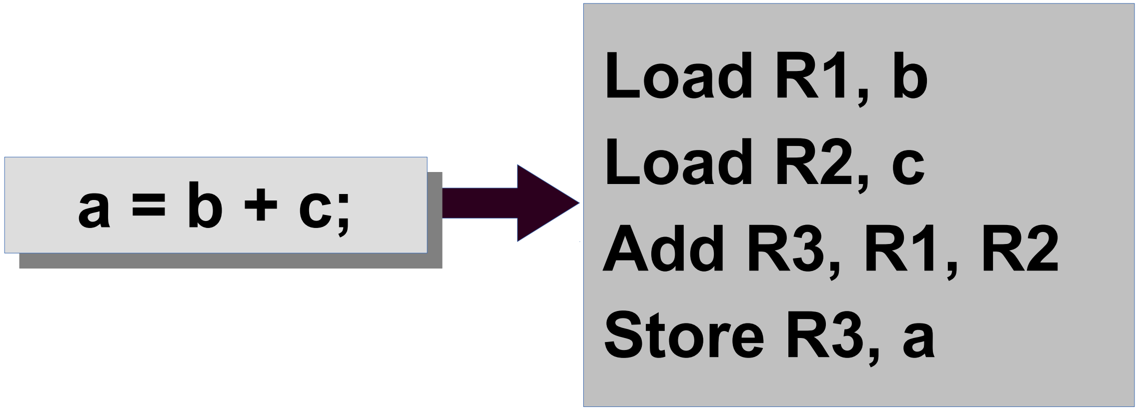
Instruction Tasks



```
a = b + c;
```

```
Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a
```

Instruction Tasks

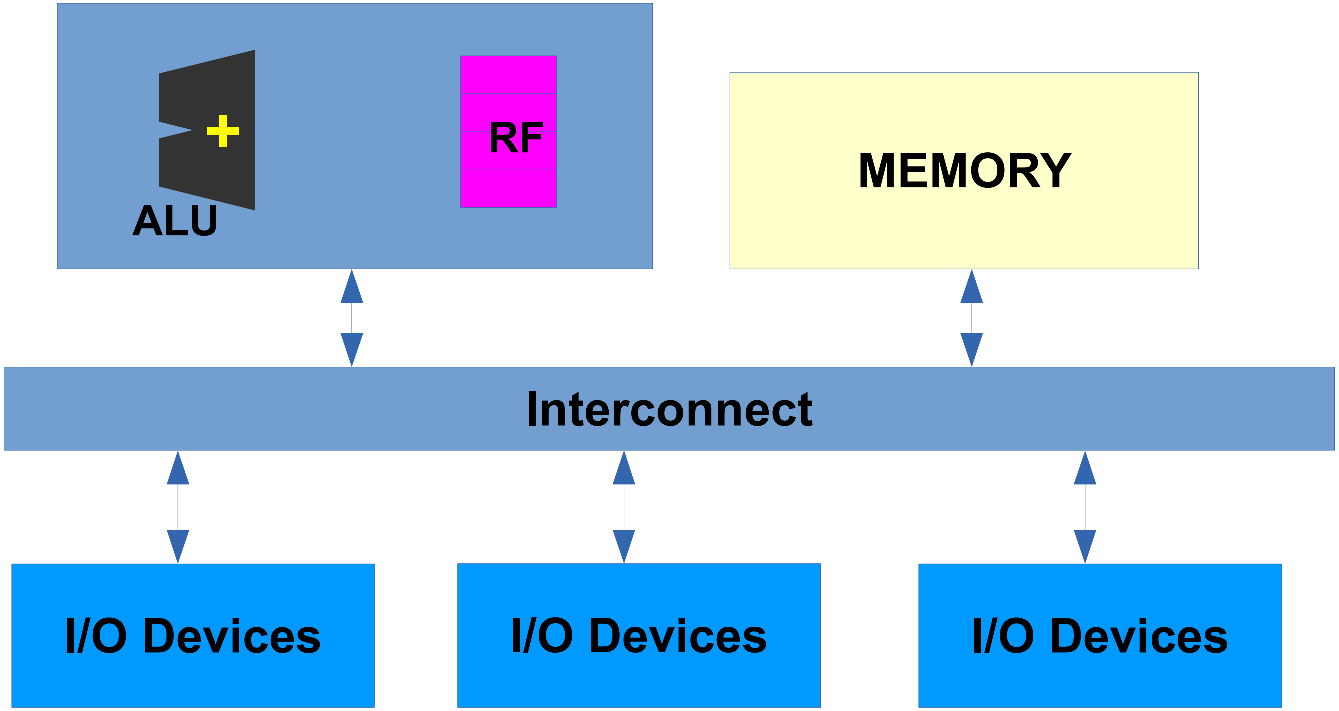


a = b + c;

Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a

- Load value from memory location b into R1
- Load value from from memory location c into R2
- Add contents of R1 and R2 and save the sum into R3
- Store contents of R3 into memory location ‘a’

The Computer System



Assembly Level Language

- Expression of high level language statements as their fundamental tasks

Assembly Level Language

- Expression of high level language statements as their fundamental tasks
- 1-to-1 mapping to the binary code

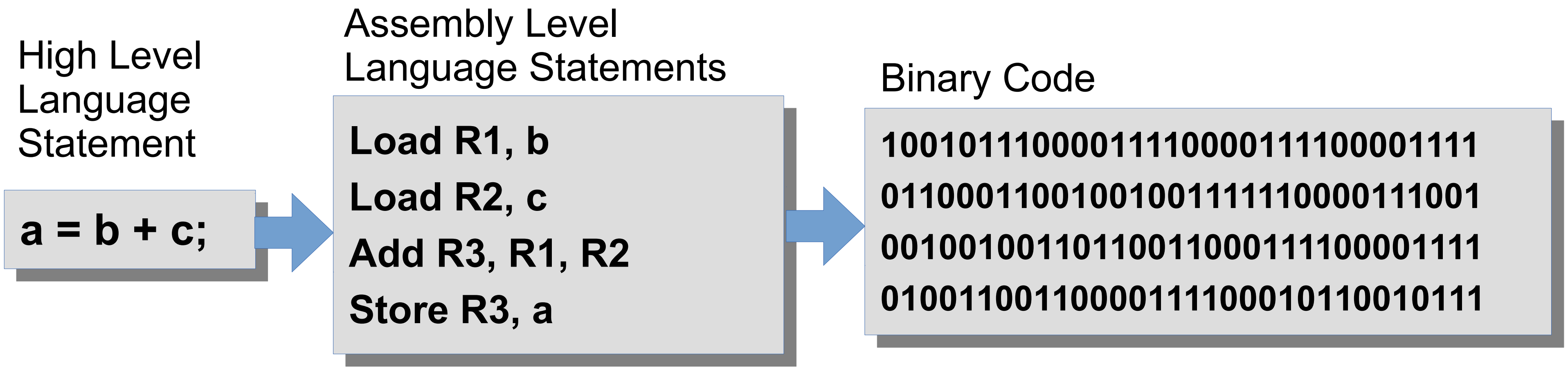
Assembly Level Language

- Expression of high level language statements as their fundamental tasks
- 1-to-1 mapping to the binary code
- **ISA** – list of instructions implemented in the processor

Assembly Level Language

- Expression of high level language statements as their fundamental tasks
- 1-to-1 mapping to the binary code
- **ISA** – list of instructions implemented in the processor
- Specific to a processor

Assembly Level Language



High Level Language Statement

a = b + c;

Assembly Level Language Statements

```
Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a
```

Binary Code

```
10010111000011110000111100001111
01100011001001001111110000111001
00100100110110011000111100001111
01001100110000111100010110010111
```

Instruction Set Architecture

- Also called “Architecture”
- Informally, a list of capabilities of the processor

Instruction Set Architecture

- Also called “Architecture”
- Informally, a list of capabilities of the processor
  - What operations are supported by this processor?

Instruction Set Architecture

- Also called “Architecture”
- Informally, a list of capabilities of the processor
  - What operations are supported by this processor?
  - What kind of data does the operation act on?

Instruction Set Architecture

- Also called “Architecture”
- Informally, a list of capabilities of the processor
  - What operations are supported by this processor?
  - What kind of data does the operation act on?
  - Where does the data for the operation come from?

Instruction Set Architecture

- Also called “Architecture”
- Informally, a list of capabilities of the processor
  - What operations are supported by this processor?
  - What kind of data does the operation act on?
  - Where does the data for the operation come from?
  - Which types of data are valid for an operation.

Instruction Set Architecture

- Also called “Architecture”
- Defines instructions (operations) the processor implements (supports)
- Input operands – number, size, type
- Input from Memory or from Registers
- Data Representation – Types/Sizes

ADD R3, R2, R1

ADD R3, R2, R1

- Operation: ADD. Input operands: R1, R2. Output Operand: R3

ADD R3, R2, R1

- Operation: ADD. Input operands: R1, R2. Output Operand: R3.
- Both input operands are 32b signed integers (R1 and R2)
  - vs. ADDU, ADDI, ADDIU

ADD R3, R2, R1

- Operation: ADD. Input operands: R1, R2. Output Operand: R3.
- Both input operands are 32b signed integers (R1 and R2)
  - vs. ADDU, ADDI, ADDIU
- Both input operands come from the Register File

ADD R3, R2, R1

- Operation: ADD. Input operands: R1, R2. Output Operand: R3.
- Both input operands are 32b signed integers (R1 and R2)
- Both input operands come from the Register File
- Processor contains hardware to feed the ALU from the RF

ADD R3, R2, R1

- Operation: ADD. Input operands: R1, R2. Output Operand: R3.
- Both input operands are 32b signed integers (R1 and R2)
- Both input operands come from the Register File
- Processor contains hardware to feed the ALU from the RF
- ALU contains hardware to add two numbers (Adder)

ADD R3, R2, R1

- Operation: ADD. Input operands: R1, R2. Output Operand: R3.
- Both input operands are 32b signed integers (R1 and R2)
- Both input operands come from the Register File
- Processor contains hardware to feed the ALU from the RF
- ALU contains hardware to add two numbers (Adder)
- Processor contains hardware to connect the ALU to update the RF with the result

Organization

- Also called **Microarchitecture**

Organization

- Also called **Microarchitecture**
- Several Adder circuits exist
  - Each with different gate counts and costs

Organization

- Also called **Microarchitecture**
- Several Adder circuits exist
  - Each with different gate counts and costs
- Choosing one adder that fits the space and cost constraints is an organization challenge

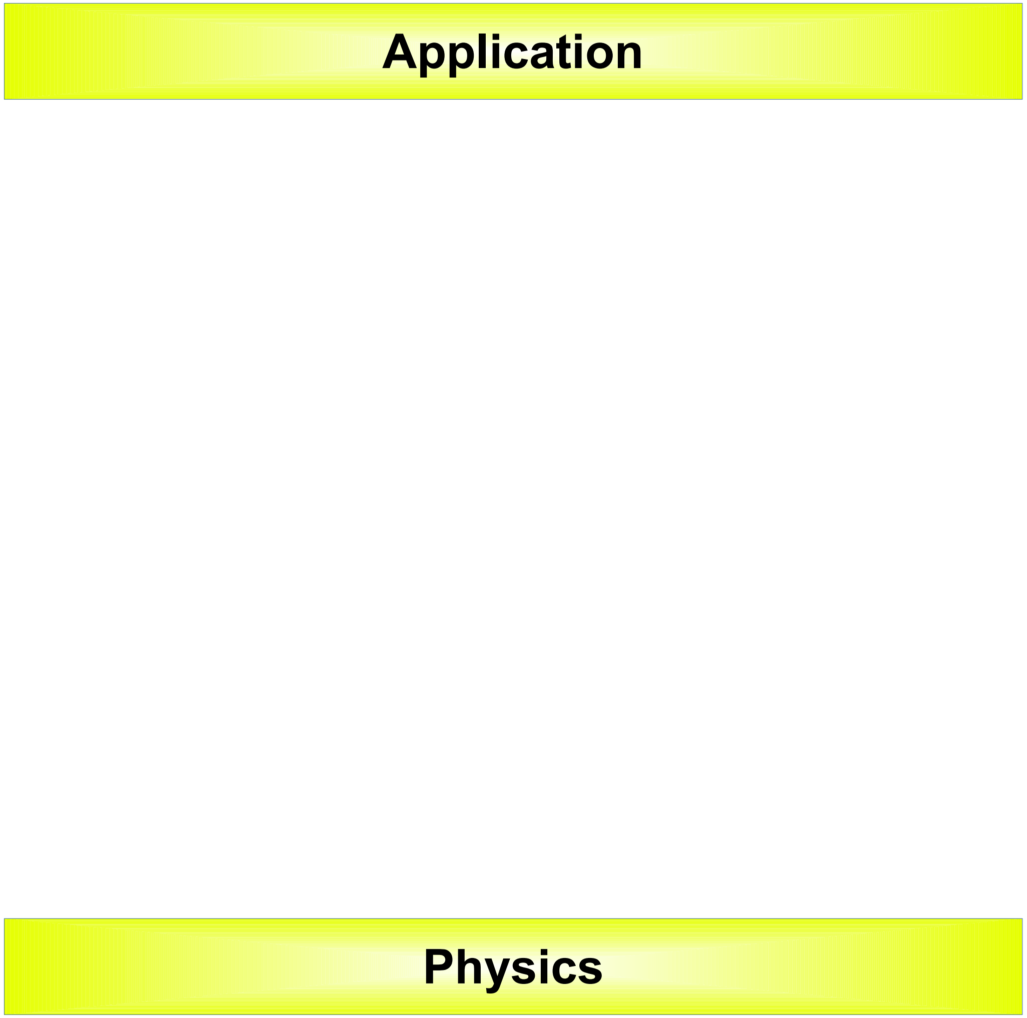
Organization

- Also called **Microarchitecture**
- Several Adder circuits exist
  - Each with different gate counts and costs
- Choosing one adder that fits the space and cost constraints is an organization challenge
- Informally, Organization is the way a given ISA is implemented in hardware on a processor

Organization

- Also called **Microarchitecture**
- Several Adder circuits exist
  - Each with different gate counts and costs
- Choosing one adder that fits the space and cost constraints is an organization challenge
- Informally, Organization is the way a given ISA is implemented in hardware on a processor
- Architecture describes **what** the computer does and organization describes **how** it does it.

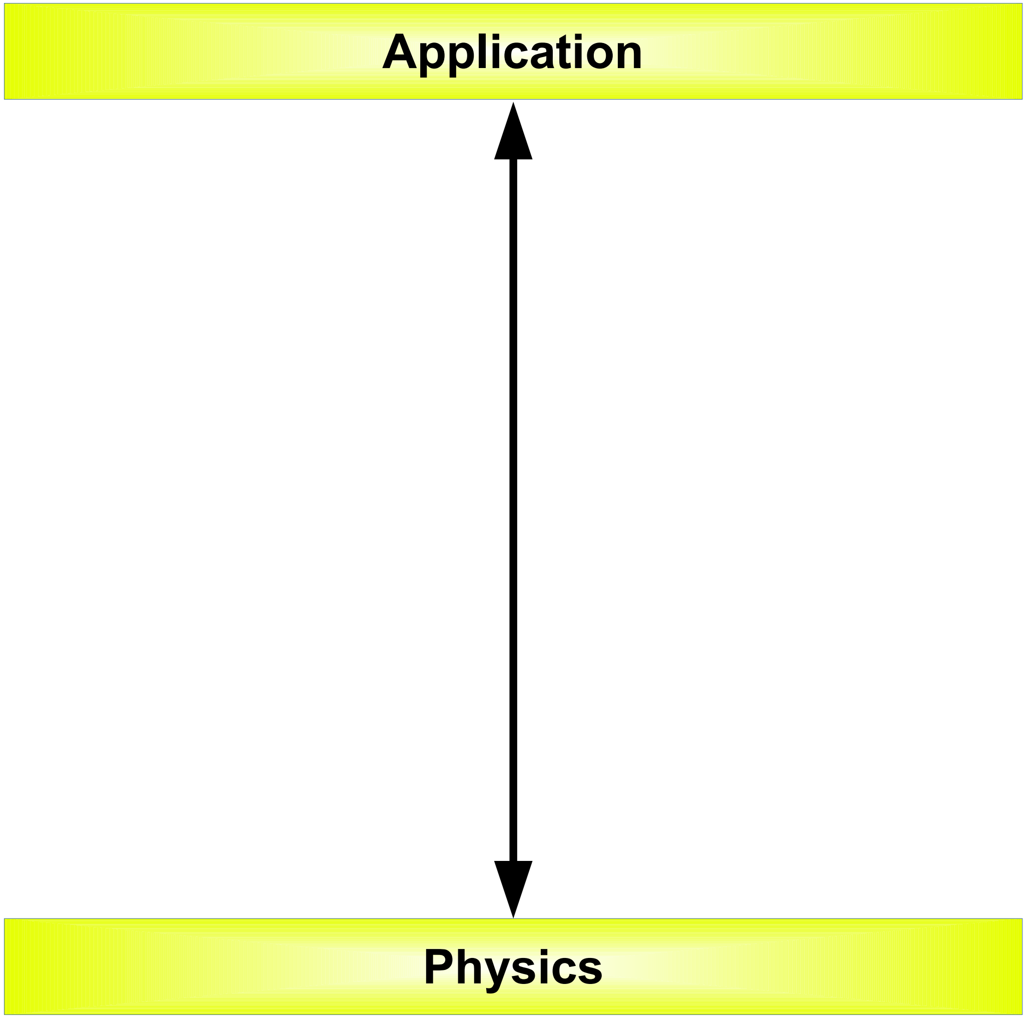
The Compute Stack



Application

Physics

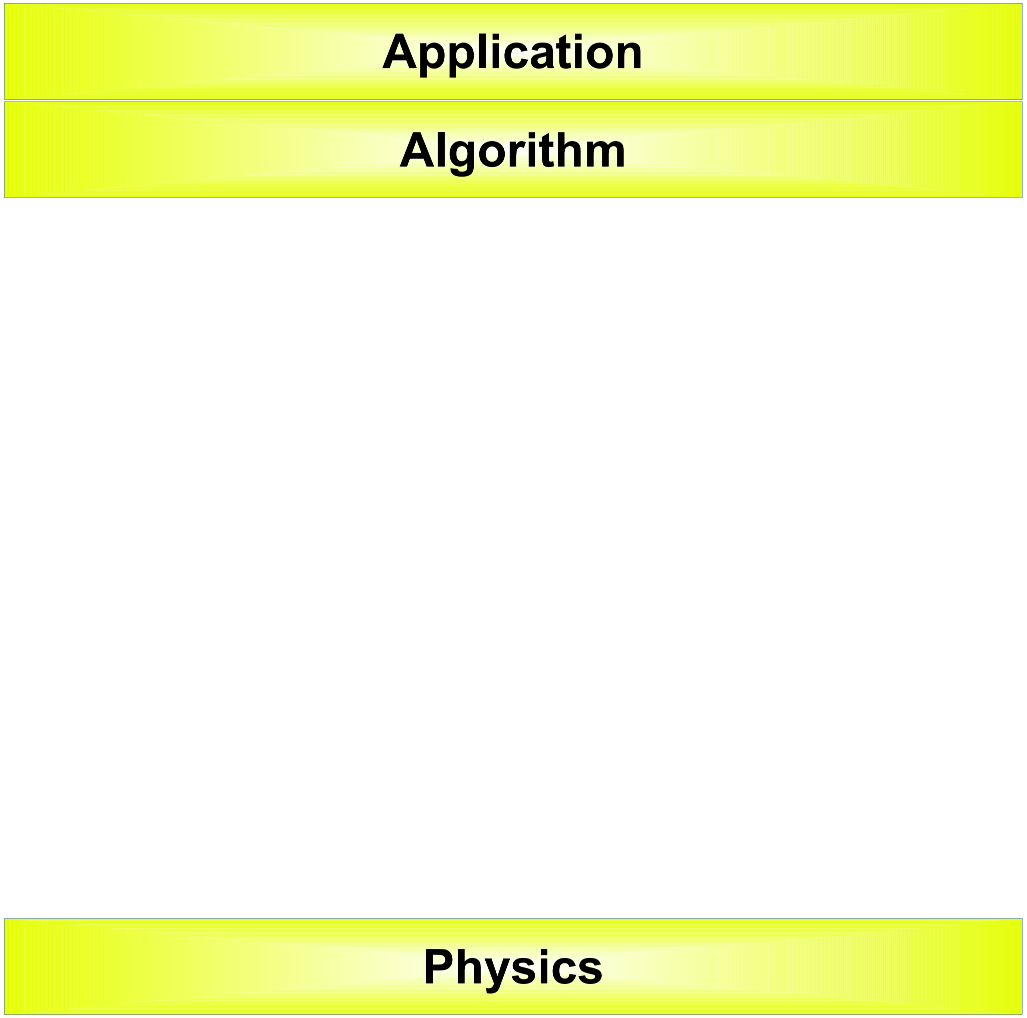
The Compute Stack



Application

Physics

The Compute Stack

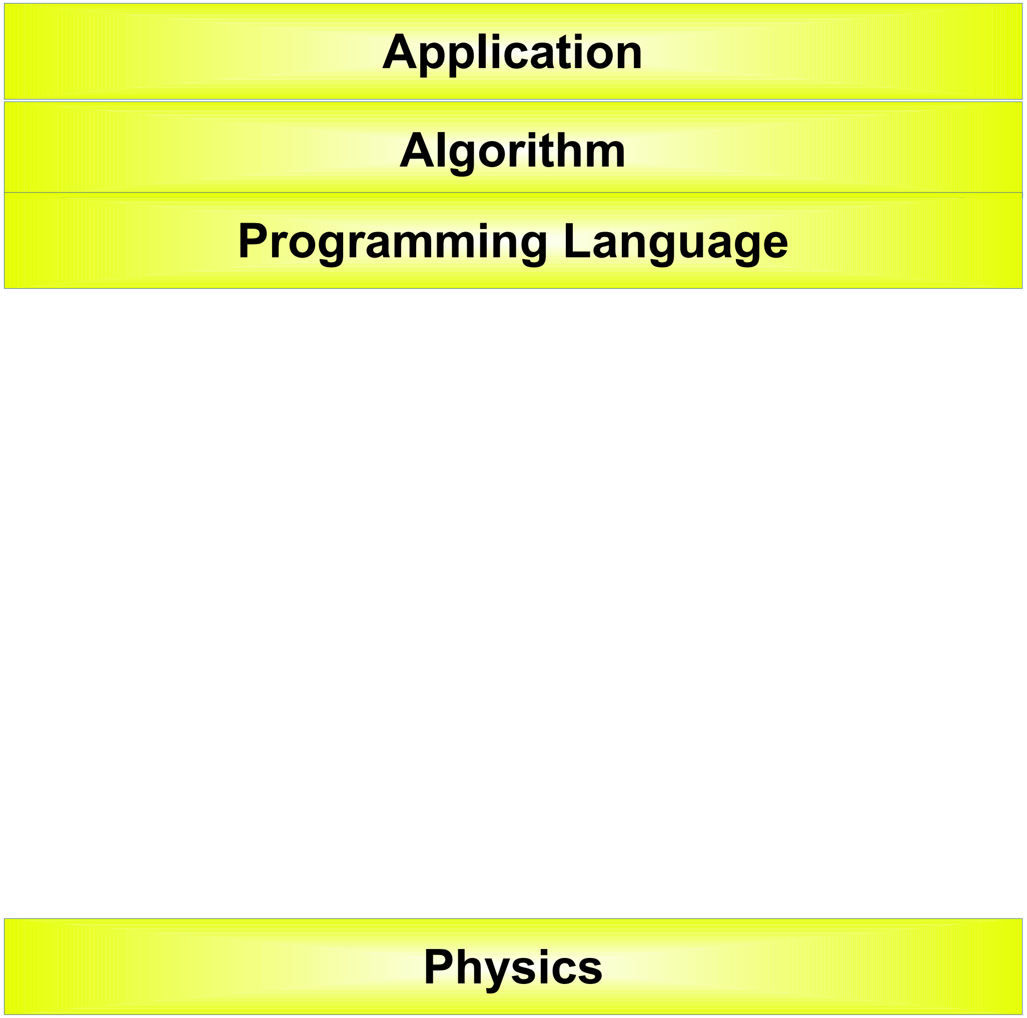


Application

Algorithm

Physics

The Compute Stack



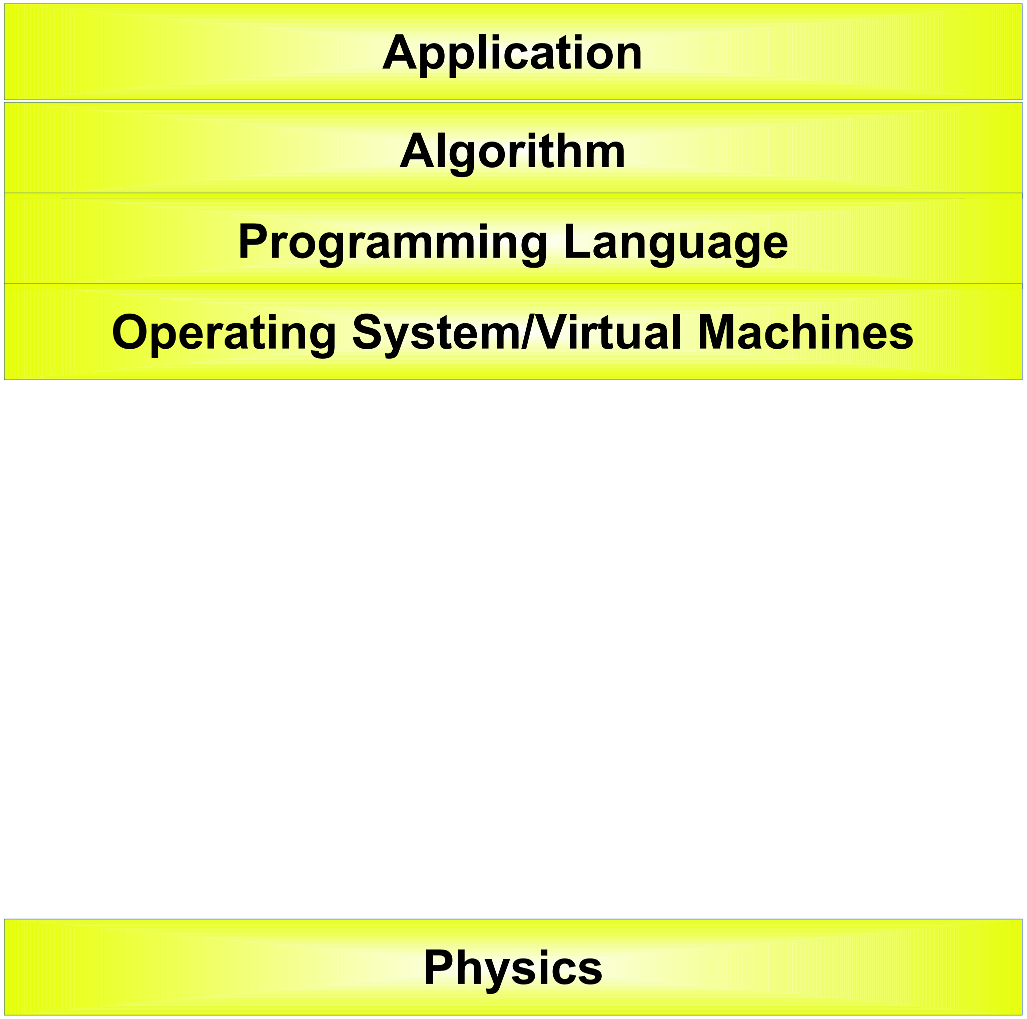
Application

Algorithm

Programming Language

Physics

The Compute Stack



Application

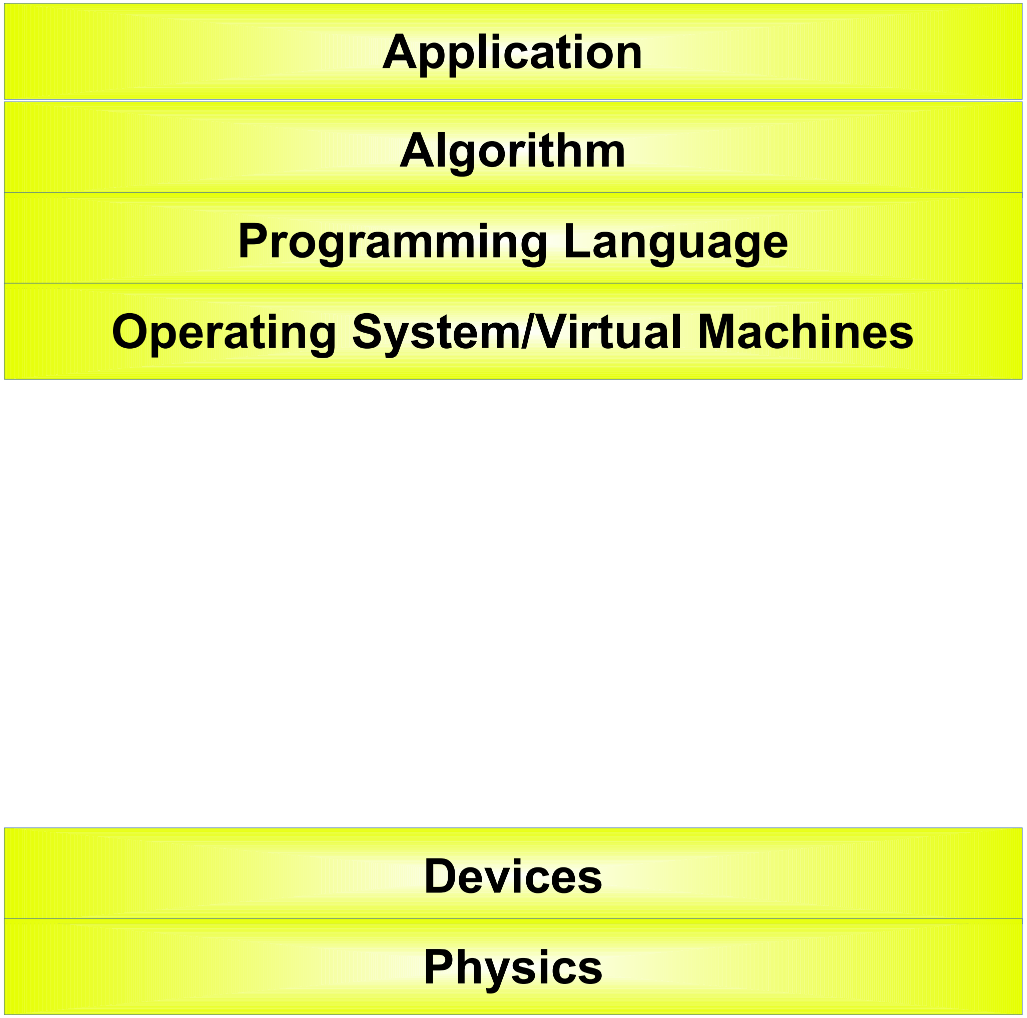
Algorithm

Programming Language

Operating System/Virtual Machines

Physics

The Compute Stack



Application

Algorithm

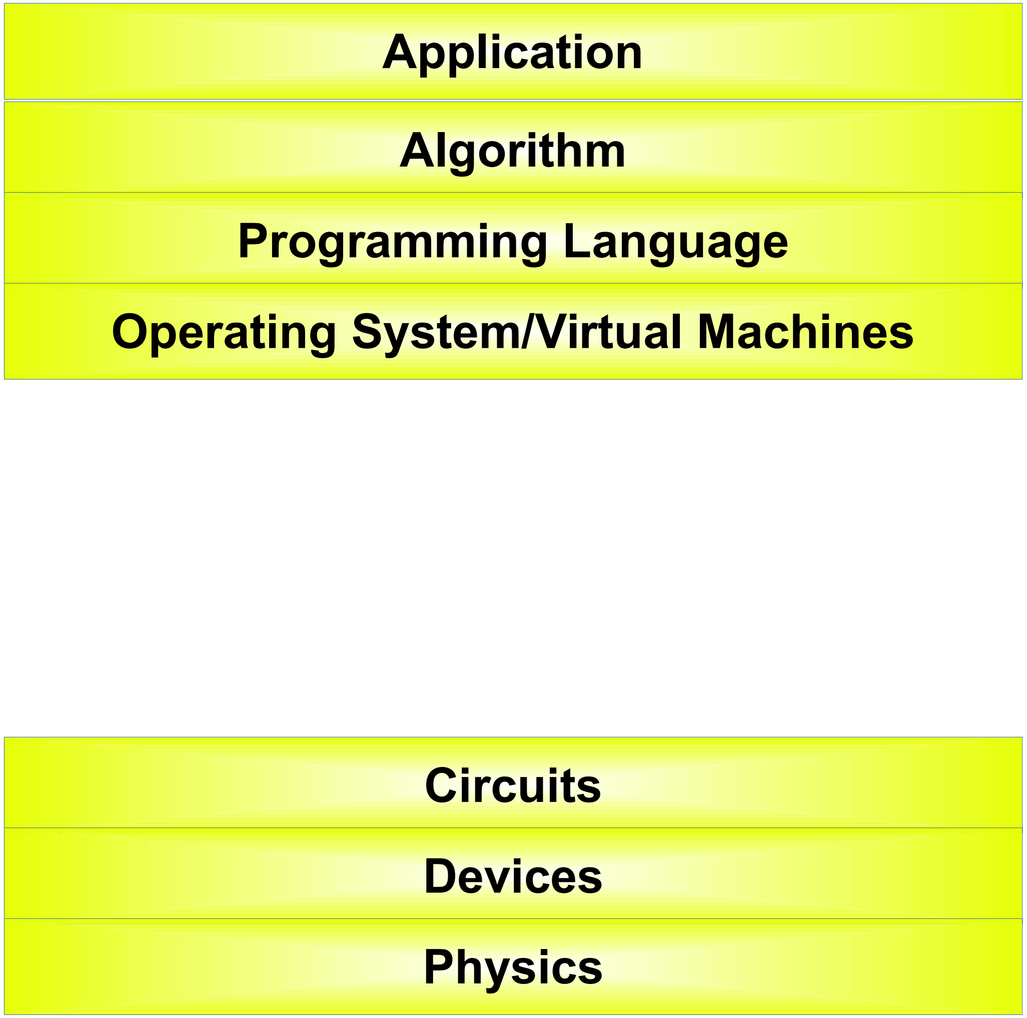
Programming Language

Operating System/Virtual Machines

Devices

Physics

The Compute Stack



Application

Algorithm

Programming Language

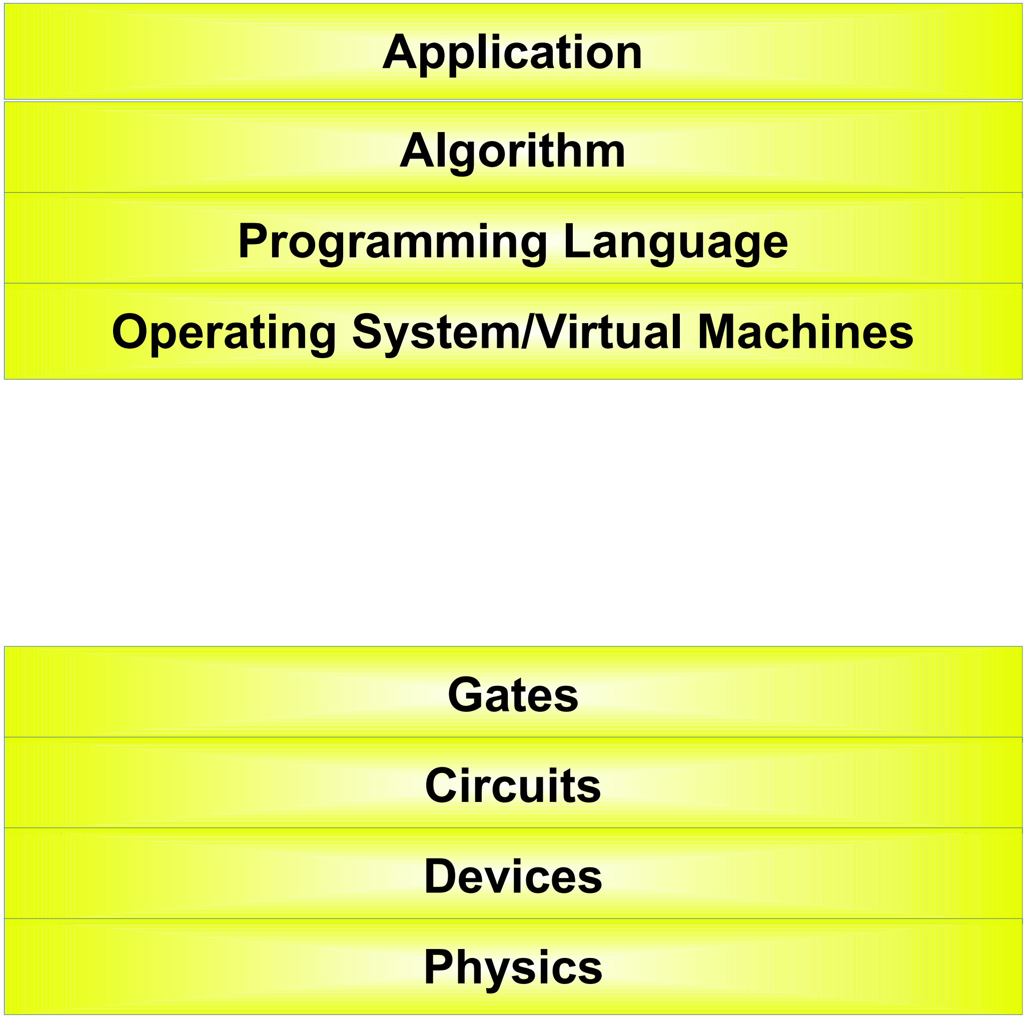
Operating System/Virtual Machines

Circuits

Devices

Physics

The Compute Stack



Application

Algorithm

Programming Language

Operating System/Virtual Machines

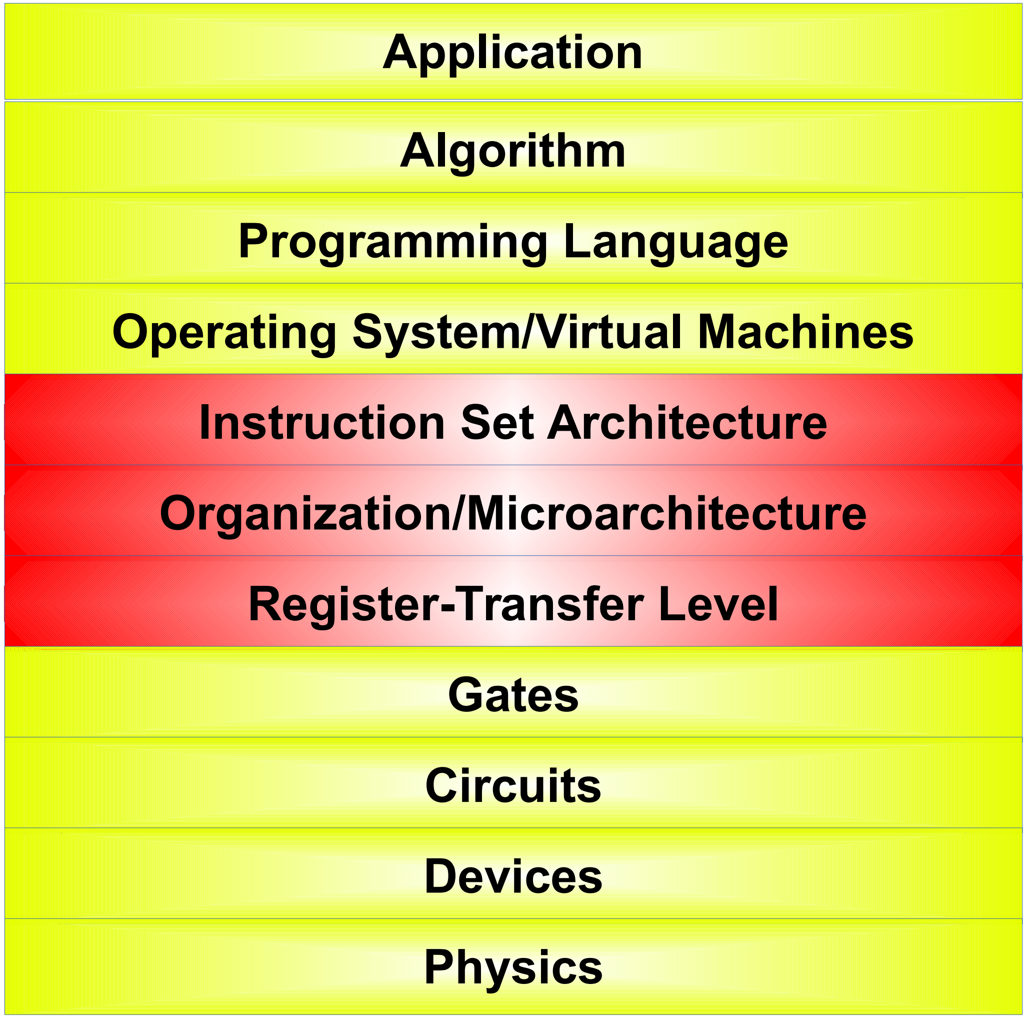
Gates

Circuits

Devices

Physics

The Compute Stack



Application

Algorithm

Programming Language

Operating System/Virtual Machines

Instruction Set Architecture

Organization/Microarchitecture

Register-Transfer Level

Gates

Circuits

Devices

Physics

Memory Detour