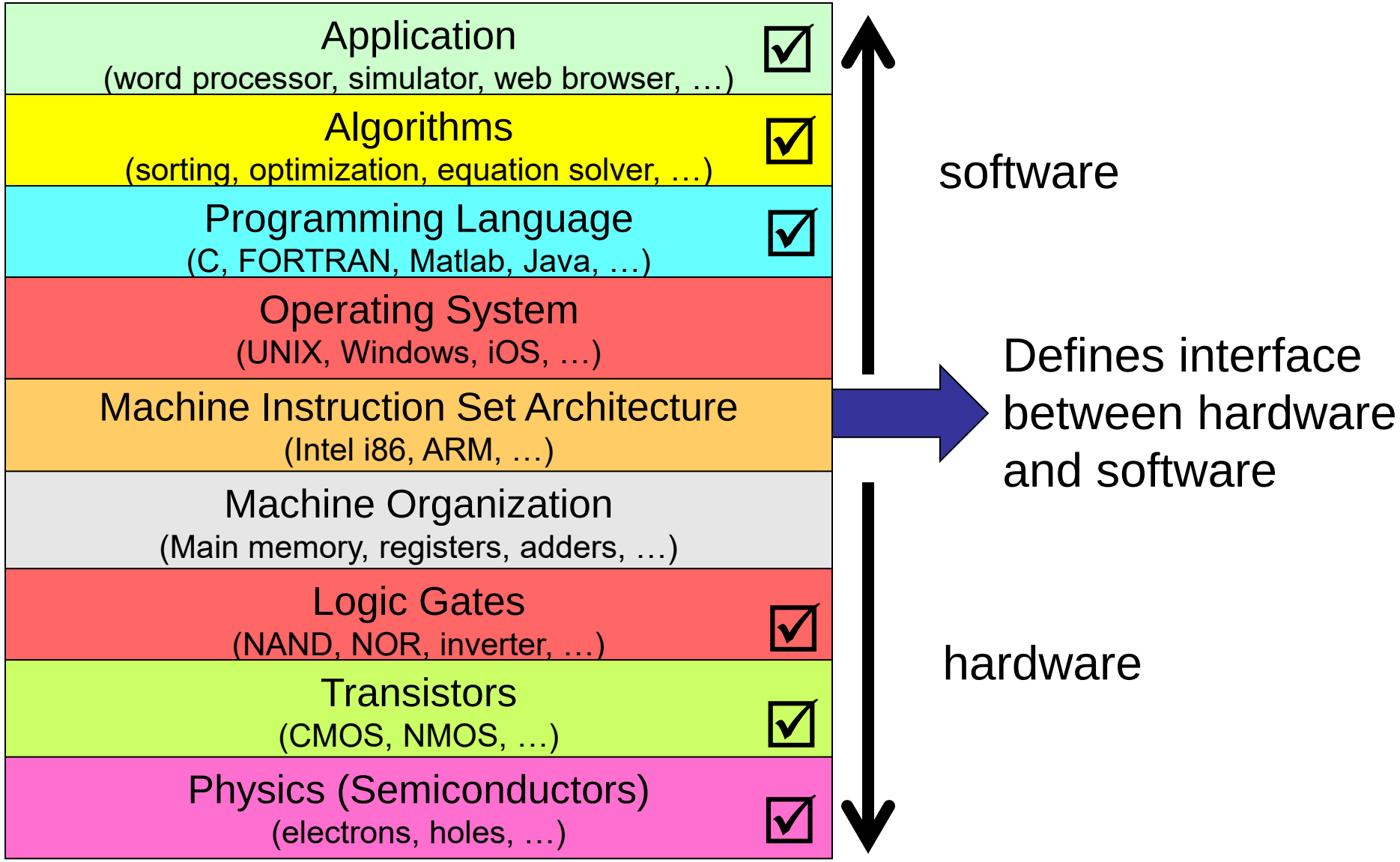
Instruction Set Architecture (Machine Language)

For use in CSE6010 only

Not for distribution

Levels of Abstraction in Computers



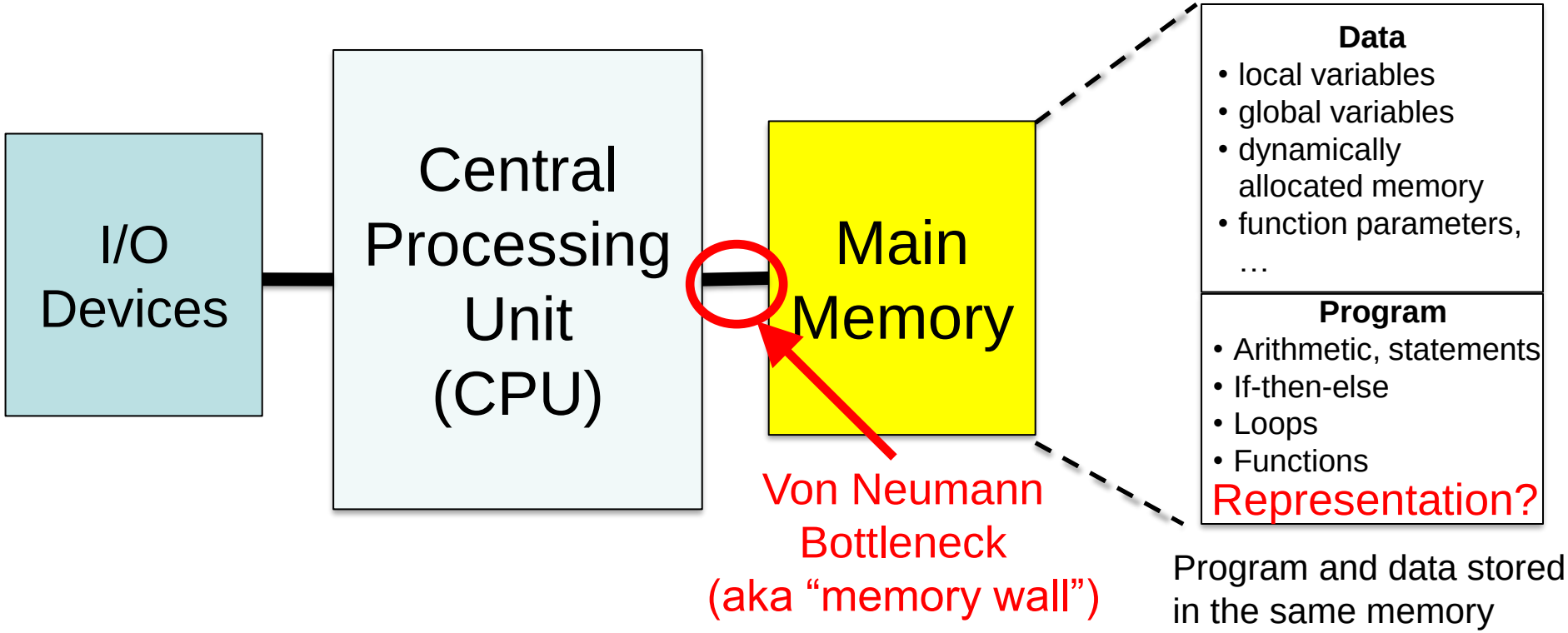
Outline

- Von Neumann Machine Model
  - Central Processing Unit (CPU)
  - Memory
- Instruction Set Architecture
  - Taxonomy
  - Operations and Data
  - Instructions

A Key Idea

- Finite State Machines: function to be performed encoded in the state diagram
  - Operation is “hardwired” into the circuits through the *next state* and *output* functions
- Computer: function to be performed is encoded in a *program* that is stored in the computer’s memory
  - Allows one to change the computer’s functionality without changing its hardware
  - Both the program and data are stored in the same memory on the machine

Von Neumann Machine Model: Main Elements



- Access to “main memory” is slow, relative to CPU speeds (10x to 100x slower than access to memory within the CPU)
- Virtually all computers first load data from main memory into the CPU, operate on the data (e.g., arithmetic) within the CPU, and then store the results back into main memory
  - Has major impact in CPU and instruction set design

Memory: Speed vs. Size

- Tradeoff between memory speed (access time) and size; small memories are fast, large ones are slow
- 1 GHz clock: one clock cycle is $1x10^{-9}$ seconds or 1 nanosecond (ns)

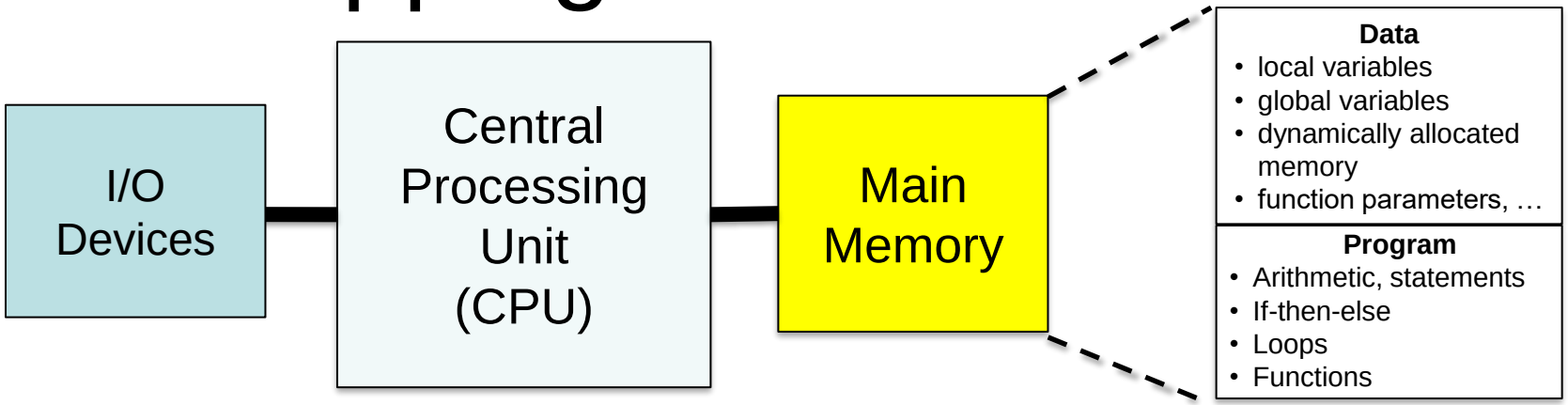
Type of Memory	Typical Size	Typical Access Time	Analogy (human terms)
Single register	32 to 64 bits	< 0.5 ns	~ second
Small register file	32 x 32 bits	< 1.0 ns	
Static RAM	1 – 100 MBytes	~1 ns – 50 ns	~ minute
Dynamic RAM	1 – 100 GBytes	~50 ns – 150 ns	
Disk / Flash Memory	100 – 10,000 GBytes	~1–10 ms (1M – 10M ns)	~ month

- Here: only single registers and small register files are used within the CPU
  - Access a single register or small register file in a single clock cycle
  - Access to main memory requires many clock cycles

Word and Word Length

- Each computer is designed to operate on a unit of memory called a ”word”
  - Typical word size today: 64 bits (we typically say the machine is a “64-bit architecture” to indicate the word size)
  - Usually, ALU operates on ”word” sized data
  - Memory addresses are typically one word
  - Machine instructions are often one word
- Computer can operate on other sizes of data, but more “effort” is required
  - Data less than a word, e.g., byte, often involves extracting what is desired from a word, and shifting to align data
  - Data more than a word, e.g., long double, often involves processing one word at a time

Mapping C to the Machine



- C: Data Types
  - `int, short, long`
  - `float, double`
  - `char`
  - `*int` (pointers)
  - arrays, `struct`
- C: Program instructions
  - Arithmetic, logic operations
  - If-then-else, loops, function call/return

compiler

- Machine Data Types
  - Byte
  - Word
  - Half-word
  - Double-word
  - Floating point
  - Address (word)
- Machine Instructions
  - Arithmetic, logic
  - Branches, jumps, subroutine call/return

Machine Language

To a human the program looks like this (C code):

```
int i, j, k;

main()
{
        i = 1;
        j = i+1;
        k = j + 2;
}
```

compiler

Assembler program: human readable version of machine code

```
        .section        __TEXT,__text,regular,pure_instr
        .globl  _main
        .align  4, 0x90
_main:
Leh_func_begin1:
        pushq   %rbp
Ltmp0:
        movq    %rsp, %rbp
Ltmp1:
        movq    _i@GOTPCREL(%rip), %rax
        movl    $1, (%rax)
        movl    (%rax), %eax
        addl    $1, %eax
        movq    _j@GOTPCREL(%rip), %rcx
        movl    %eax, (%rcx)
        movl    (%rcx), %eax
        addl    $2, %eax
        movq    _k@GOTPCREL(%rip), %rcx
        movl    %eax, (%rcx)
        movl    -4(%rbp), %eax
        popq    %rbp
        ret
Leh_func_end1:

        .comm   _i,4,2
        .comm   _j,4,2
        .comm   _k,4,2
        .section        __TEXT,__eh_frame,coalesced,no_t
EH_frame0:
Lsection_eh_frame:
Leh_frame_common:
Lset0 = Leh_frame_common_end-Leh_frame_common_begin
        .long   Lset0
```

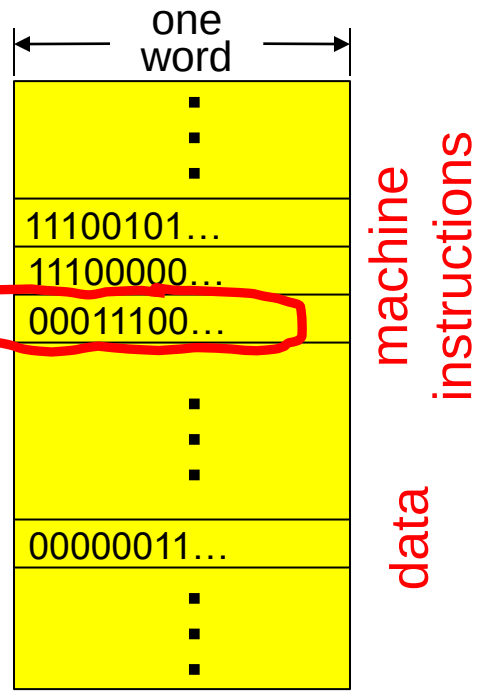
Machine instruction (typically 1 per word)

Machine instruction (assembler)

Storage for variables

assembler

To the hardware the program looks like this (machine code):



On unix systems, compile with –S flag (`% cc -S test.c`) to see assembler

Data and instructions cannot be distinguished in memory (it's all just 0s and 1s)!

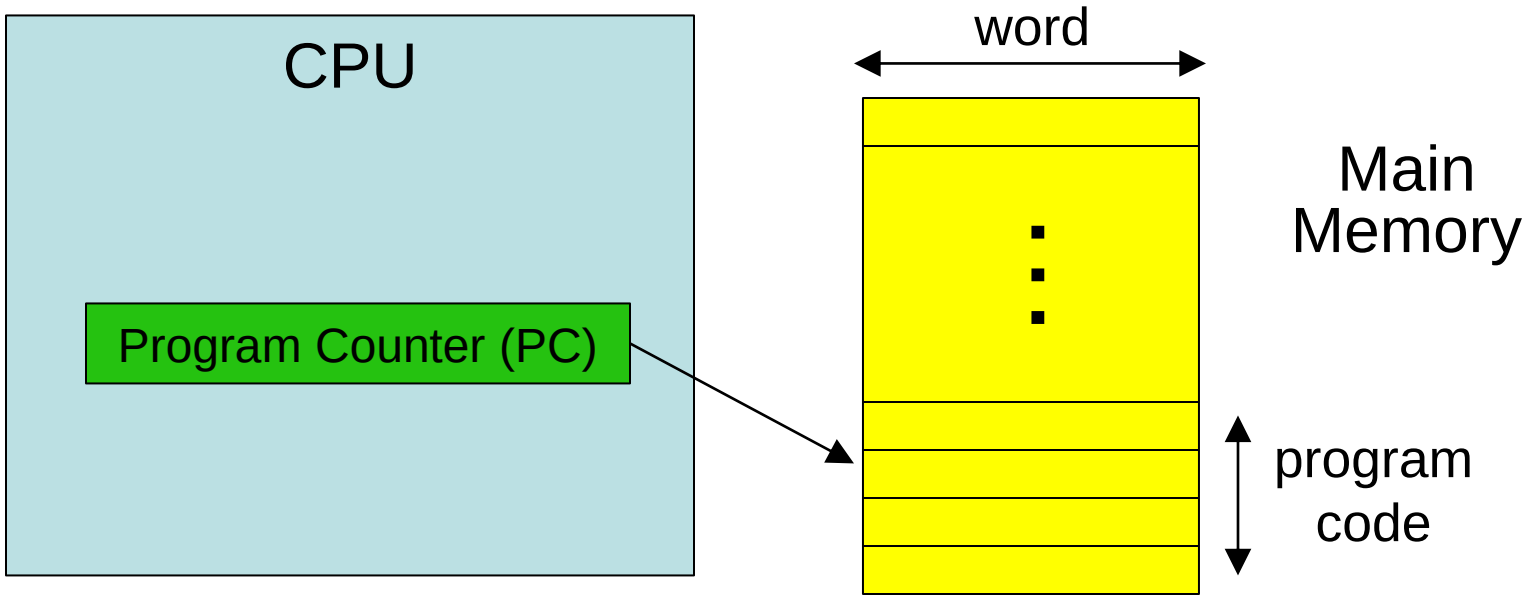
Outline

- Von Neumann Machine Model
  - Central Processing Unit (CPU)
  - Memory
- Instruction Set Architecture
  - Taxonomy
  - Operations and Data
  - Instructions

Instruction Set Architecture

- Languages like C are too complicated for machines to execute directly
- Machines can only execute a very simple set of instructions called the machine's instruction set architecture (ISA)
  - ISA defines all aspects of the CPU visible to the programmer/compiler
  - Different CPUs have different ISAs (Intel, ARM, etc.)

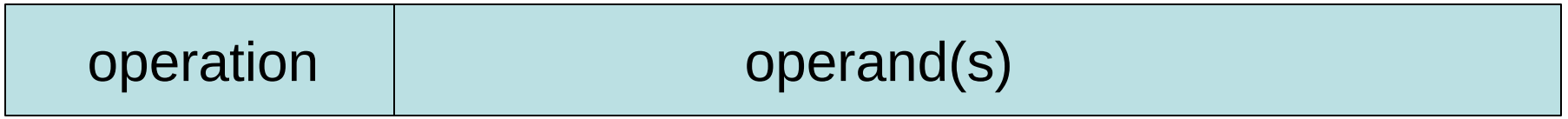
ISA Elements & the Program Counter



- ISA defines key aspects (parameters) of the machine
  - Number of bits in key data types (e.g., word)
  - Number of bits in an address (determines maximum program size)
- Defines machine instructions the CPU can execute
  - Most machine instructions fit into a single word
- CPU registers/memory visible to the programmer
  - Program Counter (PC): address of the next instruction to be executed

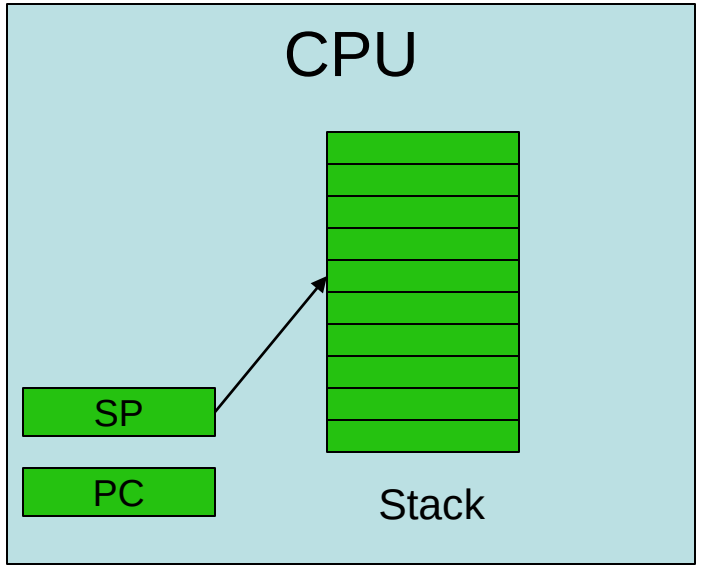
Machine Instructions

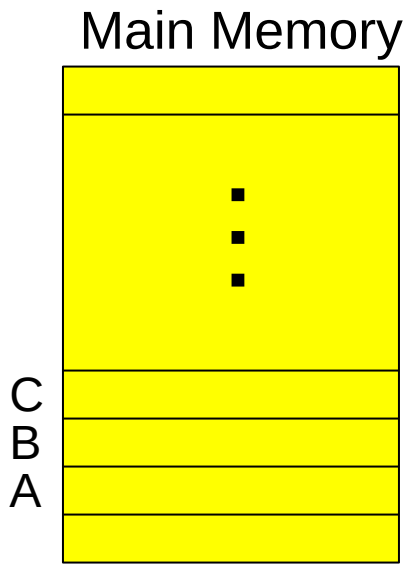
Machine instruction:



- Each machine instruction contains
  - An operation field (e.g., ADD)
  - Zero or more operand(s) (address fields)
- Major types of instruction set architectures
  - Zero address (stack machine)
  - One address (accumulator)
  - Two address (register oriented)
  - Three address (register oriented)
- Register oriented architectures most common today

Zero Address: Stack Machine





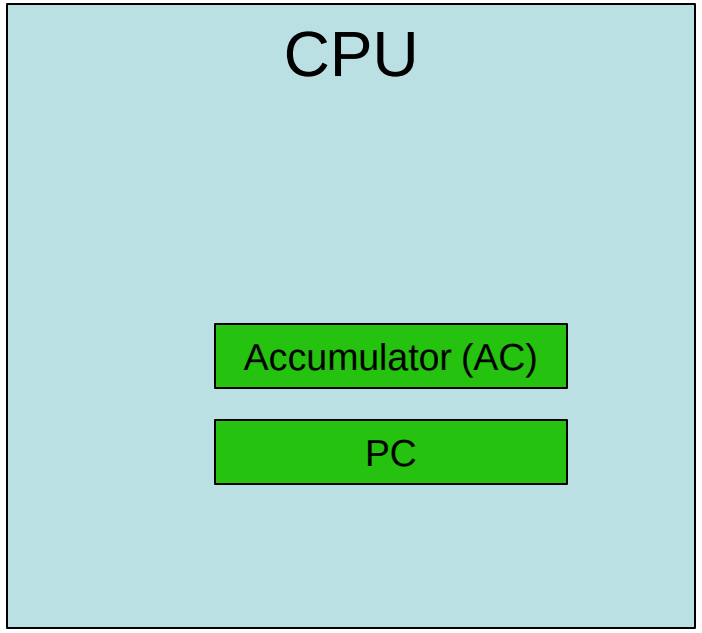
Notation:
M[A]: data stored in memory location A

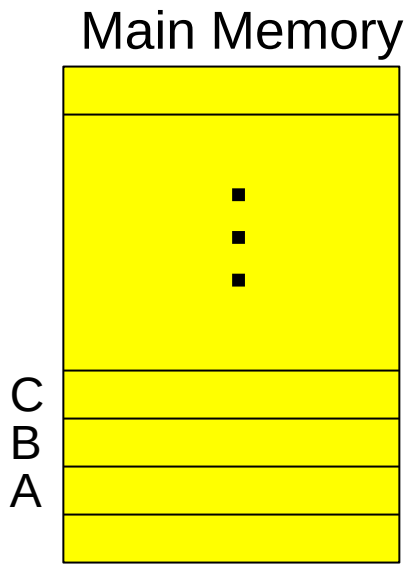
- CPU includes registers organized as a stack to hold operands, results
- SP (stack pointer) register indicates top of stack
- Not to be confused with the *runtime stack* used for functions (main memory)

Example: A = B + C

```
PUSH B // push M[B]
PUSH C // push M[C]
ADD    // pop,pop,add,push
POP A  // M[A]<-pop
```

One Address: Accumulator Machine

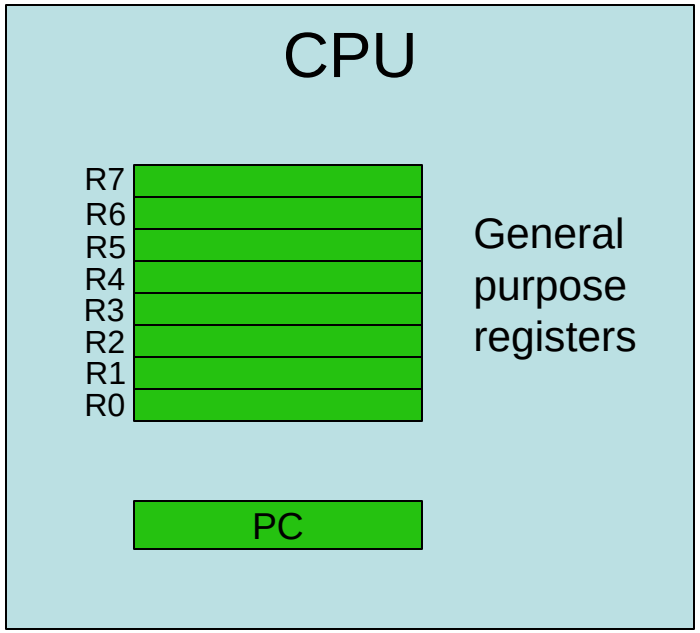


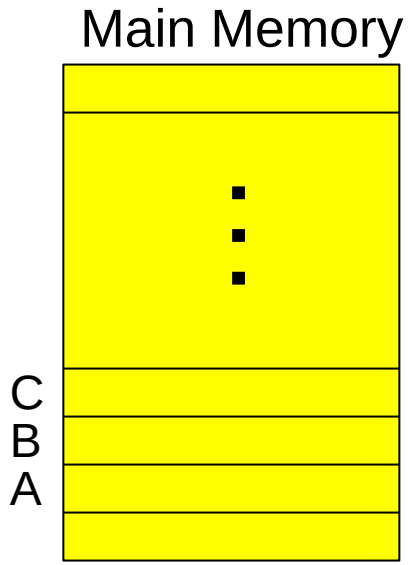


- CPU includes one register called the accumulator (AC)
- AC holds one operand and the result for arithmetic instructions

```
Example: A = B + C
LOAD B   // AC<-M[B]
ADD C    // AC<-AC+M[C]
STORE A  // M[A]<-AC
```

Two/Three Address Architecture (Register-oriented)





- CPU includes a small number of registers (e.g., R0 … R7)
- Operands and result stored in registers

Example: A = B + C

```
LOAD R1,B     // R1<-M[B]
LOAD R2,C     // R2<-M[C]
ADD R3,R1,R2  // R3<-R1+R2
STORE R3,A    // M[A]<-R3
```

Other ISA Categorizations

- "Complexity" of instruction set
  - Reduced Instruction Set Computer (RISC)
    - ARM
  - Complex Instruction Set Computer (CISC)
    - Intel x86
- Memory Access (register-oriented architectures)
  - Load/Store architecture: LOAD and STORE instructions are only ones that access data in memory; operands for other instructions are CPU registers
  - Memory-to-memory architecture: many/most instructions can have operands and the result in memory
- Instruction set design involves determining what computations occur most often in "typical" programs and implementing them efficiently

Instructions: Operations

- A number of operations have similar counterparts as part of a programming language and as part of the machine instruction set. Can you name some of these? (See what you can come up with!)

Instructions: Operations

Programming Language

- Assignment

```
temp = x;
```

- Arithmetic

```
result = x+y;
```

- Conditionals

```
if (a==b) {…}
else {…}
```

- Loops

```
for(i=0;i<n;i++){…}
```

- Function calls, return

```
swap(int *a, int *b)
return (…);
```

Machine Instruction Set

- Data movement

```
LDR R1,R2,offset
```

- Arithmetic

```
ADD R4,R5,R6
```

- Conditionals

Condition codes + branch

```
BRz offset
```

- Loops

Implement w/ conditionals

- Subroutine call, return

```
JSR offset
RET
```

Accessing Data

Programming Language

- Global variables

```
int A;
```

- Local variables and function parameters

```
{int i; …}
```

- Arrays

```
double A[100];
```

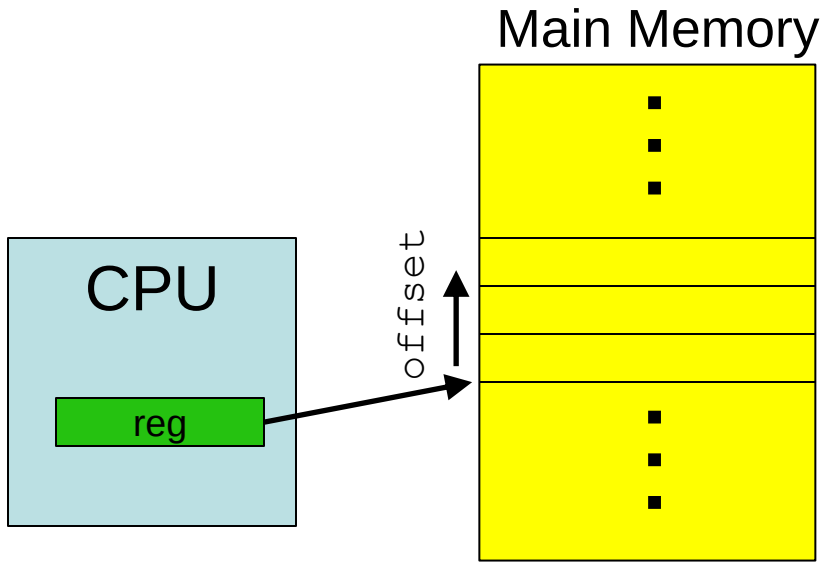
- Structures

```
struct node {
  int data;
  struct node *next;
};
```

Machine Instruction Set

One mechanism sufficient

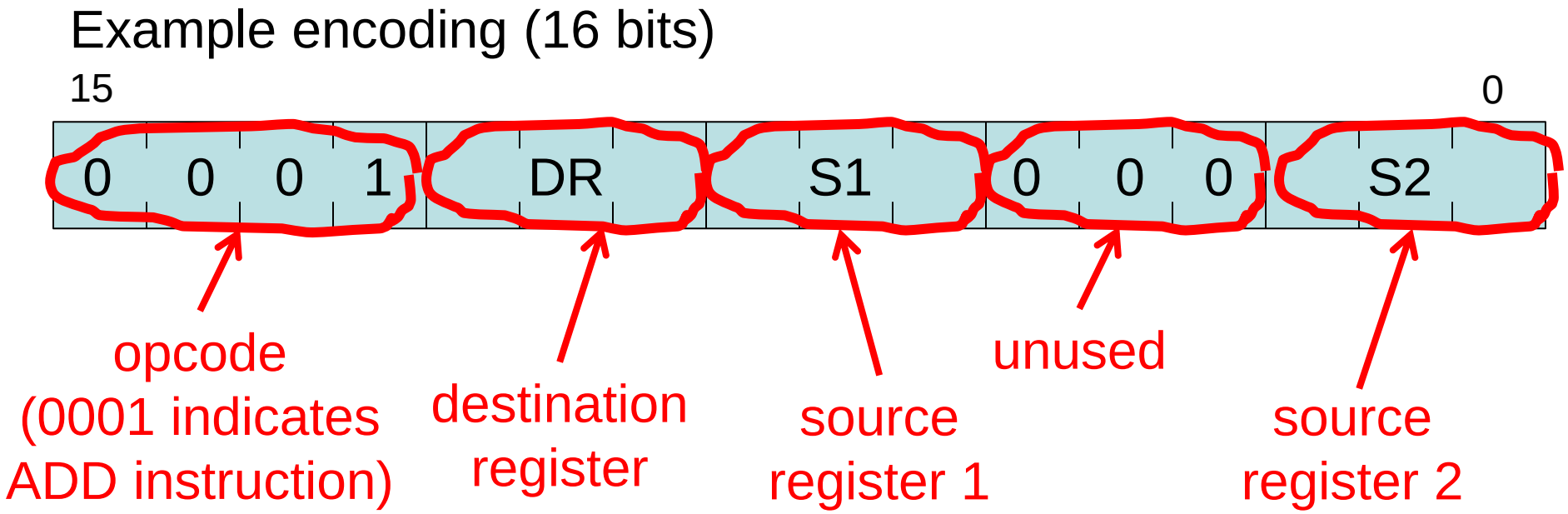
- (1) compute memory address
  (2) read/write memory
- ex: `addr=reg+offset`
  - `offset` field in instruction



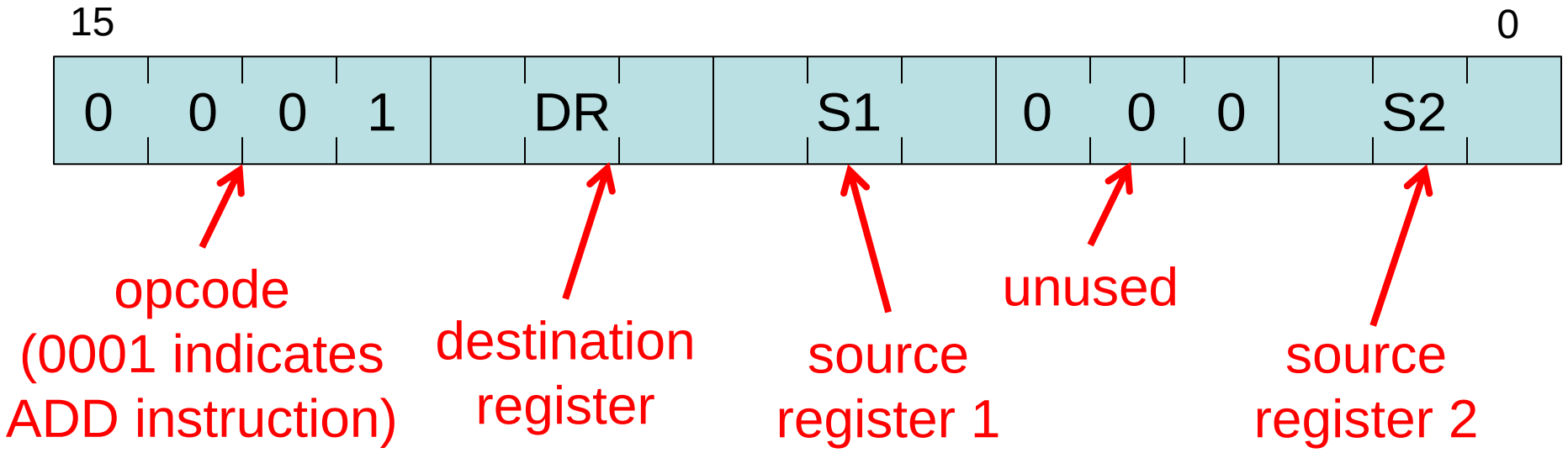
Instructions

An instruction is the most basic unit of computer processing and includes

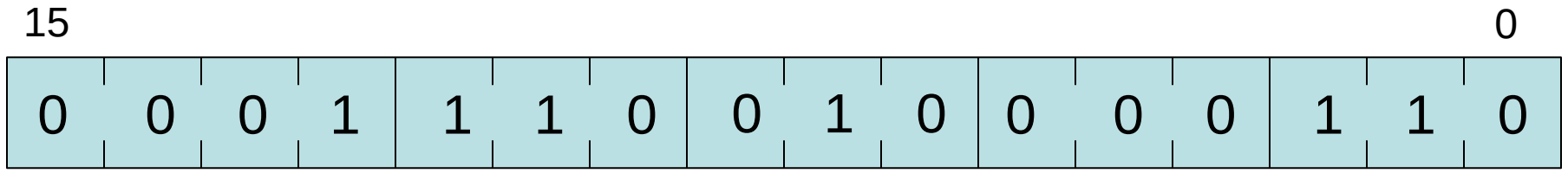
- The opcode: what the instruction does
- The operands: what the instruction is done to



Instructions

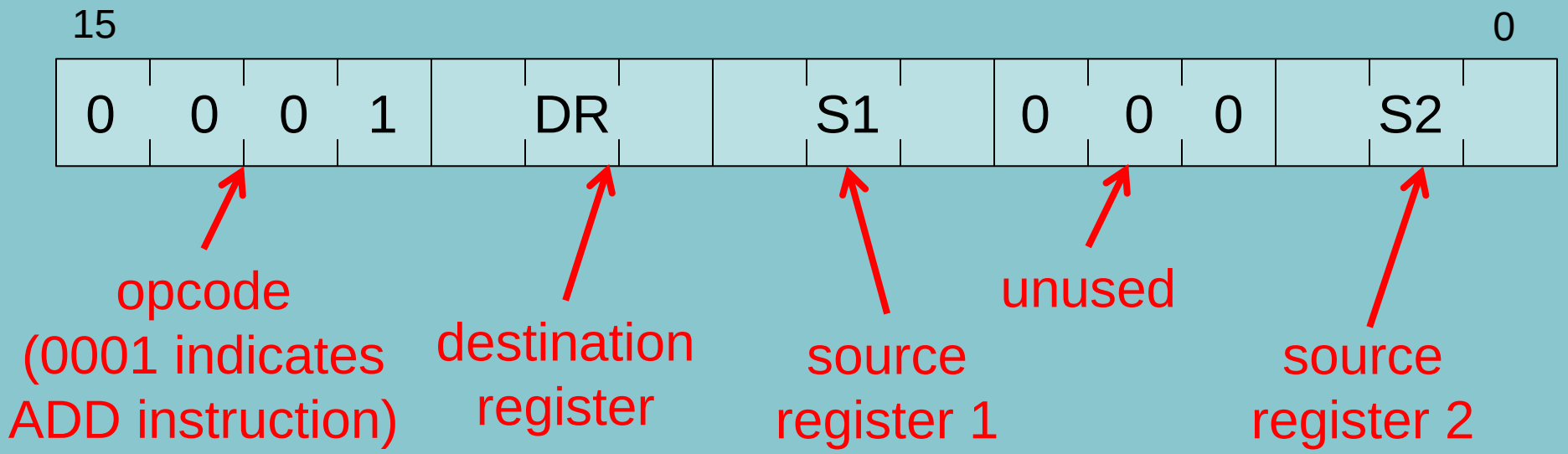


Example:



- Bits [15:12] specify opcode of 1: ADD
- Bits [11:9] specify DR of 6: result will be stored in R6
- Bits [8:6] specify SR1 of 2: operand 1 will be located in R2
- Bits [2:0] specify SR2 of 6: operand 2 will be located in R6

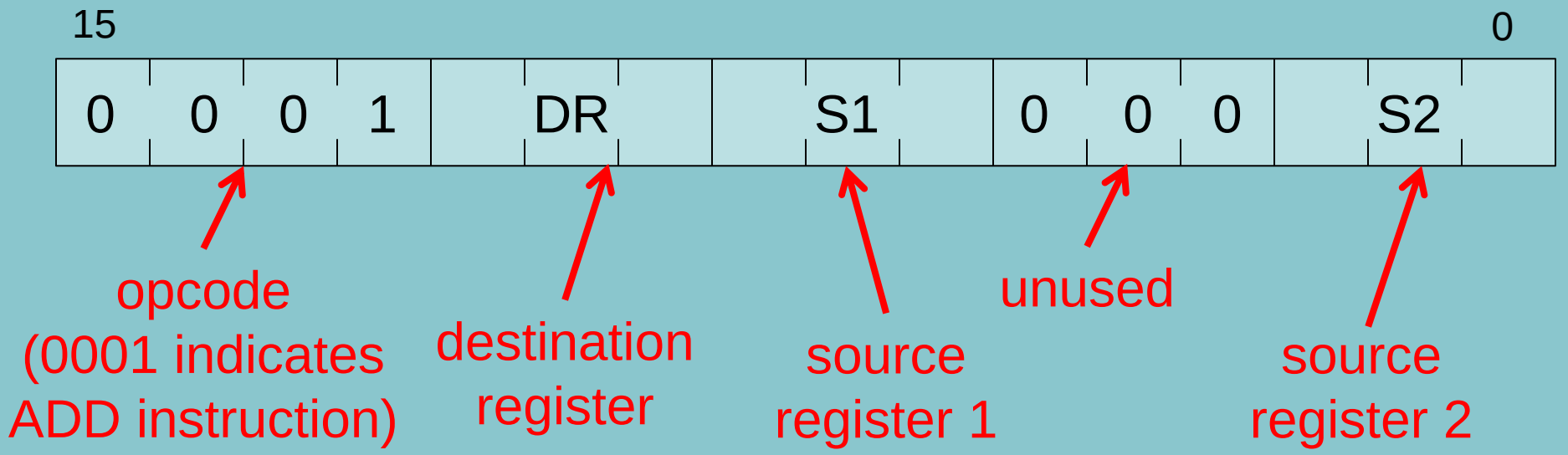
Instructions



For an instruction with this format:

- What is the maximum number of different opcodes?
- What is the maximum number of available registers for placing operands and results?

Instructions



For an instruction with this format:

- What is the maximum number of different opcodes?
  There are 4 bits for the opcode, so there can be at most 2^4=16 different opcodes.
- What is the maximum number of available registers for placing operands and results?
  There are 3 bits for each register, so there can be at most 2^3=8 registers.

Instruction cycle

- Instructions proceed through a sequence of steps:
  - FETCH: Memory's address register loaded with contents of PC and PC is incremented, memory interrogated and instruction placed in memory data register, then loaded into instruction register
  - DECODE: Instruction decoded to identify operation to perform and operands
  - EVALUATE ADDRESS: Address of memory location that may be needed to perform the instruction is computed
  - FETCH OPERANDS: Needed operands are obtained
  - EXECUTE: Instruction is executed (e.g., addition)
  - STORE RESULT: Result written to destination
- Not all instructions use all steps, but all use steps 1-2

Summary

- Von Neumann architecture dominates, but suffers from von Neumann bottleneck
- Program and data stored in memory, but must be brought into the CPU to be operated upon
  - CPU registers reduce number of memory operations
- Machine instruction set architecture defines what aspects of the hardware are visible to the programmer/compiler
- More on instructions to come!