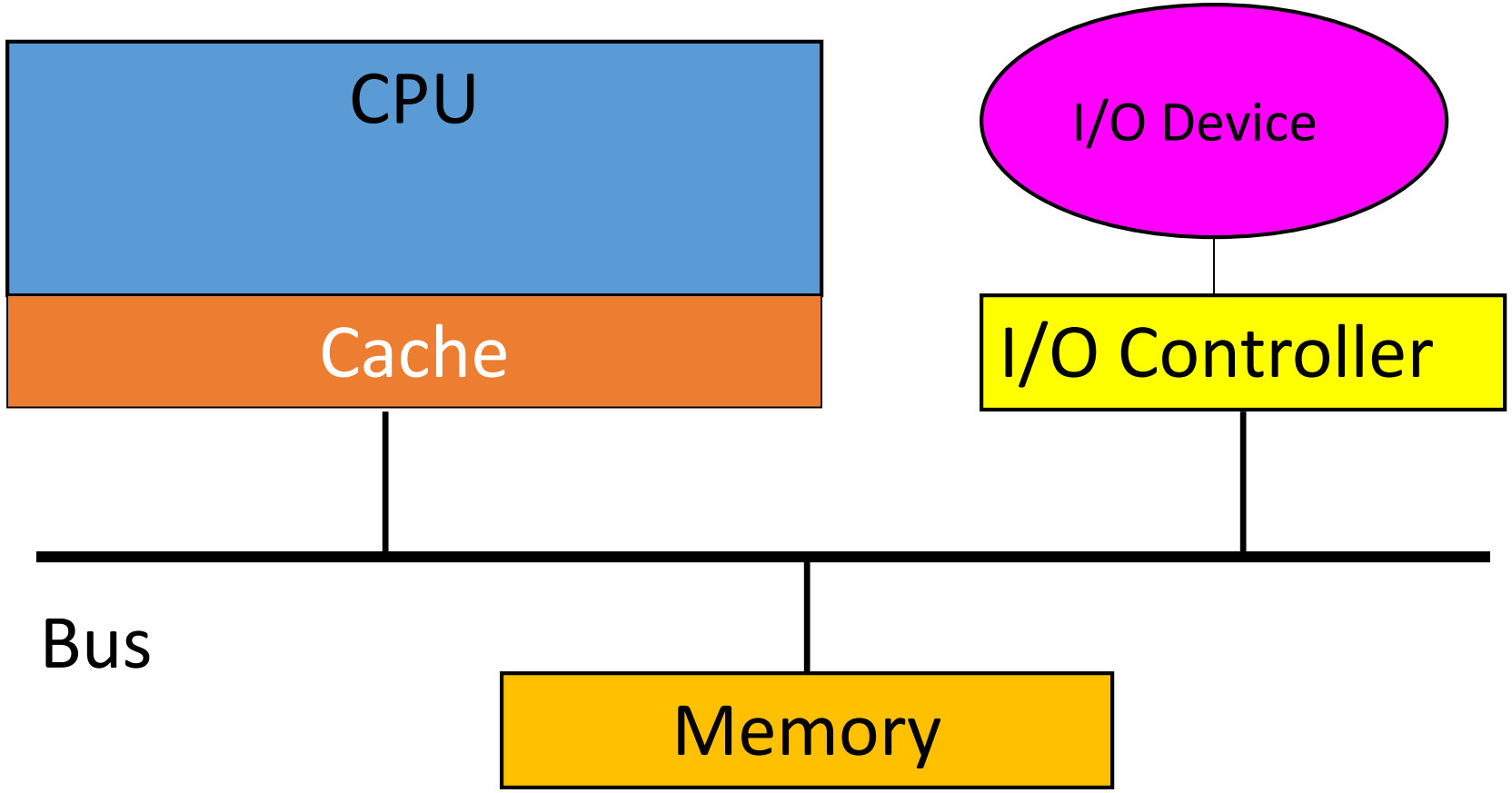
# Computer Architecture

COMP360

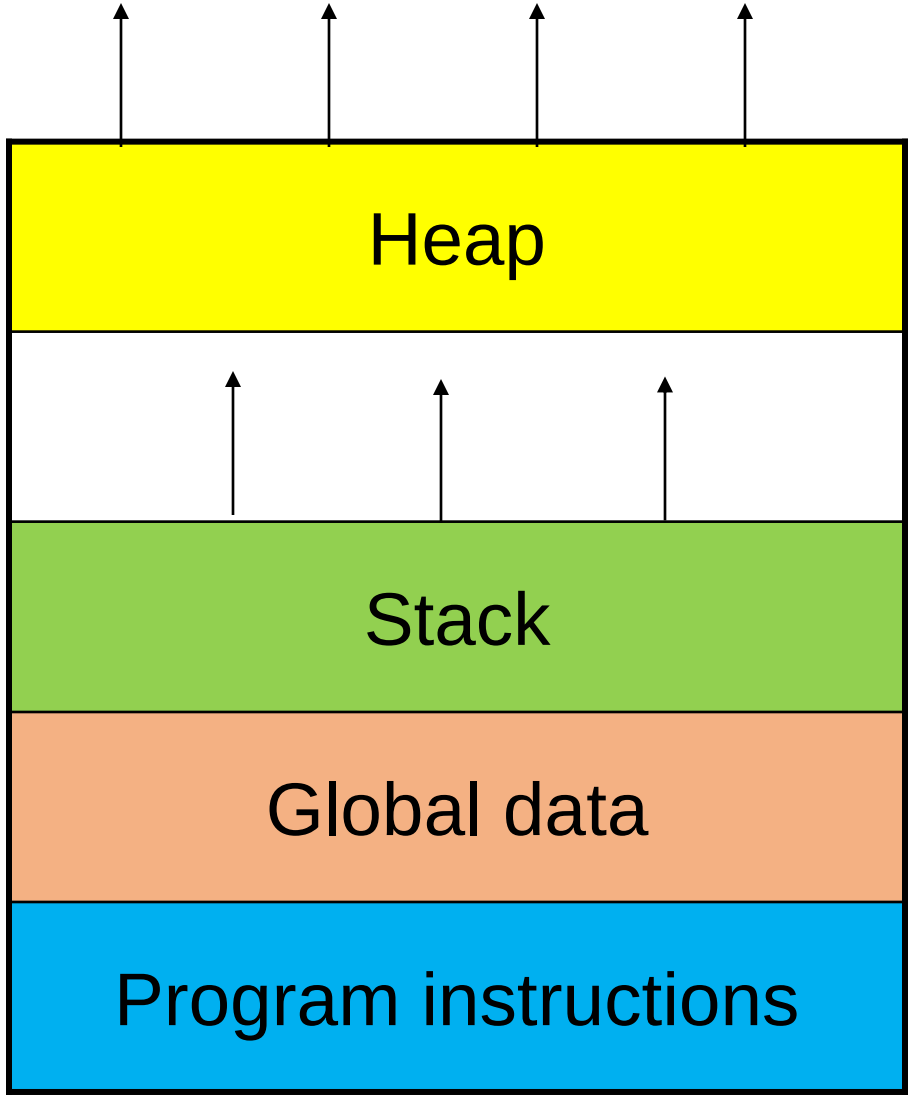
*“It’s hardware that makes a machine fast. It's software that makes a fast machine slow.”*

Craig Bruce

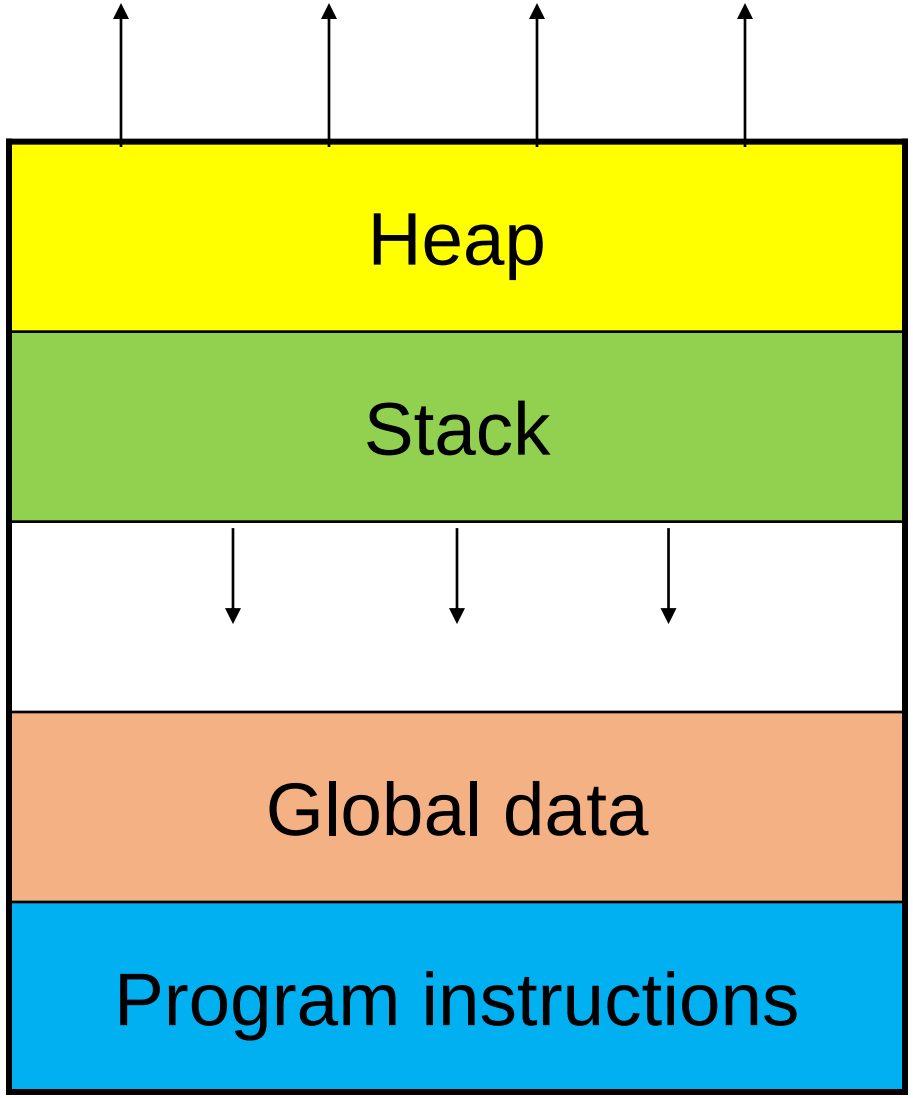
# Basic Computer Components



# Logical Program Memory Organization



# Logical Program Memory Organization




Intel method

# Logical Memory Use

- **Program** – machine language instructions
- **Global** – constants and global data (not in classes or methods)
- **Heap** – dynamically acquired memory. Used when you create a new object
- **Stack** – holds parameters, local variables and method call information

# Hardware Memory Organization

- Some architectures provide mechanisms to separate the different memory types
- Other architectures just support one big piece of memory and let the software use it as necessary

# Parts of CPU

- Program Counter
- User registers
- Instruction register including the address field
- Arithmetic Logic Unit (ALU) has circuits to do arithmetic
- Control Unit to run the fetch execute cycle

# Fetch Execute Cycle

1. Fetch the instruction from the memory address in the Program Counter register
2. Increment the Program Counter
3. Decode the type of instruction
4. Fetch the operands
5. Execute the instruction
6. Store the results

# Machine Language

- The instructions fetches are machine language
- All other computer languages are translated to machine language for execution by a **compiler**
- Machine language is unique to each architecture

# Simple Instructions

- Machine language instructions are simple
- Move data to and from registers and memory
- Do arithmetic with two operands
- “Go To” is the only control structure

# Imaginary Computer

- The computer has 16 registers labeled R0 to R15
- All numbers are integers

Instructions

- Load – move a word from memory to a register
- Store – move a word from a register to memory
- Add, sub, mult, div – do arithmetic
- jump – go to a new location
- jumpXX – go to a location based on a comparison

# Load & Store Instructions

- These two instructions move data between the registers and the memory
- You can load a register with a constant

```
Load    R2, dog
Store   R2, cow
Load    R6, 47
```

# Names

- I use animals as variable names
- In assembler, a variable represents a memory location

- When you see an animal name, you know it is just a name I made up to represent a memory location

# Arithmetic Instructions

- The add, subtract, multiple and divide instructions use three registers
- The first two are the input operands and the last is the output

```
add  R2, R5, R12
mult  R3, R5, R3
```

# Jump Instructions

- An unconditional jump goes to a new location

```
rabbit      add  R2, R3, R6
            // some other stuff
            jump      rabbit
```

# Conditional Jump

- A conditional jump compares two registers and jumps if the comparison is true
- If the comparison is false, the computer just goes to the next line of the program
- All logical comparisons are supported

```
jumpEQ   R1, R7, someplace
jumpNE   R3, R4, again
jumpGE   R2, R11, proglocation
```

# Whatever Instruction You Need

- This is an imaginary computer, you can imagine it to be whatever you need
- You can make up a new instruction if needed
- New instructions have to work within the rest of the architecture

- Only the load and store instructions access memory

# Example Program

```
cat = dog + cat * cow;
```

```
load    R4, cat
load    R7, cow
load    R12, dog
mult    R4, R7, R8
add     R8, R12, R3
store   R3, cat
```

# Example Program with IF

```
if (cow > dog) {
        cat = cow + dog;
} else {
        cat = cow + goat;
}
```

```
            load    R1, cow
            load    R2, dog
            jumpLE  R1, R2, bigdog
            add     R1, R2, R3
            jump    around
bigdog      load    R4, goat
            add     R1, R4, R3
around      store   R3, cat
```

# Example Program with Loop

```
cat = 9;
while ( cat > 0) {
        goat = goat + cow;
        cat = cat – 1;
}
```

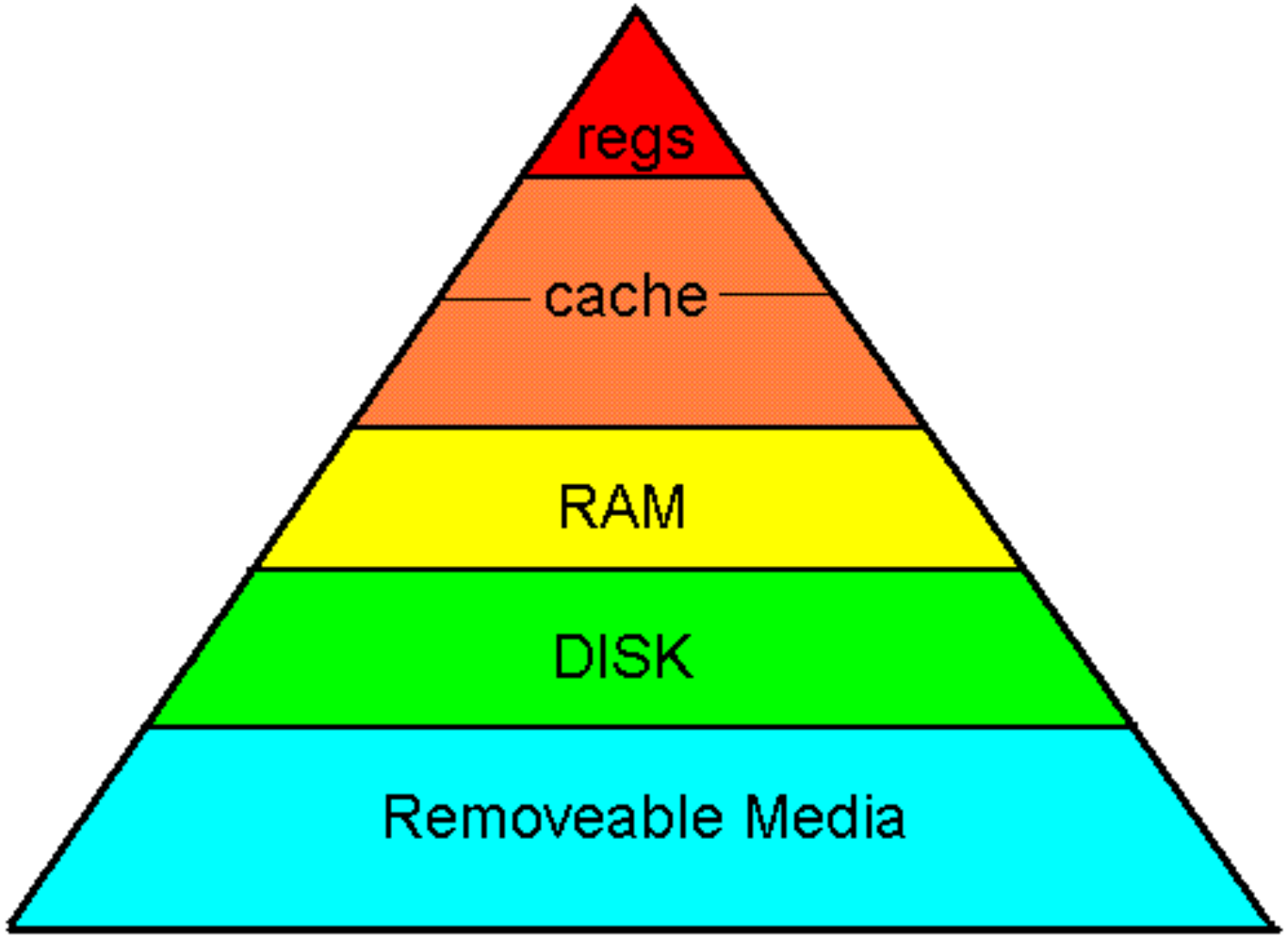
```
        load    R3, 9
        store   R3, cat
        load    R0, 0
        load    R1, 1
more    jumple  R3, R0, done
        load    R7, goat
        load    R5, cow
        add     R7, R5, R9
        store   R9, goat
        add     R3, R1, R3
        store   R3, cat
        jump    more
done
```

# Optimized Example Program with Loop

```
cat = 9;
while ( cat > 0) {
        goat = goat + cow;
        cat = cat – 1;
}
```

```
        load    R3, 9
        load    R0, 0
        load    R1, 1
        load    R7, goat
        load    R5, cow
more    jumple  R3, R0, done
        add     R7, R5, R7
        add     R3, R1, R3
        jump    more
done    store   R3, cat
        store   R7, goat
```

# Memory Hierarchy



# Convert this to assembler

```
cat = 5;
if (dog > cat) {
        cat = cat + goat;
}
```

# Possible Solution

```
cat = 5;
if (dog > cat) {
        cat = cat + goat;
}
```

```
        load    R5, 5
        load    R1, dog
        jumpLE R1, R5, bigcat
        load    R2, goat
        add     R5, R2, R5
bigcat
        store   R5, cat
```

# Compile This to Assembler

```
if (cat > goat) {
        bull = cow;
} else {
        bull = bull + 1;
}
```

# Possible Solution

```
if (cat > goat) {
        bull = cow;
} else {
        bull = bull + 1;
}
```

```
        load   R1, cat
        load   R2, goat
        jumpGT      R1, R2, bigcat
        load   R3, bull
        load   R4, 1
        add    R3, R4, R3
        jump done
bigcat load  R3, cow
done  store R3, bull
```

# Convert to assembler

```
better = number;
do {
   good = better;
   better = (better +
            number/ better) / 2;
}while (good!= better);
```

# Possible Solution

```
better = number;
do {
   good = better;
   better = (better +
              number/ better) / 2;
}while (good!= better);
```

```
// R1=number, R2 = better, R3=good
        load   R1, number
        load   R2, R1
        load   R12, 2
again   load   R3, R2
        div    R1, R2, R5
        add    R5, R2, R5
        div    R5, R12, R2
        jumpNE R3, R2, again
        store  R2, better
```

# Reading

- In the next class we will discuss parameter passing

Read

- textbook sections 9.1 & 9.2
- https://en.wikipedia.org/wiki/Parameter_(computer_programming)