Outline

- Lab Learnings
  - Box of bits (scalars) \( \rightarrow \) Bunch of boxes
    - Indexed: Arrays
    - Group: structs
    - Indirect: Pointers
  \} each combines with the others
- Where data lives
- Stacks, heaps, buffers & attacks
- More about bits

Perspective

- Scalars
  - Size \( \leq \) word width
  - Meaning \( \Leftarrow \) what process it
    - Arithmetic
    - Comparison/Test
    - Addresses
    - Characters
      - Display/keys
      - Others?
        - Pixels
        - Audio?
        - Geometric shapes

Finite

- Word width
  \( \Rightarrow \) Range/precision

Physical

- Mem
- Storage
- Process

Diagram: CPU

To Page No. .
Lab Learning

Bunch of Boxes

**Array**: sequence of like elements, indexed

- start, next/prev, count, *th element

**Struct**: collection of heterogeneous elements

- select fields, size
- move it, size
- compare
- array of structs
- struct containing array

**Pointer**

- reference to a non-contiguous element
- pointer, object it points to, dereference
- **NULL**

**Roles**

- Uniform handle on non-contiguous objects
- Sharing
- Mutation
- Delayed allocation
where data lives (and for how long)

language level

```c
int a;
extern c;
static int d;

Function foo (int b) {
    int e = a + b;
    return e;
}

main () {
    foo (17, 3);
}
```

"external" variable
- defined here, visible in rest of file and outside.
- declared here but defined elsewhere

"automatic" variable
- local to function
- local copy of call parameter value
- external for external for

Machine level
- logically all data resides in memory
- loaded into proc reg's operate store back to mem

How do language concepts map to machine?
what happens when a function is called?

machine doesn't know language run time sets it up

what happens when a function is called?

machine doesn't know language run time sets it up

stack

static

frame

local

parameters

locals

un-locals

locals

malloca

2^10 - 1

~ 10^3

256K

320K

4MB

8MB

4MB
Attacks on service applications can cause the OS (a program) to do things it shouldn't:

- Write
- Clobber data
  - Pass null strings with no terminator
  - Pass bad length parameters
  - Pass bad ptrs

- Corrupting system data may cause it to do other stupid things.

- Top of stack
  - Clobber pointer variable, clobber ret
  - Cause OS to start executing out of the stack
  - Mistakes are bad, but can be fixed.

- Safe string libraries
  - segments (protection or pointers off address space)
  - Stack checking
Bit more on bits

what happens when you assign a small scalar to a larger one

\textit{signed}
\begin{align*}
\text{long int } i &= \text{short int } j
\end{align*}

\textit{unsigned}
\begin{align*}
\text{int } i &= \text{char } j
\end{align*}

\begin{align*}
\text{zero extend (unsigned)} & \quad \text{sign extend (signed)}
\end{align*}

Prove this is valid

intuitive \quad 2008 \equiv 0_{2008} \equiv 0_{00000000}

formally
\begin{align*}
b_{n-1} \cdot 2^{n-1} + \cdots + b_1 \cdot 2^1 + b_0 &= \theta \cdot 2^{n-1} + \cdots + \theta \cdot 2^1 + b_{n-1} \cdot 2^{n-1} + \cdots + b_0
\end{align*}

Can you prove it works for \textit{signed} numbers?
Recall

\[-b_{n-1}^{n-1} + b_{n-2}^{n-2} + b_{n-3}^{n-3} + \cdots + b_0,\]

Case \(b_{n-1} = 0\) (non-negative)

\[-b_{m-1}^{m-1} + \cdots + b_1^2 + b_0 = (\frac{-2^{m-1-1}}{2^0} + 2^0) \cdot 2^{m-1} \]

\[= -2^{m-n} + 2^{m-n-1} + \cdots + 2^0 - 1\]