

The background of the slide features a photograph of a large, classical-style statue, likely the 'The Spirit of the Law' statue at the University of Illinois. The statue is a woman in a long, flowing robe with her arms outstretched. It is set in a park-like environment with trees and other statues visible in the background. The entire image is overlaid with a semi-transparent red color.

# ISAs and Instruction Sets, File Types, & Memory

**CS 240 - The University of Illinois**

Wade Fagen-Ulmschneider  
September 2, 2021

# Instruction Set Architecture (ISA)

# CPU Registers

Each CPU has a very limited number of \_\_\_\_\_ used for general purpose CPU operations.

<b>Move:</b>	<b>MOV, XCHG, PUSH, POP, ...</b>
<b>Arithmetic (int):</b>	<b>ADD, SUB, MUL, DIV, NEG, CMP, ...</b>
<b>Logic:</b>	<b>AND, OR, XOR, SHR, SHL, ...</b>
<b>Control Flow:</b>	<b>JMP, LOOP, CALL, RET, ...</b>
<b>Synchronization:</b>	<b>LOCK, ...</b>
<b>Floating Point:</b>	<b>FADD, FSUB, FMUL, FDIV, FABS, ...</b>

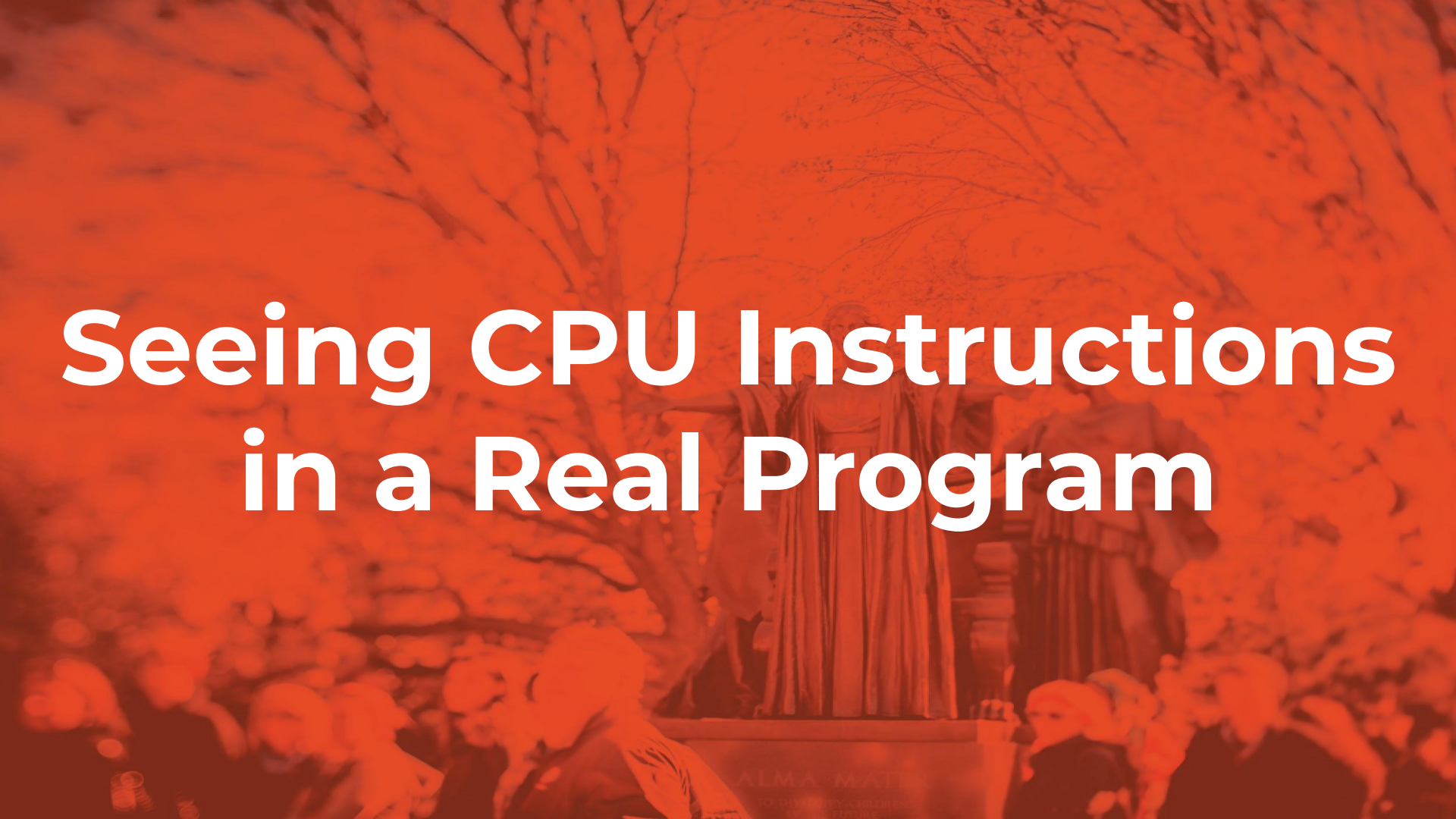
# Instruction Set Size

ARM processors have significantly fewer instructions and known as:

# Instruction Set Size

x86/x64 processors a greater set (more specialized) instructions and know as:

# RISC vs. CISC?

A photograph of a crowd of people gathered around a statue of Alma Mater, overlaid with a semi-transparent orange filter. The statue is in the center, and the crowd is in the foreground. The text is overlaid on the image.

# Seeing CPU Instructions in a Real Program



04.c

```
3 int main() {  
4     int a = 0;  
5     a = a + 3;  
6     a = a - 2;  
7     a = a * 4;  
8     a = a / 2;  
9     a = a * 5;  
10    printf("Hi");  
11    a = a * 479;  
...    . . .
```

# Compiling + Printing Instructions

```
gcc 04.c
```

```
objdump -d ./a.out
```

```
3 int main() {
```

```
f3 0f 1e fa
```

```
55
```

```
48 89 e5
```

```
48 83 ec 10
```

```
endbr64
```

```
push    %rbp
```

```
mov     %rsp,%rbp
```

```
sub     $0x10,%rsp
```

```
4 int a = 0;
```

```
c7 45 fc 00 00 00 00 movl $0x0, -0x4(%rbp)
```

5

```
a = a + 3;
```

83 45 fc 03

```
addl    $0x3, -0x4(%rbp)
```

6 `a = a - 2;`

83 6d fc 02

`subl $0x2, -0x4(%rbp)`

7

```
a = a * 4;
```

c1 65 fc 02

shll \$0x2, -0x4(%rbp)

8

```
a = a / 2;
```



8

**a = a / 2;**

8b 45 fc

89 c2

c1 ea 1f

01 d0

d1 f8

89 45 fc

mov -0x4(%rbp), %eax

mov %eax, %edx

shr \$0x1f, %edx

add %edx, %eax

sar %eax

mov %eax, -0x4(%rbp)

9

```
a = a * 5;
```

```
8b 55 fc  
89 d0  
c1 e0 02  
01 d0  
89 45 fc
```

```
mov    -0x4(%rbp), %edx  
mov    %edx, %eax  
shl   $0x2, %eax  
add   %edx, %eax  
mov   %eax, -0x4(%rbp)
```

```
10 printf("Hi");
```

```
48 8d 3d f0 0d 00 00 lea    0xdf0(%rip),%rdi  
# 2004 <_IO_stdin_used+0x4>
```

```
b8 00 00 00 00      mov    $0x0,%eax  
e8 42 fe ff ff      callq 1060<printf@plt>
```

11

```
a = a * 479;
```

```
8b 45 fc
```

```
69 c0 df 01 00 00
```

```
89 45 fc
```

```
mov    -0x4(%rbp),%eax
```

```
imul  $0x1df,%eax,%eax
```

```
mov    %eax,-0x4(%rbp)
```

# Program Counter

<b>f3</b>	<b>0f</b>	<b>1e</b>	<b>fa</b>				<b>endbr64</b>
<b>55</b>							<b>push %rbp</b>
<b>48</b>	<b>89</b>	<b>e5</b>					<b>mov %rsp,%rbp</b>
<b>48</b>	<b>83</b>	<b>ec</b>	<b>10</b>				<b>sub \$0x10,%rsp</b>
<b>c7</b>	<b>45</b>	<b>fc</b>	<b>00</b>	<b>00</b>	<b>00</b>	<b>00</b>	<b>movl \$0x0,-0x4(%rbp)</b>
<b>83</b>	<b>45</b>	<b>fc</b>	<b>03</b>				<b>addl \$0x3,-0x4(%rbp)</b>
<b>83</b>	<b>6d</b>	<b>fc</b>	<b>02</b>				<b>subl \$0x2,-0x4(%rbp)</b>
<b>c1</b>	<b>65</b>	<b>fc</b>	<b>02</b>				<b>shll \$0x2,-0x4(%rbp)</b>
<b>8b</b>	<b>45</b>	<b>fc</b>					<b>mov -0x4(%rbp),%eax</b>
<b>89</b>	<b>c2</b>						<b>mov %eax,%edx</b>
<b>c1</b>	<b>ea</b>	<b>1f</b>					<b>shr \$0x1f,%edx</b>
<b>01</b>	<b>d0</b>						<b>add %edx,%eax</b>
<b>d1</b>	<b>f8</b>						<b>sar %eax</b>

# Operation Timings

**Latency:** This is the delay that the instruction generates in a dependency chain if the next dependent instruction starts in the same execution unit. The numbers are minimum values. Cache misses, misalignment, and exceptions may increase the clock counts considerably.

Arithmetic instructions												
ADD, SUB	r,r	1	0	1	0	0.25	0/1	alu0/1		86	c	
<i>An ADD with no input carry can be executed in as little as 1 cycle!</i>												
ADC, SBB	r,r/i	3	0	10	0	10	1	int,alu		86		
<i>Addition with input carry (ADC) can take at least 10x longer.</i>												
CMP	r,r	1	0	1	0	0.25	0/1	alu0/1		86	c	
<i>Comparing two values can be executed in as little as 1 cycle.</i>												
MUL, IMUL	r8	1	0	10	0		1	int	mul	86		
<i>Multiplication runs in similar time as ADC.</i>												
DIV	r8/m8	1	20	74	0	34	1	int	fpdiv	86	a	
<i>Best case for division is 74x slower than best case for addition!</i>												



# Endianness:

**Big Idea:** How do we represent multi-byte characters in memory?

```
3 int main() {
4     int i = 3 + (2 << 8) + (1 << 16);
5     char *s = (char *)&i;
6     printf("%02x %02x %02x %02x\n",
7           s[0], s[1], s[2], s[3]);
8     return 0;
}
```

04b.c

s[0]	s[1]	s[2]	s[3]

# Big Endian

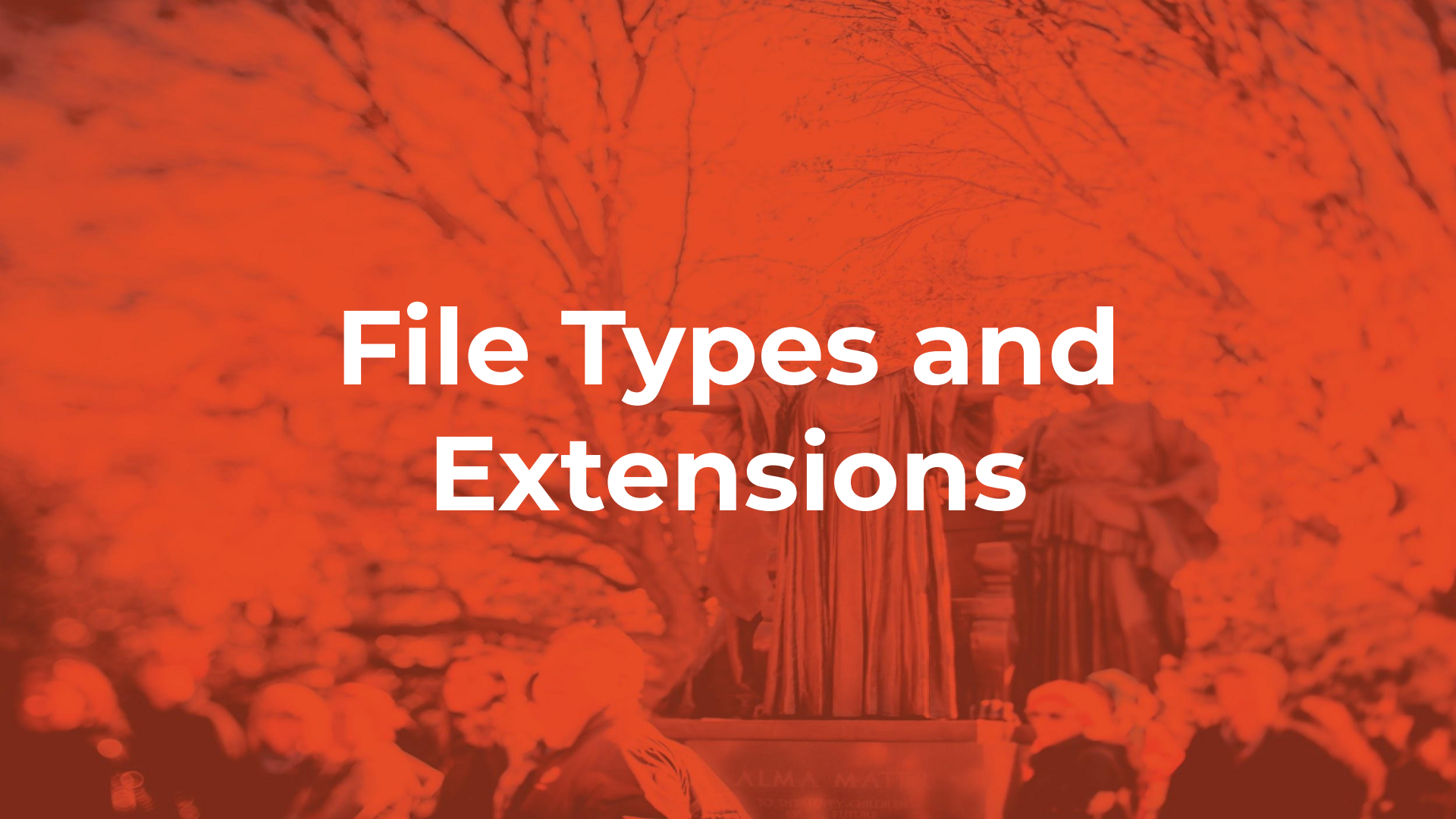
s[0]	s[1]	s[2]	s[3]

# Little Endian

s[0]	s[1]	s[2]	s[3]

# Network Byte Order

# Host Byte Order



# File Types and Extensions

# File Extensions

The most common way to identify the contents of a file is by the **file extension**.

The file extension is defined as:

**cs240.png**

**mp1.c**

**mp1.h**

**taylor.swift.mp4**



# “Plain Text” File

# PNG File Format



## Portable Network Graphics (PNG) Specification (Second Edition)

Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E)

W3C Recommendation 10 November 2003

**This version:**

<http://www.w3.org/TR/2003/REC-PNG-20031110>

**Latest version:**

<http://www.w3.org/TR/PNG>

**Previous version:**

<http://www.w3.org/TR/2003/PR-PNG-20030520>

**Editor:**

David Duce, Oxford Brookes University (Second Edition)

**Authors:**

See [author list](#)

Please refer to the [errata](#) for this document, which may include some normative corrections.

See also the [translations](#) of this document.

Copyright © 2003 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C [liability](#), [trademark](#), [document use](#) and [software licensing](#) rules apply.

### Abstract

This document describes PNG (Portable Network Graphics), an extensible file format for the lossless, portable, well-compressed storage of raster images. PNG provides a patent-free replacement for GIF and can also replace many common uses of TIFF. Indexed-color, grayscale, and truecolor images are supported, plus an optional alpha channel. Sample depths range from 1 to 16 bits.

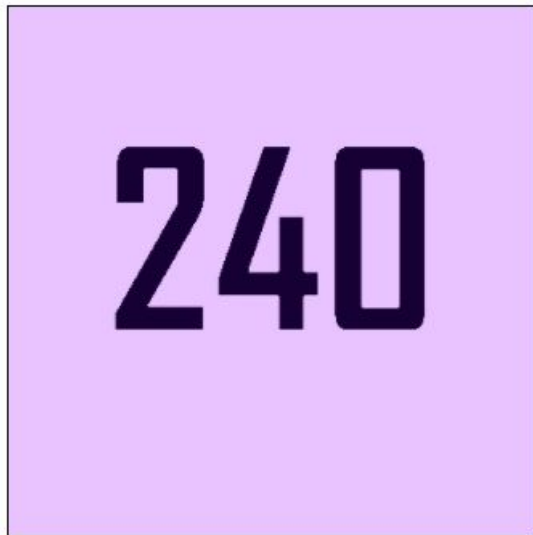


# Overview

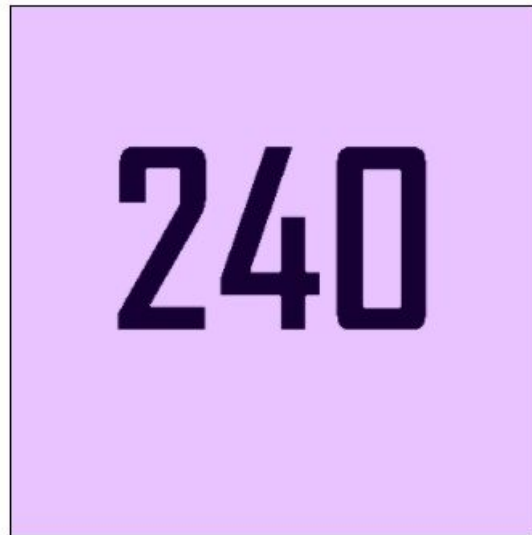
Your second MP in CS 240 is using C to manipulate low-level data. Specifically, we have hidden a few of our favorite GIFs inside of the “classic” CS 240 PNG image:



File: 240.png



File: sample/natalia.png



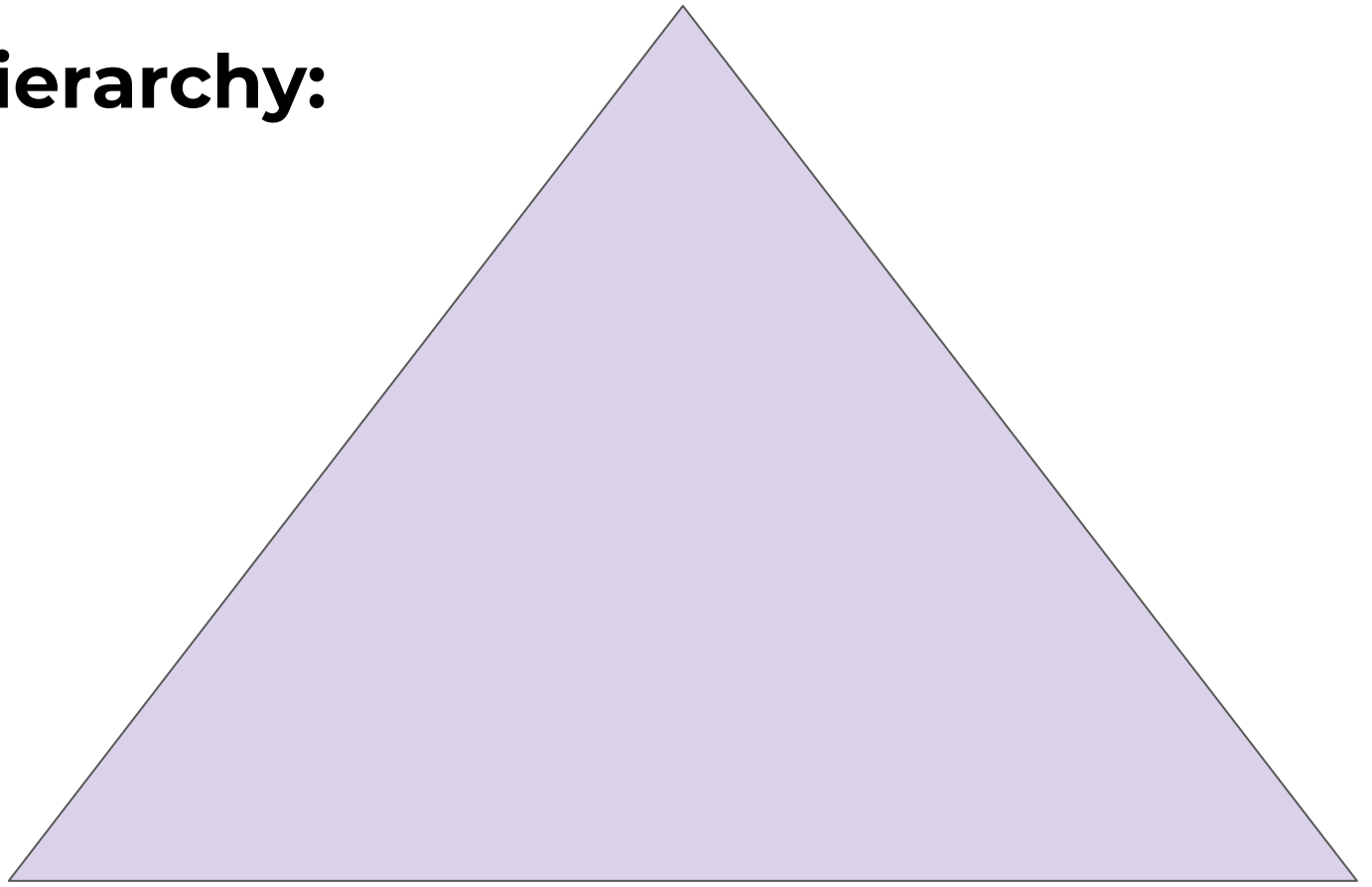
File: sample/waf.png

*All of these images look the same, but two have a hidden GIF inside of them!*

# Memory Hierarchy



# Memory Hierarchy:



# Processor Registers:

# Processor Cache:

# Random Access Memory (RAM):



# Solid State (“Flash”) Memory/Storage:

# Hard Disk Storage:

# High-Density / “Offline” / “Tape” Storage:

# Memory Hierarchy

The image features a photograph of a crowd of people gathered around a statue of Alma Mater, overlaid with a semi-transparent orange filter. The text "Memory Hierarchy" is centered in white. The background shows a large group of people, some looking towards the camera and others looking towards the statue. The statue is a classical figure, likely representing a personification of memory or wisdom. The overall scene is outdoors, with trees visible in the background.

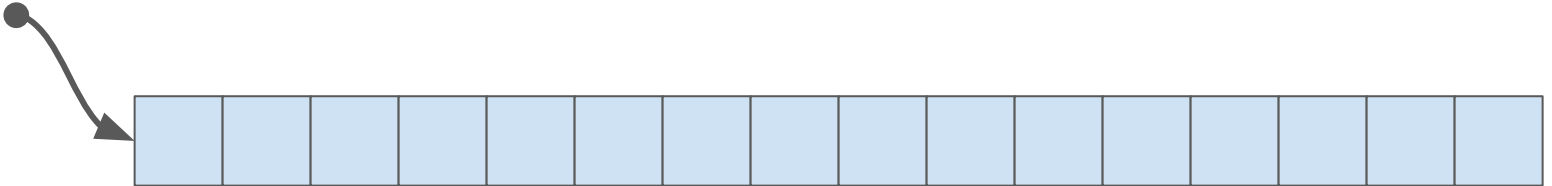
# Program #1: 04rc.c

```
12 // Allocate an array of SIZE x SIZE of `unsigned ints`:
13 unsigned int *array = malloc(SIZE * SIZE *
                               sizeof(unsigned int));
14
15 // Add data to each element of the array:
16 for (unsigned int c = 0; c < SIZE; c++) {
17     for (unsigned int r = 0; r < SIZE; r++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```

# Example: SIZE=4

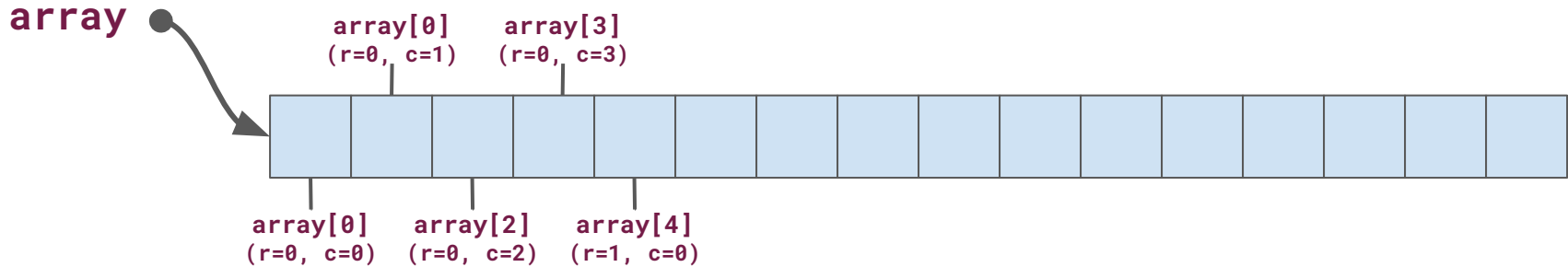
```
12 // Allocate an array of SIZE x SIZE of `unsigned ints`:  
13 unsigned int *array = malloc(SIZE * SIZE *  
                               sizeof(unsigned int));
```

array

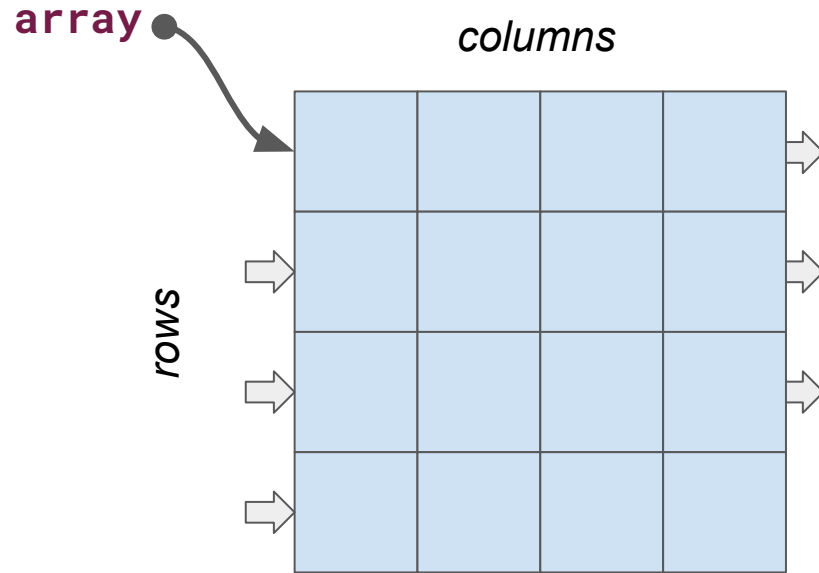


# Example: SIZE=4

```
15 // Add data to each element of the array:
16 for (unsigned int c = 0; c < SIZE; c++) {
17     for (unsigned int r = 0; r < SIZE; r++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```

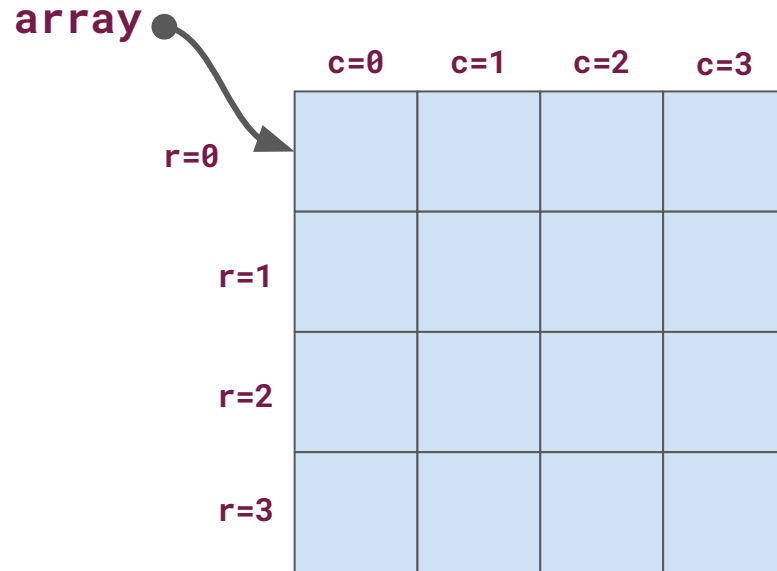


# Example: SIZE=4

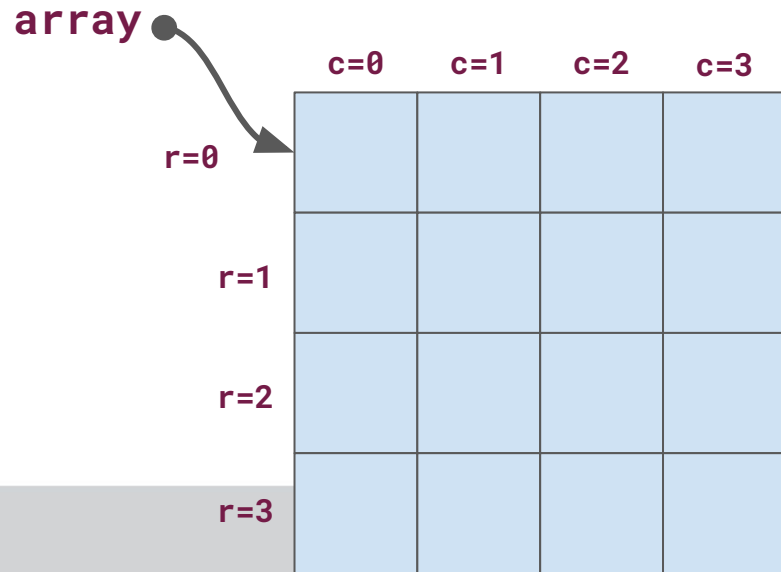




# Example: SIZE=4

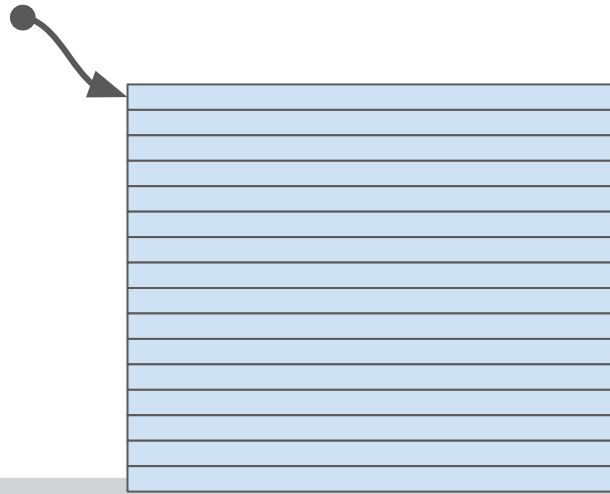


```
15 // Add data to each element of the array:
16 for (unsigned int c = 0; c < SIZE; c++) {
17     for (unsigned int r = 0; r < SIZE; r++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```



```
15 // Add data to each element of the array:
16 for (unsigned int c = 0; c < SIZE; c++) {
17     for (unsigned int r = 0; r < SIZE; r++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```

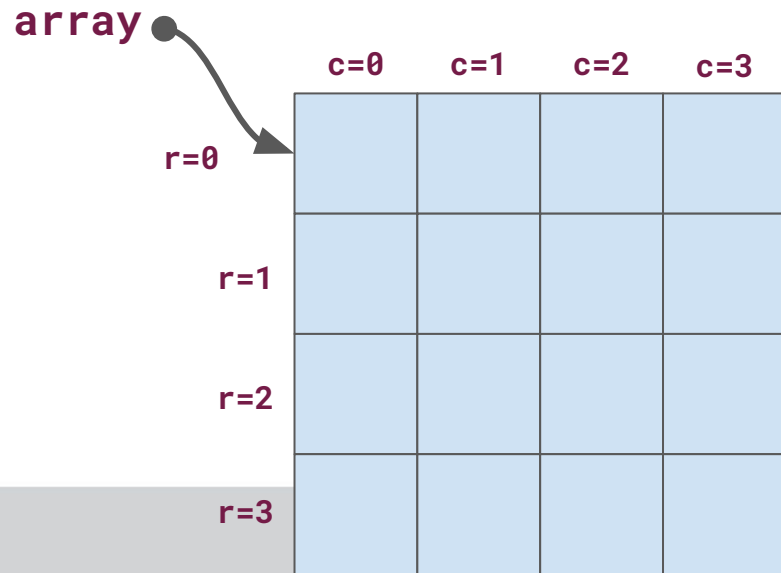
array



## Program #2: 04rc.c

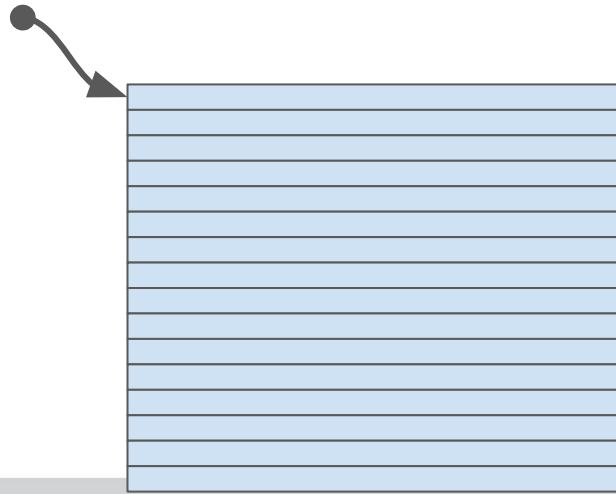
```
12 // Allocate an array of SIZE x SIZE of `unsigned ints`:
13 unsigned int *array = malloc(SIZE * SIZE *
                               sizeof(unsigned int));
14
15 // Add data to each element of the array:
16 for (unsigned int r = 0; r < SIZE; r++) {
17     for (unsigned int c = 0; c < SIZE; c++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```

```
15 // Add data to each element of the array:
16 for (unsigned int r = 0; r < SIZE; r++) {
17     for (unsigned int c = 0; c < SIZE; c++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```



```
15 // Add data to each element of the array:
16 for (unsigned int r = 0; r < SIZE; r++) {
17     for (unsigned int c = 0; c < SIZE; c++) {
18         array[(r * SIZE) + c] = (r * SIZE) + c;
19     }
20 }
```

array



# Let's Test It!

```
$ make  
$ time ./04cr
```

```
$ time ./04rc
```

# How SIZE impacts performance?

