

15-213: S19 Midterm Review Session

~~Big Brain~~ Cyrus, Eugene, Niko and Emma
3 Mar 2019

Agenda

- Review midterm problems
 - Cache
 - Floats
 - Arrays
 - Stack
 - Structs (time permitting)
- Q&A for general midterm problems

Reminders

- There will be office hours this week, but with fewer TAs than usual! If you need any help with midterm questions after today, please make a public Piazza post (and specify exactly which question!)
- Cheat sheet: ONE 8½ x 11 in. sheet, both sides. Please use only English!
- Lecture is still happening! Go learn things!

Problem 1: Cache

- Things to remember/put on a cheat sheet because please don't try to memorize all of this:
 - Direct mapped vs. n-way associative vs. fully associative
 - Tag/Set/Block offset bits, how do they map depending on cache size?
 - LRU policies

Problem 1: Cache

- A. Assume you have a cache of the following structure:
 - a. 32-byte blocks
 - b. 2 sets
 - c. Direct-mapped
 - d. 8-bit address space
 - e. The cache is cold prior to access
- B. What does the address decomposition look like?

0 0 0 0 0 0 0 0

Problem 1: Cache

- A. Assume you have a cache of the following structure:
 - a. 32-byte blocks
 - b. 2 sets
 - c. Direct-mapped
 - d. 8-bit address space
 - e. The cache is cold prior to access
- B. What does the address decomposition look like?

0 0 0 0 0 0 0 0

Problem 1: Cache

Address	Set	Tag	H/M	Evict? Y/N
0x56				
0x6D				
0x49				
0x3A				

Problem 1: Cache

Address	Set	Tag	H/M	Evict? Y/N
0101 0110				
0110 1101				
0100 1001				
0011 1010				

Problem 1: Cache

Address	Set	Tag	H/M	Evict? Y/N
0101 0110	0	01	M	N
0110 1101				
0100 1001				
0011 1010				

Problem 1: Cache

Address	Set	Tag	H/M	Evict? Y/N
0101 0110	0	01	M	N
0110 1101	1	01	M	N
0100 1001				
0011 1010				

Problem 1: Cache

Address	Set	Tag	H/M	Evict? Y/N
0101 0110	0	01	M	N
0110 1101	1	01	M	N
0100 1001	0	01	H	N
0011 1010				

Problem 1: Cache

Address	Set	Tag	H/M	Evict? Y/N
0101 0110	0	01	M	N
0110 1101	1	01	M	N
0100 1001	0	01	H	N
0011 1010	1	00	M	Y

Problem 1: Cache

- A. Assume you have a cache of the following structure:
 - a. 2-way associative
 - b. 4 sets, 64-byte blocks
- B. What does the address decomposition look like?

... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Problem 1: Cache

- A. Assume you have a cache of the following structure:
 - a. 2-way associative
 - b. 4 sets, 64-byte blocks
- B. What does the address decomposition look like?

... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Problem 1: Cache

B. Assume A and B are 128 ints and cache-aligned.

- a. What is the miss rate of pass 1?**
- b. What is the miss rate of pass 2?**

```
int get prod and copy(int *A, int *B) {  
    int length = 64;  
    int prod = 1;  
    // pass 1  
    for (int i = 0; i < length; i+=4) {  
        prod*=A[i];  
    }  
    // pass 2  
    for (int j = length-1; j > 0; j-=4) {  
        A[j] = B[j];  
    }  
    return prod;  
}
```

Problem 1: Cache

B. Pass 1: Only going through 64 ints with step size 4. Each miss loads 16 ints into a cache line, giving us 3 more hits before loading into a new line.

```
int get prod and copy(int *A, int *B) {  
    int length = 64;  
    int prod = 1;  
    // pass 1  
    for (int i = 0; i < length; i+=4) {  
        prod*=A[i];  
    }  
    // pass 2  
    for (int j = length-1; j > 0; j-=4) {  
        A[j] = B[j];  
    }  
    return prod;  
}
```


Problem 1: Cache

B. Pass 1: 25% miss

```
int get prod and copy(int *A, int *B) {  
    int length = 64;  
    int prod = 1;  
    // pass 1  
    for (int i = 0; i < length; i+=4) {  
        prod*=A[i];  
    }  
    // pass 2  
    for (int j = length-1; j > 0; j-=4) {  
        A[j] = B[j];  
    }  
    return prod;  
}
```

Problem 1: Cache

B. Pass 2: Our cache is the same size as our working set! Due to cache alignment, we won't evict anything from A, but still get a 1:3 miss:hit ratio for B.

```
int get prod and copy(int *A, int *B) {  
    int length = 64;  
    int prod = 1;  
    // pass 1  
    for (int i = 0; i < length; i+=4) {  
        prod*=A[i];  
    }  
    // pass 2  
    for (int j = length-1; j > 0; j-=4) {  
        A[j] = B[j];  
    }  
    return prod;  
}
```

Problem 1: Cache

B. Pass 2: For every 4 loop iterations, we get all hits for accessing A and 1 miss for accessing B, which gives us $\frac{1}{8}$ miss.

```
int get prod and copy(int *A, int *B) {  
    int length = 64;  
    int prod = 1;  
    // pass 1  
    for (int i = 0; i < length; i+=4) {  
        prod*=A[i];  
    }  
    // pass 2  
    for (int j = length-1; j > 0; j-=4) {  
        A[j] = B[j];  
    }  
    return prod;  
}
```

Problem 1: Cache

B. Pass 2: 12.5% miss

```
int get prod and copy(int *A, int *B) {  
    int length = 64;  
    int prod = 1;  
    // pass 1  
    for (int i = 0; i < length; i+=4) {  
        prod*=A[i];  
    }  
    // pass 2  
    for (int j = length-1; j > 0; j-=4) {  
        A[j] = B[j];  
    }  
    return prod;  
}
```

Problem 2: Float

- A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.
- a) $31/8$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 1: Convert the fraction into the form $(-1)^s M 2^E$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 1: Convert the fraction into the form $(-1)^s M 2^E$

$s = 0$

$M = 31/16$ (M should be in the range $[1.0, 2.0)$ for normalised numbers)

$E = 1$

Problem 2: Float

- A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 2: Convert M into binary and find value of exp

$s = 0$

$M = 31/16$ (M should be in the range [1.0, 2.0) for normalised numbers)

$E = 1$

Problem 2: Float

- A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 2: Convert M into binary and find value of exp
 $s = 0$

$$M = 31/16 \Rightarrow 1.1111$$

$$\text{bias} = 2^{k-1} - 1 \text{ (k is the number of exponent bits)} = 1$$

$$E = 1 \Rightarrow \text{exponent} = 1 + \text{bias} = 2$$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 3: Find the fraction bits and exponent bits

$s = 0$

$M = 1.1111 \Rightarrow$ fraction bits are 1111

exponent bits are 10

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 4: Take care of rounding issues

Current number is 0 10 111 1 \leq excess bit

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 4: Take care of rounding issues

Current number is 0 10 111 1 \leq excess bit

Guard bit = 1

Round bit = 1

Round up! (add 1 to the fraction bits)

Problem 2: Float

- A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 4: Take care of rounding issues

Current number is 0 10 111 1 \leq excess bit

Adding 1 overflows the floating bits, so we increment the exponent bits by 1 and set the fraction bits to 0

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) $31/8$

Step 4: Take care of rounding issues

Result is 0 11 000 \leq Infinity!

Problem 2: Float

- A.** Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.
- b) $-7/8$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

b) $-7/8$

Step 1: Convert the fraction into the form $(-1)^s M 2^E$

$$s = 1$$

$$M = 7/4$$

$$E = -1$$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

b) $-7/8$

Step 2: Convert M into binary and find value of exp

$s = 1$

$$M = 7/4 \Rightarrow 1.11$$

$$\text{bias} = 2^{k-1} - 1 \text{ (k is the number of exponent bits)} = 1$$

$$E = -1 \Rightarrow \text{exponent} = -1 + \text{bias} = 0$$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

b) $-7/8$

Step 2: Convert M into binary and find value of exp
 $s = 1$

$M = 7/4 \Rightarrow 1.11$ **\leq (We assumed M was in the range [1.0, 2.0). Need to update the value of M)**

$\text{bias} = 2^{k-1} - 1$ (k is the number of exponent bits) = 1

$E = -1 \Rightarrow \text{exponent} = -1 + \text{bias} = 0$ **\leq denormalized**

Problem 2: Float

- A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

b) $-7/8$

Step 2: Convert M into binary and find value of exp

$s = 1$

$M = 7/8 \Rightarrow 0.111$ $\leq M$ should be in the range $[0.0, 1.0)$ for denormalized numbers so we divide it by 2

$\text{exp} = 0$

Problem 2: Float

A. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

b) $-7/8$

Step 3: Find the fraction bits and exponent bits

$s = 1$

$M = 0.111 \Rightarrow$ Fraction bits = 111

exp bits = 00

Result = 1 00 111

Problem 2: Float

- B.** Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.
- b) 0 10 101

Problem 2: Float

B. Consider a floating point representation with 1 sign bit, 2 exponent bits and 3 fraction bits. Convert the following numbers into their floating point representation.

a) 0 10 101

$$s = 0$$

$$\text{exp} = 2 \Rightarrow E = \text{exp} - \text{bias} = 1 \text{ (normalized)}$$

$$M = 1.101 \text{ (between 1 and 2 since it is normalised)}$$

$$\text{Result} = 2 * 1.101 = 2 * (13/8) = 13/4$$

Problem 2: Float

- Things to remember/ put on your cheat sheet:
 - Floating point representation $(-1)^s M 2^E$
 - Values of M in normalized vs denormalized
 - Difference between normalized, denormalized and special floating point numbers
 - Rounding
 - Bit values of smallest and largest normalized and denormalized numbers



Problem 3: Arrays

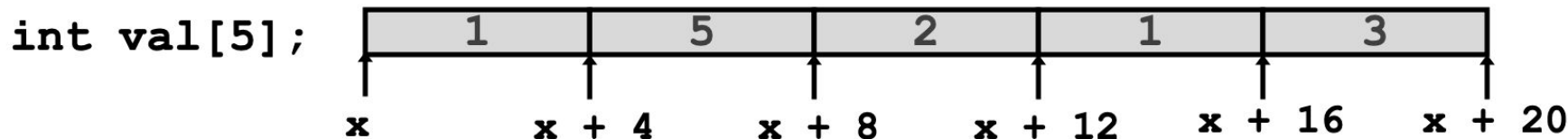
IMPORTANT POINTS + TIPS:

- *Remember your indexing rules! They'll take you 95% of the way there.*
- Be careful about addressing (&) vs. dereferencing (*)
- You may be asked to look at assembly!
- Feel free to put lecture/recitation/textbook examples in your cheatsheet.



Problem 3: Arrays

Good toy examples (for your cheatsheet and/or big brain):



- A can be used as the pointer to the first array element: `A[0]`

Type

Value

`val`

`val[2]`

`*(val + 2)`

`&val[2]`

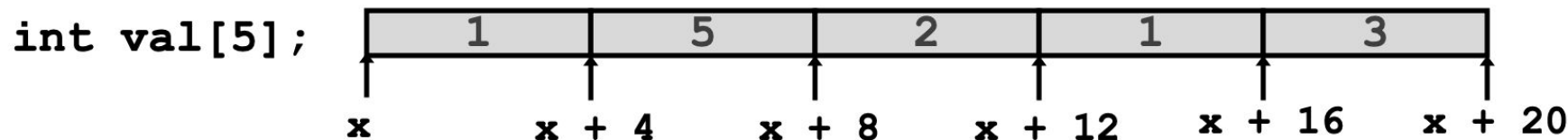
`val + 2`

`val + i`



Problem 3: Arrays

Good toy examples (for your cheatsheet and/or big brain):



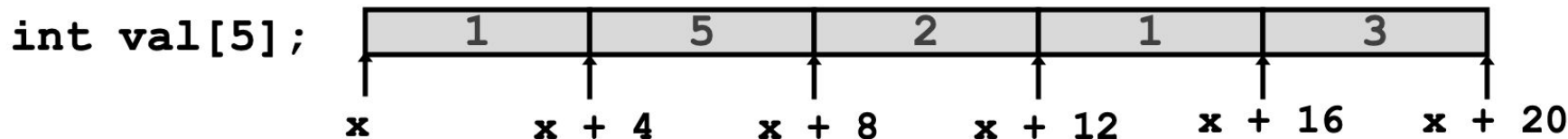
- A can be used as the pointer to the first array element: `A[0]`

	<u>Type</u>	<u>Value</u>
<code>val</code>	<code>int *</code>	<code>x</code>
<code>val[2]</code>	<code>int</code>	<code>2</code>
<code>*(val + 2)</code>	<code>int</code>	<code>2</code>
<code>&val[2]</code>	<code>int *</code>	<code>x + 8</code>
<code>val + 2</code>	<code>int *</code>	<code>x + 8</code>
<code>val + i</code>	<code>int *</code>	<code>x + (4 * i)</code>



Problem 3: Arrays

Good toy examples (for your cheatsheet and/or big brain):



- `A` can be used as the pointer to the first array element: `A[0]`

	<u>Type</u>	<u>Value</u>
<code>val</code>	<code>int *</code>	<code>x</code>
<code>val[2]</code>	<code>int</code>	2
<code>*(val + 2)</code>	<code>int</code>	2
<code>&val[2]</code>	<code>int *</code>	<code>x + 8</code>
<code>val + 2</code>	<code>int *</code>	<code>x + 8</code>
<code>val + i</code>	<code>int *</code>	<code>x + (4 * i)</code>

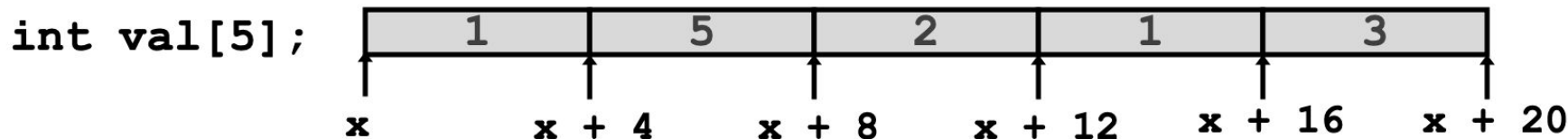
Accessing methods:

- `val[index]`
- `*(val + index)`



Problem 3: Arrays

Good toy examples (for your cheatsheet and/or big brain):



- `A` can be used as the pointer to the first array element: `A[0]`

	<u>Type</u>	<u>Value</u>	
<code>val</code>	<code>int *</code>	<code>x</code>	<div>Accessing methods:</div> <ul style="list-style-type: none"> • <code>val[index]</code> • <code>*(val + index)</code>
<code>val[2]</code>	<code>int</code>	2	
<code>*(val + 2)</code>	<code>int</code>	2	<div>Addressing methods:</div> <ul style="list-style-type: none"> • <code>&val[index]</code> • <code>val + index</code>
<code>&val[2]</code>	<code>int *</code>	<code>x + 8</code>	
<code>val + 2</code>	<code>int *</code>	<code>x + 8</code>	
<code>val + i</code>	<code>int *</code>	<code>x + (4 * i)</code>	

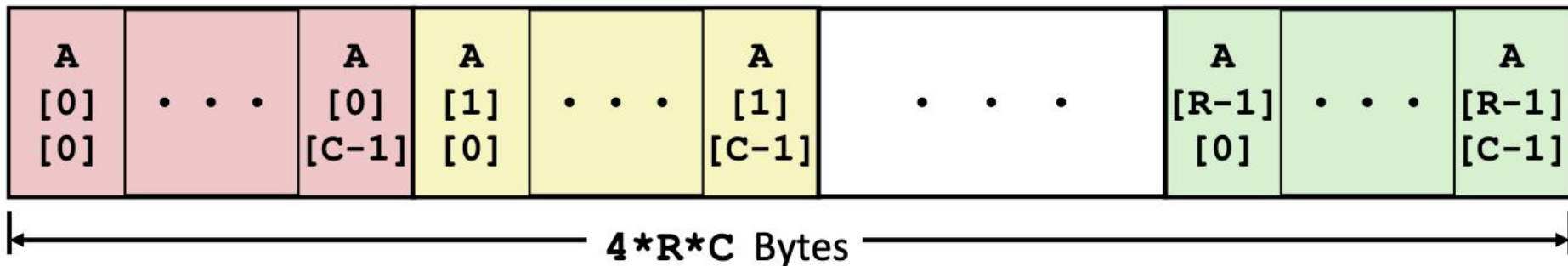


Problem 3: Arrays

Nested indexing rules (for your cheatsheet and/or big brain):

- Declared: `T A[R] [C]`
- Contiguous chunk of space (think of multiple arrays lined up next to each other)

`int A[R] [C];`



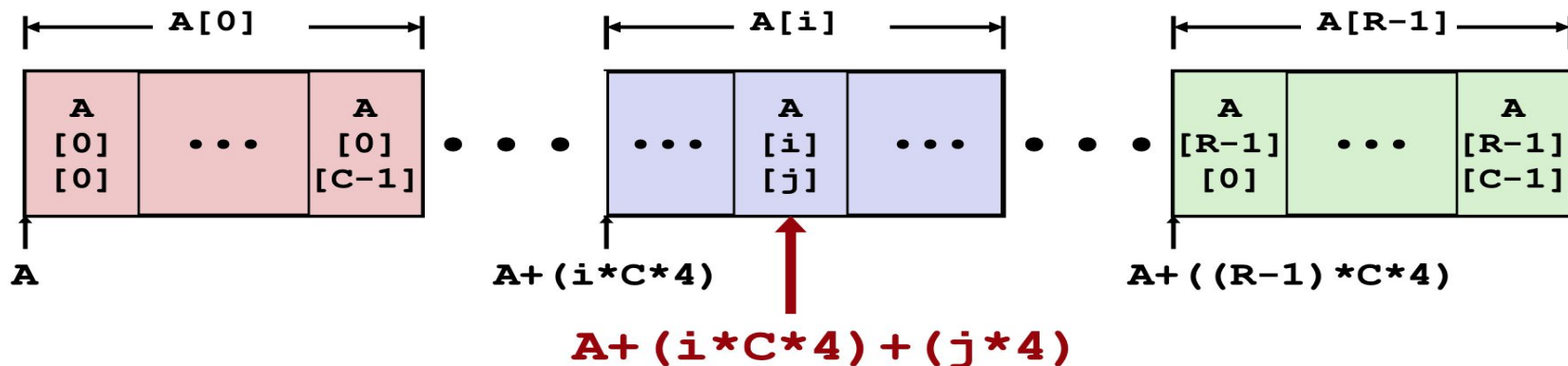


Problem 3: Arrays

Nested indexing rules (for your cheatsheet and/or big brain):

- Arranged in ROW-MAJOR ORDER - think of row vectors
- $A[i]$ is an array of C elements (“columns”) of type T

```
int A[R][C];
```





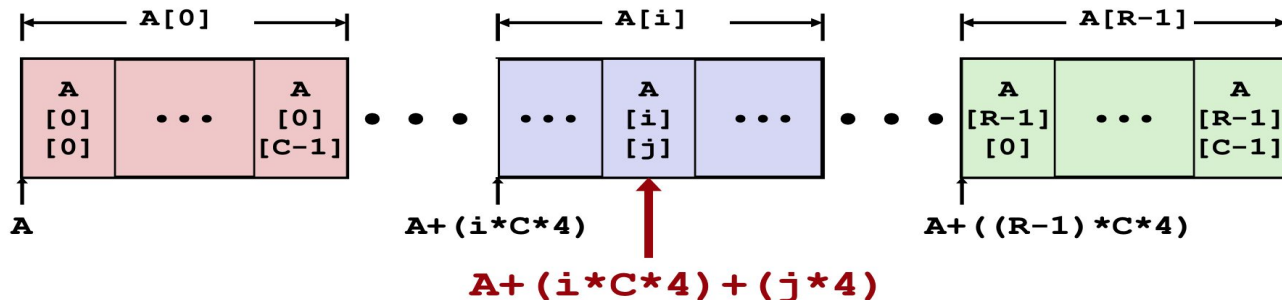
Problem 3: Arrays

Nested indexing rules (for your cheatsheet and/or big brain):

$\mathbf{A}[\mathbf{i}][\mathbf{j}]$ is element of type T , which requires K bytes

$$\begin{aligned} \text{Address } \mathbf{A} + \mathbf{i} * (\mathbf{C} * \mathbf{K}) + \mathbf{j} * \mathbf{K} \\ = \mathbf{A} + (\mathbf{i} * \mathbf{C} + \mathbf{j}) * \mathbf{K} \end{aligned}$$

```
int A[R][C];
```





Problem 3: Arrays

Consider accessing elements of **A**....

	<u>Compiles</u>	<u>Bad Deref?</u>	<u>Size (bytes)</u>
<code>int A1[3][5]</code>			
<code>int *A2[3][5]</code>			
<code>int (*A3)[3][5]</code>			
<code>int *(A4[3][5])</code>			
<code>int (*A5[3])[5]</code>			



Problem 3: Arrays

Consider accessing elements of **A**....

	<u>Compiles</u>	<u>Bad Deref?</u>	<u>Size (bytes)</u>
<code>int A1[3][5]</code>	Y	N	$3 * 5 * 4 = 60$
<code>int *A2[3][5]</code>			
<code>int (*A3)[3][5]</code>			
<code>int *(A4[3][5])</code>			
<code>int (*A5[3])[5]</code>			



Problem 3: Arrays

Consider accessing elements of **A**....

	<u>Compiles</u>	<u>Bad Deref?</u>	<u>Size (bytes)</u>
<code>int A1[3][5]</code>	Y	N	$3 * 5 * (4) = 60$
<code>int *A2[3][5]</code>	Y	N	$3 * 5 * (8) = 120$
<code>int (*A3)[3][5]</code>			
<code>int *(A4[3][5])</code>			
<code>int (*A5[3])[5]</code>			



Problem 3: Arrays

Consider accessing elements of **A**....

	<u>Compiles</u>	<u>Bad Deref?</u>	<u>Size (bytes)</u>
<code>int A1[3][5]</code>	Y	N	$3 * 5 * (4) = 60$
<code>int *A2[3][5]</code>	Y	N	$3 * 5 * (8) = 120$
<code>int (*A3)[3][5]</code>	Y	N	$1 * 8 = 8$
<code>int *(A4[3][5])</code>			
<code>int (*A5[3])[5]</code>			



Problem 3: Arrays

Consider accessing elements of **A**....

	<u>Compiles</u>	<u>Bad Deref?</u>	<u>Size (bytes)</u>
<code>int A1[3][5]</code>	Y	N	$3 * 5 * (4) = 60$
<code>int *A2[3][5]</code>	Y	N	$3 * 5 * (8) = 120$
<code>int (*A3)[3][5]</code>	Y	N	$1 * 8 = 8$
<code>int *(A4[3][5])</code>	Y	N	$3 * 5 * (8) = 120$
<code>int (*A5[3])[5]</code>			

A4 is a pointer to a 3x5 (int *) element array



Problem 3: Arrays

Consider accessing elements of **A**....

	<u>Compiles</u>	<u>Bad Deref?</u>	<u>Size (bytes)</u>
<code>int A1[3][5]</code>	Y	N	$3 * 5 * (4) = 60$
<code>int *A2[3][5]</code>	Y	N	$3 * 5 * (8) = 120$
<code>int (*A3)[3][5]</code>	Y	N	$1 * 8 = 8$
<code>int *(A4[3][5])</code>	Y	N	$3 * 5 * (8) = 120$
<code>int (*A5[3])[5]</code>	Y	N	$3 * 8 = 24$

↖ A5 is an array of 3 elements of type (int *)



Problem 3: Arrays

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
<code>int A1[3][5]</code>	Y	N	60	Y	N	20	Y	N	4
<code>int *A2[3][5]</code>	Y	N	120	Y	N	40	Y	N	8
<code>int (*A3)[3][5]</code>	Y	N	8	Y	Y	60	Y	Y	20
<code>int *(A4[3][5])</code>	Y	N	120	Y	N	40	Y	N	8
<code>int (*A5[3])[5]</code>	Y	N	24	Y	N	8	Y	Y	20

ex., A3: pointer to a 3x5 int array
 *A3: 3x5 int array (3 * 5 elements * each 4 bytes = 60)
 **A3: BAD, but means stepping inside one of 3 “rows” c



Problem 3: Arrays

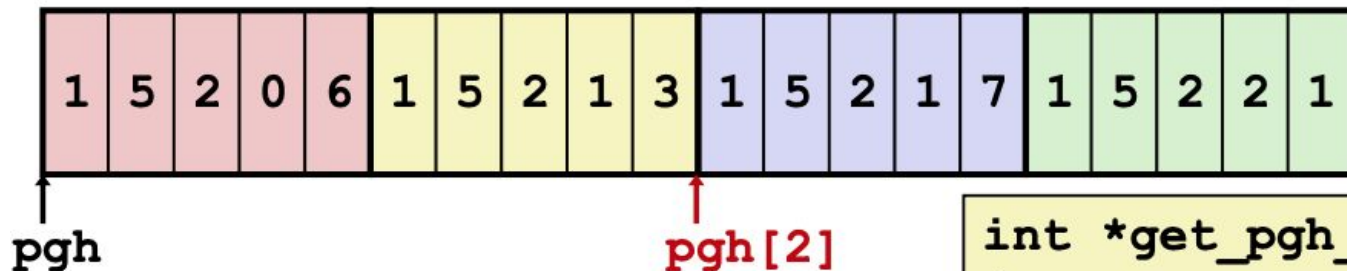
Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
<code>int A1[3][5]</code>	Y	N	60	Y	N	20	Y	N	4
<code>int *A2[3][5]</code>	Y	N	120	Y	N	40	Y	N	8
<code>int (*A3)[3][5]</code>	Y	N	8	Y	Y	60	Y	Y	20
<code>int *(A4[3][5])</code>	Y	N	120	Y	N	40	Y	N	8
<code>int (*A5[3])[5]</code>	Y	N	24	Y	N	8	Y	Y	20

ex., A5: array of 3 (int *) pointers
 *A5: 1 (int *) pointer, points to an array of 5 ints
 **A5: BAD, means accessing 5 individual ints of the pointer
 (stepping inside “row”)



Problem 3: Arrays

Sample assembly-type questions



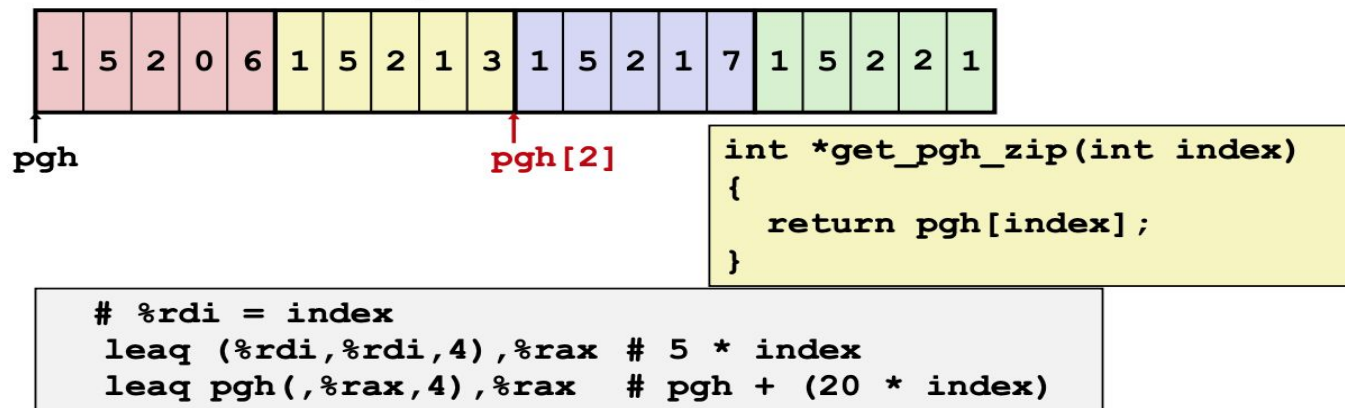
```
int *get_pgh_zip(int index)
{
    return pgh[index];
}
```

```
# %rdi = index
leaq (%rdi,%rdi,4),%rax # 5 * index
leaq pgh(,%rax,4),%rax  # pgh + (20 * index)
```




Problem 3: Arrays

Nested Array Row Access Code



■ Row Vector

- `pgh[index]` is array of 5 `int`'s
- Starting address `pgh+20*index`

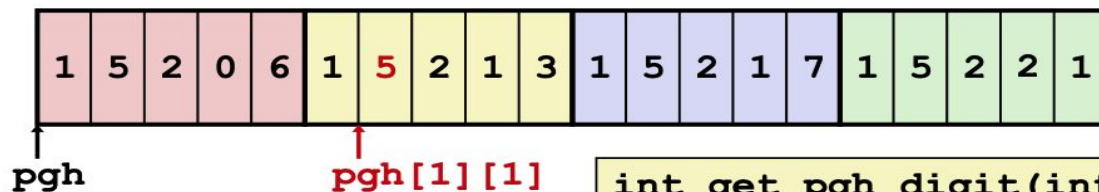
■ Machine Code

- Computes and returns address
- Compute as `pgh + 4*(index+4*index)`



Problem 3: Arrays

Nested Array Element Access Code



```
int get_pgh_digit(int index, int dig)
{
    return pgh[index][dig];
}
```

```
leaq    (%rdi,%rdi,4), %rax    # 5*index
addl    %rax, %rsi             # 5*index+dig
movl    pgh(,%rsi,4), %eax     # M[pgh + 4*(5*index+dig)]
```

■ Array Elements

- `pgh[index][dig]` is `int`
- Address: $\text{pgh} + 20 \cdot \text{index} + 4 \cdot \text{dig}$
 $= \text{pgh} + 4 \cdot (5 \cdot \text{index} + \text{dig})$

Problem 4: Stack

- Important things to remember:
 - Stack grows DOWN!
 - %rsp = stack pointer, always point to “top” of stack
 - Push and pop, call and ret
 - Stack frames: how they are allocated and freed
 - Which registers used for arguments? Return values?
 - Little endianness
- ALWAYS helpful to draw a stack diagram!!
- Stack questions are like Assembly questions on steroids

Problem 4: Stack

Consider the following code:

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}

void caller() {
    foo("midtermexam", 0x15213);
}
```

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1

.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy

.L1:
    addq    $24, %rsp
    ret

caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
.string "midtermexam"
```

Hints:

- `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating `'\0'` character) to address `dst`.
- Keep endianness in mind!
- Table of hex values of characters in "midtermexam"

Assumptions:

- `%rsp = 0x800100` just before `caller()` calls `foo()`
- `.LC0` is at address `0x400300`

Problem 4: Stack

Consider the following code:

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}

void caller() {
    foo("midtermexam", 0x15213);
}
```

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1

.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy

.L1:
    addq    $24, %rsp
    ret

caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section      .rodata.str1.1,"aMS",@progbits,1
.LC0:= 0x400300
.string "midtermexam"
```

→ %rsp = 0x800100

Hints:

- `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating `'\0'` character) to address `dst`.
- Keep endianness in mind!
- Table of hex values of characters in "midtermexam"

Assumptions:

-
- `%rsp = 0x800100` just before `caller()` calls `foo()`
 - `.LC0` is at address `0x400300`

Problem 4: Stack

Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}

void caller() {
    foo("midtermexam", 0x15213);
}
```

Hints:

- Step through the program instruction by instruction from start to end
- Draw a stack diagram!!!
- Keep track of registers too

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1

.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    End call    strcpy

.L1:
    addq    $24, %rsp
    ret

caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    Start call    foo           %rsp = 0x800100
    addq    $8, %rsp
    ret

        .section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
        .string "midtermexam"
```

Problem 4: Stack

Arrow is instruction that will
execute NEXT

Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```


<code>%rsp</code>	0x800100
<code>%rdi</code>	.LC0
<code>%rsi</code>	0x15213

0x800100	
0x8000f8	
0x8000f0	
0x8000e8	
0x8000e0	
0x8000d8	
0x8000d0	
0x8000c8	
0x8000c0	
0x8000b8	

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
```

```
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    End call    strcpy
```

```
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
     call    foo
    addq    $8, %rsp
    ret
```

`%rsp = 0x800100`

```
.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack

Question 1: What is the hex value of `%rsp` just **before** `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

<code>%rsp</code>	<code>0x8000f8</code>
<code>%rdi</code>	<code>.LC0</code>
<code>%rsi</code>	<code>0x15213</code>

<code>0x800100</code>	<code>?</code>
<code>0x8000f8</code>	ret address for <code>foo()</code>
<code>0x8000f0</code>	
<code>0x8000e8</code>	
<code>0x8000e0</code>	
<code>0x8000d8</code>	
<code>0x8000d0</code>	
<code>0x8000c8</code>	
<code>0x8000c0</code>	
<code>0x8000b8</code>	

```
foo:
    → subq    $24, %rsp
    cmpl     $0xdeadbeef, %esi
    je       .L2
    movl     $0xdeadbeef, %esi
    call     foo
    jmp      .L1

.L2:
    movq     %rdi, %rsi
    movq     %rsp, %rdi
    End call  strcpy
.L1:
    addq     $24, %rsp
    ret
```

```
caller:
    subq     $8, %rsp
    movl     $86547, %esi
    movl     $.LC0, %edi
    call     foo
    addq     $8, %rsp
    ret

.section     .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```


Problem 4: Stack

Hint: \$24 in decimal = 0x18


Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

<code>%rsp</code>	0x8000e0
<code>%rdi</code>	.LC0
<code>%rsi</code>	0x15213

0x800100	?
0x8000f8	ret address for <code>foo()</code>
0x8000f0	?
0x8000e8	?
0x8000e0	?
0x8000d8	
0x8000d0	
0x8000c8	
0x8000c0	
0x8000b8	

```
foo:
    subq    $24, %rsp
     cml     $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    End call    strcpy
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section    .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack


Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

<code>%rsp</code>	<code>0x8000e0</code>
<code>%rdi</code>	<code>.LC0</code>
<code>%rsi</code>	<code>0xdeadbeef</code>

<code>0x800100</code>	<code>?</code>
<code>0x8000f8</code>	ret address for <code>foo()</code>
<code>0x8000f0</code>	<code>?</code>
<code>0x8000e8</code>	<code>?</code>
<code>0x8000e0</code>	<code>?</code>
<code>0x8000d8</code>	
<code>0x8000d0</code>	
<code>0x8000c8</code>	
<code>0x8000c0</code>	
<code>0x8000b8</code>	

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
     call    foo
    jmp     .L1
```

```
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    End    call    strcpy
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret
```

```
.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack

Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

<code>%rsp</code>	<code>0x8000d8</code>
<code>%rdi</code>	<code>.LC0</code>
<code>%rsi</code>	<code>0xdeadbeef</code>

<code>0x800100</code>	<code>?</code>
<code>0x8000f8</code>	ret address for <code>foo()</code>
<code>0x8000f0</code>	<code>?</code>
<code>0x8000e8</code>	<code>?</code>
<code>0x8000e0</code>	<code>?</code>
<code>0x8000d8</code>	ret address for <code>foo()</code>
<code>0x8000d0</code>	
<code>0x8000c8</code>	
<code>0x8000c0</code>	
<code>0x8000b8</code>	

```
foo:
    → subq    $24, %rsp
    cmpl     $0xdeadbeef, %esi
    je      .L2
    movl     $0xdeadbeef, %esi
    call     foo
    jmp      .L1
.L2:
    movq     %rdi, %rsi
    movq     %rsp, %rdi
    End call    strcpy
.L1:
    addq     $24, %rsp
    ret
```

```
caller:
    subq     $8, %rsp
    movl     $86547, %esi
    movl     $.LC0, %edi
    call     foo
    addq     $8, %rsp
    ret

.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack


Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

<code>%rsp</code>	<code>0x8000c0</code>
<code>%rdi</code>	<code>.LC0</code>
<code>%rsi</code>	<code>0xdeadbeef</code>

<code>0x800100</code>	?
<code>0x8000f8</code>	ret address for <code>foo()</code>
<code>0x8000f0</code>	?
<code>0x8000e8</code>	?
<code>0x8000e0</code>	?
<code>0x8000d8</code>	ret address for <code>foo()</code>
<code>0x8000d0</code>	?
<code>0x8000c8</code>	?
<code>0x8000c0</code>	?
<code>0x8000b8</code>	

```
foo:
    subq    $24, %rsp
     cml     $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    End call    strcpy
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack

Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?


```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

<code>%rsp</code>	<code>0x8000c0</code>
<code>%rdi</code>	<code>.LC0</code>
<code>%rsi</code>	<code>0xdeadbeef</code>

<code>0x800100</code>	?
<code>0x8000f8</code>	ret address for <code>foo()</code>
<code>0x8000f0</code>	?
<code>0x8000e8</code>	?
<code>0x8000e0</code>	?
<code>0x8000d8</code>	ret address for <code>foo()</code>
<code>0x8000d0</code>	?
<code>0x8000c8</code>	?
<code>0x8000c0</code>	?
<code>0x8000b8</code>	

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
```

```
.L2:
     movq    %rsi, %rdi
    movq    %rsp, %rdi
    End call    strcpy
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack

Question 1: What is the hex value of `%rsp` just before `strcpy()` is called for the first time in `foo()` ?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

Answer!

<code>%rsp</code>	<code>0x8000c0</code>
<code>%rdi</code>	<code>0x8000c0</code>
<code>%rsi</code>	<code>.LC0</code>

0x800100	?
0x8000f8	ret address for <code>foo()</code>
0x8000f0	?
0x8000e8	?
0x8000e0	?
0x8000d8	ret address for <code>foo()</code>
0x8000d0	?
0x8000c8	?
0x8000c0	?
0x8000b8	

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1

.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

→ End

Problem 4: Stack

Question 2: What is the hex value of `buf[0]` when `strcpy()` returns?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

%rsp	0x8000c0
%rdi	0x8000c0
%rsi	.LC0

0x800100	?
0x8000f8	ret address for foo()
0x8000f0	?
0x8000e8	?
0x8000e0	?
0x8000d8	ret address for foo()
0x8000d0	?
0x8000c8	?
0x8000c0	?
0x8000b8	

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
```

```
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy
```

```
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret
```

```
.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack

Question 2: What is the hex value of `buf[0]` when `strcpy()` returns?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
```

```
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy
```

```
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret
```

```
.section      .rodata.s
.LC0: = 0x400300
.string "midtermexam"
```

%rsp	0x8000c0
%rdi	0x8000c0
%rsi	.LC0

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8								
0x8000c0							'd'	'i'
	c7						c2	c1
0x8000b8								c0

Problem 4: Stack

Question 2: What is the hex value of `buf[0]` when `strcpy()` returns?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
```

```
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy
```

```
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret
```

```
.section      .rodata.s
.LC0: = 0x400300
.string "midtermexam"
```

%rsp	0x8000c0
%rdi	0x8000c0
%rsi	.LC0

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8	c7		c2		c1		c0	

%rsp	0x8000c0
%rdi	0x8000c0
%rsi	.LC0

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret
```

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8	<div> <div>c3</div> <div>buf[0]</div> <div>c0</div> </div>							

$$= \begin{array}{|c|c|c|c|} \hline 74 & 64 & 69 & 6d \\ \hline \end{array}$$

```
(as int) = 0x7464696d
```

Char	Hex	Char	Hex
a	61	m	6d
d	64	r	72
e	65	t	74
i	69	x	78

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8	buf[0]							

Problem 4: Stack

Question 3: What is the hex value of `buf[1]` when `strcpy()` returns?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
        foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
    call    foo
    jmp     .L1
```

```
.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy
```

```
.L1:
    addq    $24, %rsp
    ret
```

```
caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret
```

```
.section      .rodata.s
.LC0: = 0x400300
.string "midtermexam"
```

%rsp	0x8000c0
%rdi	0x8000c0
%rsi	.LC0

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8	buf[1]				buf[0]			

$$= \begin{array}{|c|c|c|c|} \hline 65 & 6d & 72 & 65 \\ \hline \end{array}$$


```
(as int) = 0x656d7265
```

Char	Hex	Char	Hex
a	61	m	6d
d	64	r	72
e	65	t	74
i	69	x	78

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8	buf[1]							


Problem 4: Stack

Question 4: What is the hex value of `%rdi` at the point where `foo()` is called recursively in the successful arm of the `if` statement?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
         foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```

```
void caller() {
    foo("midtermexam", 0x15213);
}
```

This is before the
FIRST time we call
`foo()`

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
     call     foo
    jmp     .L1

.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy


.L1:
    addq    $24, %rsp
    ret

caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi
    call    foo
    addq    $8, %rsp
    ret

.section      .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```


Problem 4: Stack

Question 4: What is the hex value of `%rdi` at the point where `foo()` is called recursively in the successful arm of the `if` statement?

```
void foo(char *str, int a) {
    int buf[2];
    if (a != 0xdeadbeef) {
         foo(str, 0xdeadbeef);
        return;
    }
    strcpy((char*) buf, str);
}
```


```
void caller() {
    foo("midtermexam", 0x15213);
}
```

- This is before the FIRST time we call `foo()`
- Going backwards, `%rdi` was loaded in `caller()`
- `%rdi = $.LC0 = 0x400300` (based on hint)

```
foo:
    subq    $24, %rsp
    cmpl    $0xdeadbeef, %esi
    je      .L2
    movl    $0xdeadbeef, %esi
     call     foo
    jmp     .L1

.L2:
    movq    %rdi, %rsi
    movq    %rsp, %rdi
    call    strcpy

.L1:
    addq    $24, %rsp
    ret

caller:
    subq    $8, %rsp
    movl    $86547, %esi
    movl    $.LC0, %edi  loaded %rdi
    call    foo
    addq    $8, %rsp
    ret

.section .rodata.str1.1,"aMS",@progbits,1
.LC0: = 0x400300
.string "midtermexam"
```

Problem 4: Stack

Question 5: What part(s) of the stack will be corrupted by invoking `caller()`?
Check all that apply.

- return address from `foo()` to `caller()`
- return address from the recursive call to `foo()`
- `strcpy()`'s return address
- there will be no corruption

- return address from `foo()` to `caller()`
- return address from the recursive call to `foo()`
- `strcpy()`'s return address
- there will be no corruption

0x800100	?							
0x8000f8	ret address for foo()							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address for foo()							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8								

Bonus! Another Cache problem

- Consider you have the following cache:
 - 64-byte capacity
 - Directly mapped
 - You have an 8-bit address space

Bonus!

A. How many tag bits are there in the cache?

- Do we know how many set bits there are? What about offset bits?

$$2^6 = 64$$

- If we have a 64-byte **direct-mapped** cache, we know the number of $s + b$ bits there are total!

- Then $t + s + b = 8 \rightarrow t = 8 - (s + b)$

- Thus, we have 2 tag bits!

Bonus!

B. Fill in the following table, indicating the set number based on the hit/miss pattern.

- a. ~~By the power of guess and check~~ tracing through, identify which partition of $s + b$ bits matches the H/M pattern.

Load	Binary Address	Set	H/M
1	1011 0011		M
2	1010 0111		M
3	1101 1001		M
4	1011 1100		H
5	1011 1001		H

Bonus!

B. Fill in the following table, indicating the set number based on the hit/miss pattern.

- a. ~~By the power of guess and check~~ tracing through, identify which partition of $s + b$ bits matches the H/M pattern.

Load	Binary Address	Set	H/M
1	1011 0011		M
2	1010 0111		M
3	1101 1001		M
4	1011 1100		H
5	1011 1001		H

Bonus!

B. Fill in the following table, indicating the set number based on the hit/miss pattern.

- a. ~~By the power of guess and check~~ tracing through, identify which partition of $s + b$ bits matches the H/M pattern.

Load	Binary Address	Set	H/M
1	10 <u>11</u> 0011		M
2	10 <u>10</u> 0111		M
3	11 <u>01</u> 1001		M
4	10 <u>11</u> 1100		H
5	10 <u>11</u> 1001		H

Bonus!

B. Fill in the following table, indicating the set number based on the hit/miss pattern.

- a. ~~By the power of guess and check~~ tracing through, identify which partition of $s + b$ bits matches the H/M pattern.

Load	Binary Address	Set	H/M
1	10 <u>11</u> 0011	3	M
2	10 <u>10</u> 0111	2	M
3	11 <u>01</u> 1001	1	M
4	10 <u>11</u> 1100	3	H
5	10 <u>11</u> 1001	3	H

Bonus!

- C. How many sets are there? 2 bits \rightarrow 4 sets
How big is each cache line? 4 bits \rightarrow 16 bytes

In summary...

- Read the ~~write-up~~ textbook!
- Also read the ~~write-up~~ lecture slides!
- Midterm covers CS:APP Ch. 1-3, 6
- Ask questions on Piazza! For the midterm, make them public and specific if from the practice server!
- G~O~O~D~~L~U~C~K (also go Knicks)