213: M19 Midterm Review Session

Kashish, and Katherine
26 June 2019

Reminders

- Midterm Friday (June 28) GHC 5207 (10:30 am - noon)

- Cheat sheet: ONE 8½ x 11 in. sheet, both sides
  - ONLY English
  - No previous exam questions
  - No code from previous labs

- No lecture tomorrow - TAs will hold OH during lecture
- Practice exam server is up! Will be stopped Thursday evening!

Agenda

- Midterm problem categories (in order of review)
  - Assembly
  - Stack
  - Cache
  - Struct
  - Arrays (appendix)
  - Floats (appendix)
  - Bitops (not in review)
- Q&A for general midterm problems

Problem 2: 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 2: Stack9

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 2: Stack9

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             %rsp = 0x800100
        addq    $8, %rsp
        ret

        .section        .rodata.str1.1,"aMS",@progbits,1
.LC0:= 0x400300
        .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 2: Stack9

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
        addq    $8, %rsp
        ret

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

`%rsp = 0x800100`

Problem 2: Stack9

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);
}
```

%rsp	0x800100
%rdi	.LC0
%rsi	0x15213

```
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"
```

%rsp = 0x800100

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

Problem 2: Stack9

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);
}
```

%rsp	0x8000f8
%rdi	.LC0
%rsi	0x15213

```
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"
```

0x800100	?
0x8000f8	ret address to `caller()`
0x8000f0	
0x8000e8	
0x8000e0	
0x8000d8	
0x8000d0	
0x8000c8	
0x8000c0	
0x8000b8	

Problem 2: Stack9

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);
}
```

%rsp	0x8000e0
%rdi	.LC0
%rsi	0x15213

```
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"
```

0x800100	?
0x8000f8	ret address to `caller()`
0x8000f0	?
0x8000e8	?
0x8000e0	?
0x8000d8	
0x8000d0	
0x8000c8	
0x8000c0	
0x8000b8	

Problem 2: Stack9

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);
}
```

%rsp	0x8000e0
%rdi	.LC0
%rsi	0xdeadbeef

```
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"
```

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

Problem 2: Stack9

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);
}
```

%rsp	0x8000d8
%rdi	.LC0
%rsi	0xdeadbeef

```
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"
```

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

Problem 2: Stack9

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);
}
```

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

```
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"
```

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

Problem 2: Stack9

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);
}
```

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

```
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"
```

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

Problem 2: Stack9

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!

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

```
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"
```

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

Problem 2: Stack9

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

```
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"
```

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

Problem 2: Stack9

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

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

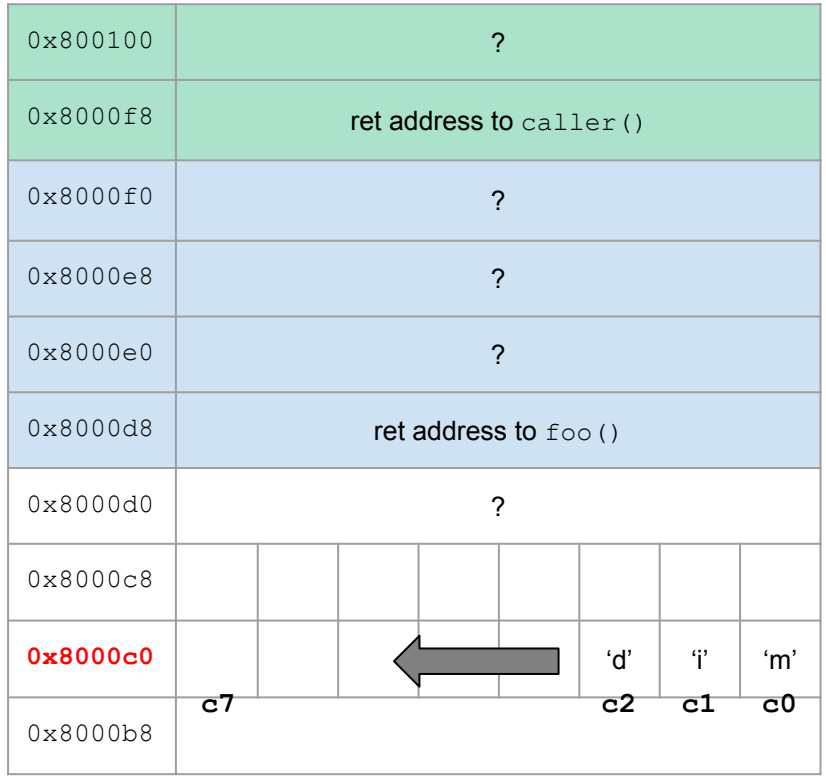
```
void foo(char *str, int a) {
   int buf[2];
   if (a != 0xdeadbeef) {
      foo(str, 0xdeadbeef);
      return;
   }
   strcpy((char*) buf, str);
}
```

```
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"
```



Problem 2: Stack9

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

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

```
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"
```

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

Problem 2: Stack9

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

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

```
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"
```

0x800100	?							
0x8000f8	ret address to `caller()`							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address to `foo()`							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8					**c3**	**buf[0]**		**c0**

Problem 2: Stack9

`buf[0]` = | 't' | 'd' | 'i' | 'm' |

= | 74 | 64 | 69 | 6d |

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

Problem 2: Stack9

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

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

```
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"
```

0x800100	?							
0x8000f8	ret address to `caller()`							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address to `foo()`							
0x8000d0	?							
0x8000c8	?	?	?	?	'\0'	'm'	'a'	'x'
0x8000c0	'e'	'm'	'r'	'e'	't'	'd'	'i'	'm'
0x8000b8	**c7**	**buf[1]**		**c4**	**buf[0]**			

Problem 2: Stack9

`buf[1]` = | ‘e’ | ‘m’ | ‘r’ | ‘e’ |

= | 65 | 6d | 72 | 65 |

`(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 to `caller()`							
0x8000f0	?							
0x8000e8	?							
0x8000e0	?							
0x8000d8	ret address to `foo()`							
0x8000d0	?							
0x8000c8	?	?	?	?	‘\0’	‘m’	‘a’	‘x’
0x8000c0	‘e’	‘m’	‘r’	‘e’	‘t’	‘d’	‘i’	‘m’
0x8000b8	**buf[1]**							

Problem 2: Stack9

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 time we call `foo()` recursively

```
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 2: Stack9

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);
}
```

```
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
```

loaded %rdi

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

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

Problem 2: Stack9

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

Problem 2: Stack9

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

The strcpy didn't overwrite any return addresses, so there was 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								

Problem 3: cache_kgarg

- Things to remember/put on a cheat sheet
  - Direct mapped vs. n-way associative vs. fully associative
  - Load vs. Store (Dirty bytes)
  - Tag/Set/Block offset bits, how do they map depending on cache size?
  - LRU policies

Problem 3: cache_kgarg

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 3: cache_kgarg

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
t t s b b b b b

Problem 3: cache_kgarg

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

0 0 0 0 0 0 0 0
t t s b b b b b

Problem 3: cache_kgarg

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

0 0 0 0 0 0 0 0

t t s b b b b b

Problem 3: cache_kgarg

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

0 0 0 0 0 0 0 0

t t s b b b b b

Problem 3: cache_kgarg

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

0 0 0 0 0 0 0 0

t t s b b b b b

Problem 3: cache_kgarg

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				

0 0 0 0 0 0 0 0

t t s b b b b b

Problem 3: cache_kgarg

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

0 0 0 0 0 0 0 0

t t s b b b b b

Problem 3: cache_kgarg

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 3: cache_kgarg

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 3: cache_kgarg

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 3: cache_kgarg

Pass 1: Going through 64 ints with step size 4.

Consider i = 0:

- **A[0] is a cold miss.**
- **Accessing A[0] loads A[0]-A[15] into set0.**

```
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;
}
```

Recall: 4 sets, 2 lines, 16 ints per line

Problem 3: cache_kgarg

Now i = 4:
- **A[4] is a hit!**

Now i = 8:
- **A[8] is a hit!**

Now i = 12:
- **A[12] is a hit!**

We got 1 miss for the first 4 accesses

```
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 3: cache_kgarg

Consider i = 16:

- **A[16] is a cold miss.**
- **Accessing A[16] loads A[16]-A[31] into set1.**

Same pattern as i = 0 would repeat. Then:

- **A[32]-A[47] goes to set2**
- **A[48]-A[63] goes to set3**

```
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 3: cache_kgarg

Pass 1: Miss rate = 25%

One line of set0-set3 is now full of A[0]-A[63].

So cache is exactly halfway full!

```
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 3: cache_kgarg

Pass 2: We get all hits for accessing A.

For B, we have a pattern similar to A in the first pass. We get 1 miss, and 3 hits for every 4 accesses of 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 3: cache_kgarg

Total of 8 access per 4-loop period.

- **4 hits from A**
- **1 miss and 3 hits from B**

→ ⅛ misses for each 4-loop period

So, Pass 2: Miss rate = 12.5%

```
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: Assembly_imanjarr

Consider the following x86-64 code (Recall that `%cl` is the low-order byte of `%rcx`):

```
# On entry:
#   %rdi = x
#   %rsi = y
#   %rdx = z

4004f0 <mysterious>:
  4004f0:   mov    $0x0,%eax
  4004f5:   lea    -0x1(%rsi),%r9d
  4004f9:   jmp    400510 <mysterious+0x20>
  4004fb:   lea    0x2(%rdx),%r8d
  4004ff:   mov    %esi,%ecx
  400501:   shl    %cl,%r8d
  400504:   mov    %r9d,%ecx
  400507:   sar    %cl,%r8d
  40050a:   add    %r8d,%eax
  40050d:   add    $0x1,%edx
  400510:   cmp    %edx,%edi
  400512:   ja     4004fb <mysterious+0xb>
  400514:   retq
```

Problem 1: Assembly_imanjarr

1) Please fill in the corresponding blanks below to make the C source equivalent to the assembly.

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;
  int e;
  for(i = z ; ; ){
    e = i + 2;
    e = ;
    e = ;
    d = ;
  }
  return ;
}
```

```
4004fb:    lea     0x2(%rdx),%r8d
```

Problem 1: Assembly_imanjarr

1) Please fill in the corresponding blanks below to make the C source equivalent to the assembly.

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;
  int e;
  for(i = z ; ; i++ ){
    e = i + 2;
    e = ;
    e = ;
    d = ;
  }
  return ;
}
```

```
4004fb:   lea    0x2(%rdx),%r8d
```

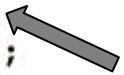
loop end

```
40050d:   add    $0x1,%edx
400510:   cmp    %edx,%edi
400512:   ja     4004fb <mysterious+0xb>
```

Problem 1: Assembly_imanjarr

1) Please fill in the corresponding blanks below to make the C source equivalent to the assembly.

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;
  int e;
  for(i = z ; x > i ; i++ ){
    e = i + 2;
    e = ____ ;
    e = ____ ;
    d = ____ ;
  }
  return ____ ;
}
```



```
4004fb:   lea    0x2(%rdx),%r8d
```

loop end

```
40050d:   add    $0x1,%edx
400510:   cmp    %edx,%edi
400512:   ja     4004fb <mysterious+0xb>
```

edi - edx > 0 same as x > i

Problem 1: Assembly_imanjarr

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;
  int e;
  for(i =  z  ;  x > i  ;  i++  ){
    e = i + 2;
    e =          ;
    e =          ;
    d =          ;
  }
  return         ;
}
```

```
4004fb:   lea    0x2(%rdx),%r8d
```

Problem 1: Assembly_imanjarr

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;
  int e;
  for(i = z ; x > i ; i++ ){
    e = i + 2;
    e = e << y ;
    e =        ;
    d =        ;
  }
  return       ;
}
```

```
4004fb:   lea    0x2(%rdx),%r8d
4004ff:   mov    %esi,%ecx
400501:   shl    %cl,%r8d
```

Problem 1: Assembly_imanjarr

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;
  int e;
  for(i = z ; x > i ; i++ ){
    e = i + 2;
    e = e << y ;
    e = e >> (y - 1) ;
    d =  ;
  }
  return  ;
}
```

```
4004fb:   lea    0x2(%rdx),%r8d
4004ff:   mov    %esi,%ecx
400501:   shl    %cl,%r8d
4004f5:   lea    -0x1(%rsi),%r9d
400504:   mov    %r9d,%ecx
400507:   sar    %cl,%r8d
```

Problem 1: Assembly_imanjarr

```
int mysterious(int x, int y, int z){
  unsigned i;
  int d = 0;                                  4004f0:   mov    $0x0,%eax
  int e;
  for(i =  z  ;  x > i  ;  i++  ){
    e = i + 2;                                4004fb:   lea    0x2(%rdx),%r8d
    e =  e << y  ;                            4004ff:   mov    %esi,%ecx
                                              400501:   shl    %cl,%r8d
    e =  e >> (y - 1)  ;                      4004f5:   lea    -0x1(%rsi),%r9d
                                              400504:   mov    %r9d,%ecx
                                              400507:   sar    %cl,%r8d
    d =  e + d  ;                             40050d:   add    $0x1,%edx
  }
  return  d  ;                                400514:   retq
}
```

Problem 4: struct_saclark

Consider the following structs on an x86-64 Linux machine:

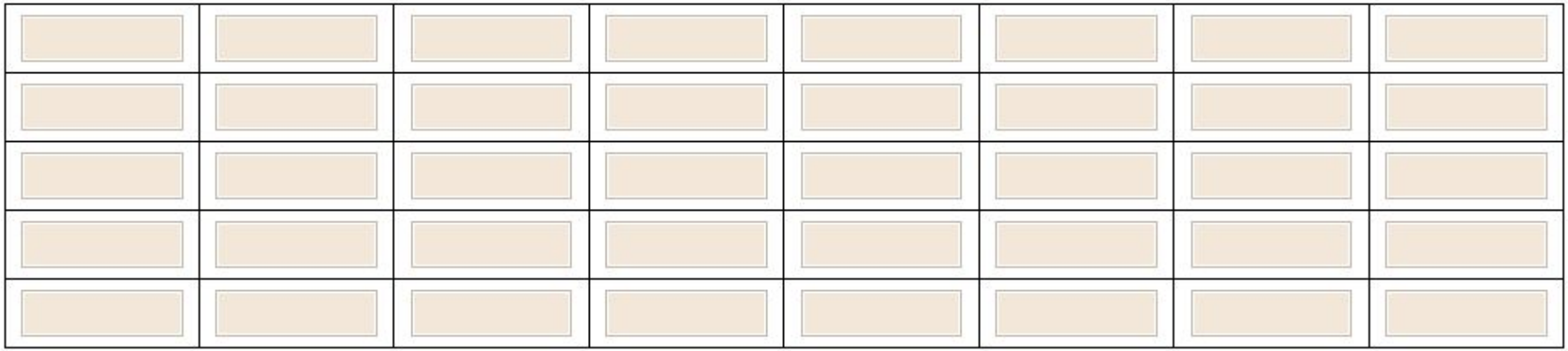
```
struct academic {
    char school[4];
    char major[16];
    int year;
    char gradIn213;
};

struct student {
    struct academic academic;
    short myAge;
    char favoriteAnimal[12];
    float favoriteNumber;
};
```

Problem 4: struct_saclark

How would the struct academic would be allocated?

- Use the first letter to indicate the field
- Use an "x" to indicate padding within the struct
- Use "-" to indicate any extra space not needed to allocate the struct.



Problem 4: struct_saclark

Question: Why do we need the three bytes of padding at the end of the struct?

s	s	s	s	m	m	m	m
m	m	m	m	m	m	m	m
m	m	m	m	y	y	y	y
g	x	x	x	-	-	-	-
-	-	-	-	-	-	-	-

Answer: The struct must be aligned to multiples of 4 because of the int field

Problem 4: struct_saclark

1. How many bytes would the academic struct need if it were rearranged in a way to reducing internal padding?

2. How many bytes does the student struct currently take in memory?

3. How many bytes would the student struct need it were rearranged in a way to reducing internal padding?

4. The char school[4] field has been removed from the academic struct. How many bytes would the academic struct need if it were rearranged to reducing internal padding?

Problem 4: struct_saclark

1. How many bytes would the academic struct need if it were rearranged in a way to reducing internal padding?
   28 bytes
2. How many bytes does the student struct currently take in memory?
   48 bytes
3. How many bytes would the student struct need it were rearranged in a way to reducing internal padding?
   48 bytes
4. The char school[4] field has been removed from the academic struct. How many bytes would the academic struct need if it were rearranged to reducing internal padding?
   24 bytes

Problem 4: struct_saclark

6) Now, consider the following assembly function compiled on an x86-64 Linux machine:

```
0x400625 <main+149>     movzwl 0x1c(%rax),%eax
0x40062a <main+154>     mov    %eax,%esi
0x40062c <main+156>     mov    $0x400740,%edi
0x400631 <main+161>     mov    $0x0,%eax
0x400636 <main+166>     callq  0x4003f8 <printf@plt>
```

Assume that at the beginning of the function we know that %rax points to a student struct that has been allocated in memory.
What field of the student struct does the program print?

Problem 4: struct_saclark

0x	school	school	school	school	major	major	major	major
0x8	major	major	major	major	major	major	major	major
0x10	major	major	major	major	year	year	year	year
0x18	grade	x	x	x	age	age	animal	animal
0x20	animal	animal	animal	animal	animal	animal	animal	animal
0x28	animal	animal	x	x	number	number	number	number
0x30								

The assembly will print out age
movzwl => move zero extended, a word to a long
from the address of rax + 0x1c

Some extra questions :)

Appendix

Problem: Arrays

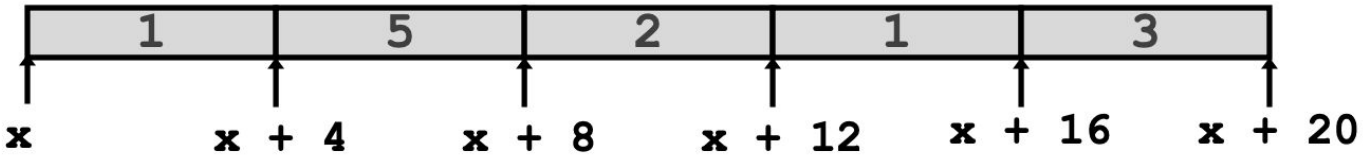
IMPORTANT POINTS + TIPS:

- ***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: Arrays

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

```
int val[5];
```

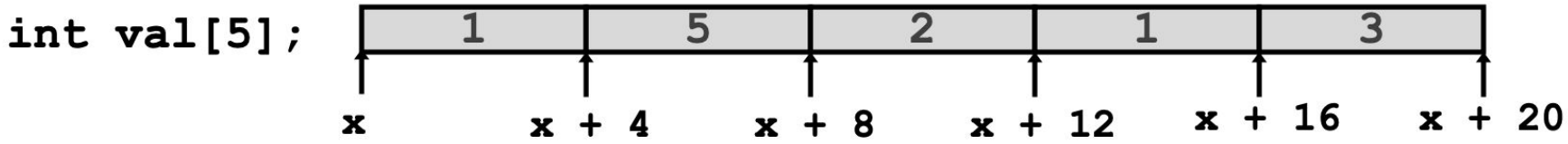


- `A` can be used as the pointer to the first array element: `A[0]`

	Type	Value
`val`		
`val[2]`		
`*(val + 2)`		
`&val[2]`		
`val + 2`		
`val + i`		

Problem: Arrays

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

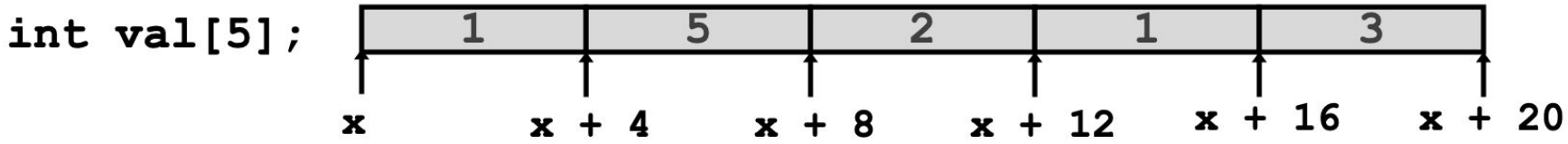


- `A` can be used as the pointer to the first array element: `A[0]`

	Type	Value
`val`	`int *`	`x`
`val[2]`	`int`	`2`
`*(val + 2)`	`int`	`2`
`&val[2]`	`int *`	`x + 8`
`val + 2`	`int *`	`x + 8`
`val + i`	`int *`	`x + (4 * i)`

Problem: Arrays

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



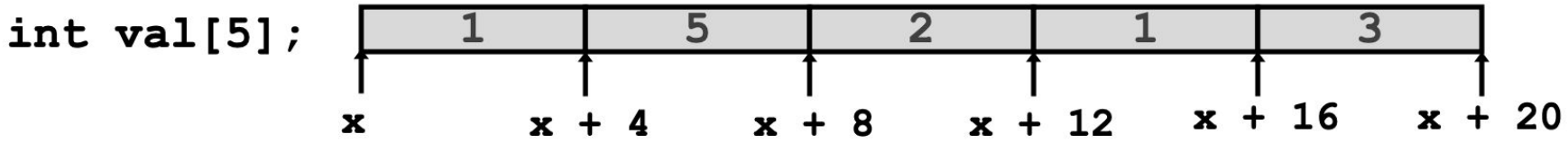
- A can be used as the pointer to the first array element: `A[0]`

	Type	Value
`val`	`int *`	`x`
`val[2]`	`int`	`2`
`*(val + 2)`	`int`	`2`
`&val[2]`	`int *`	`x + 8`
`val + 2`	`int *`	`x + 8`
`val + i`	`int *`	`x + (4 * i)`

Accessing methods:
- *val[index]*
- **(val + index)*

Problem: 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`	`int *`	`x`
`val[2]`	`int`	`2`
`*(val + 2)`	`int`	`2`
`&val[2]`	`int *`	`x + 8`
`val + 2`	`int *`	`x + 8`
`val + i`	`int *`	`x + (4 * i)`

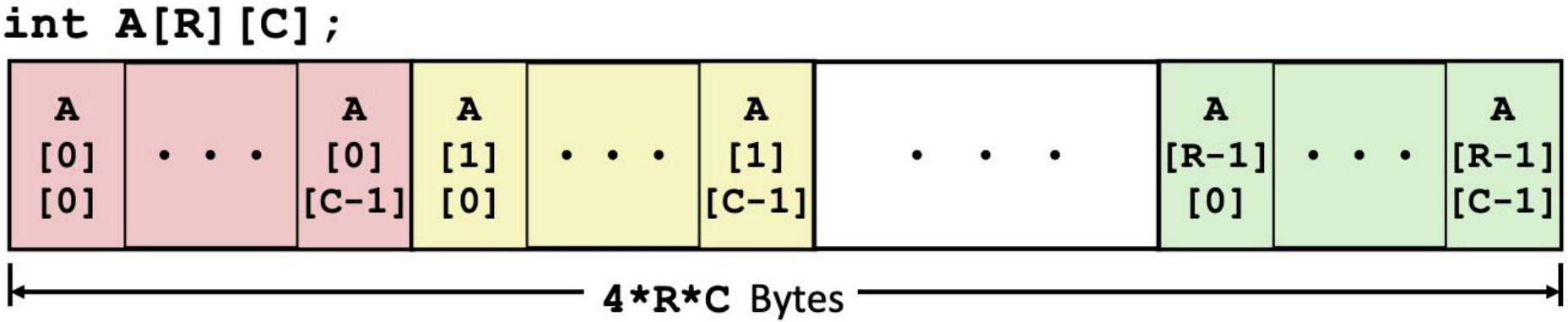
Accessing methods:
- *val[index]*
- **(val + index)*

Addressing methods:
- *&val[index]*
- *val + index*

Problem: Arrays

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

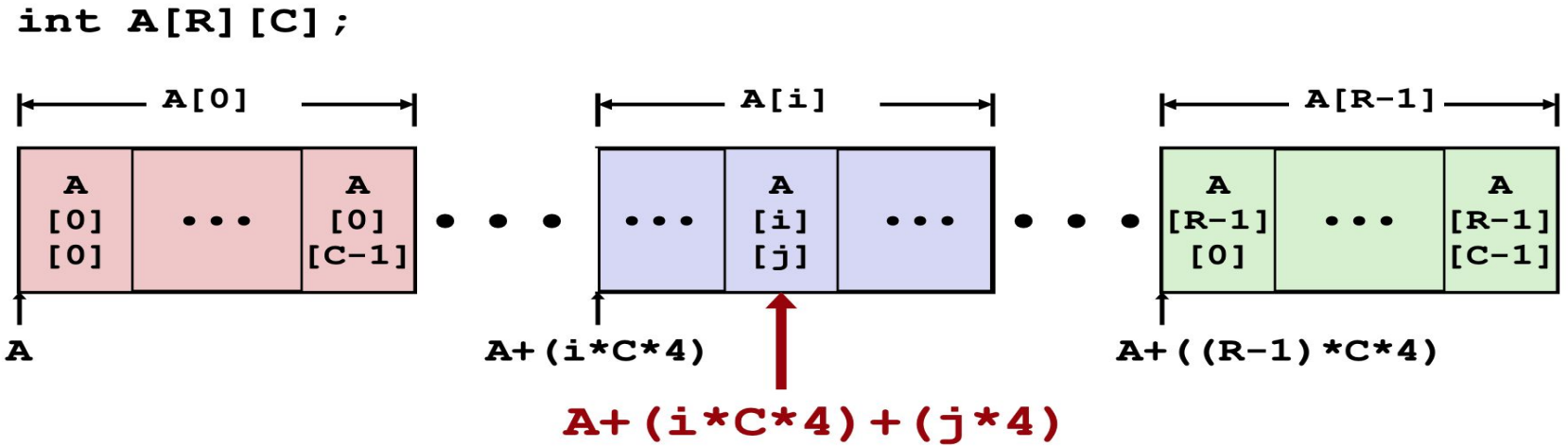
- Declared: `T A[R][C]`
- Contiguous chunk of space (think of multiple arrays lined up next to each other)



Problem: 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



Problem: Arrays

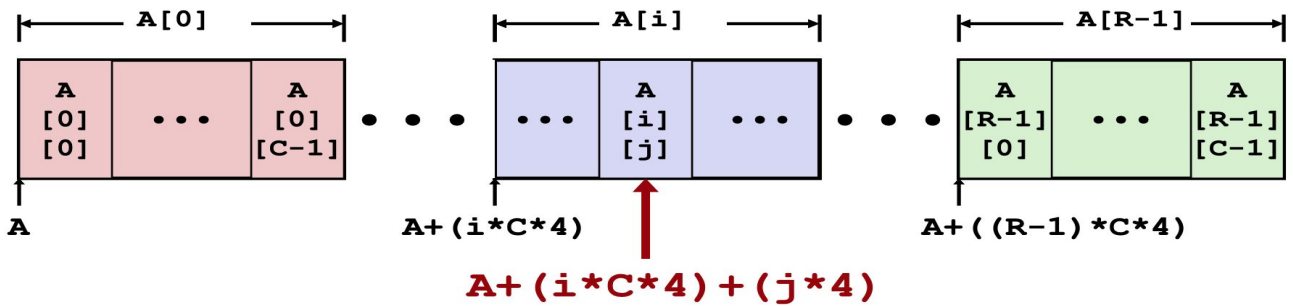
Nested indexing rules (for your cheatsheet and/or big brain):

`A[i][j]` is element of type *T*, which requires *K* bytes

Address `A + i * (C * K) + j * K`

`= A + (i * C + j) * K`

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



Problem: Arrays

Consider accessing elements of `A`....

	Compiles	Bad Deref?	Size (bytes)
`int A1[3][5]`			
`int *A2[3][5]`			
`int (*A3)[3][5]`			
`int *(A4[3][5])`			
`int (*A5[3])[5]`			

Problem: Arrays

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

	Compiles	Bad Deref?	Size (bytes)
`int A1[3][5]`	Y	N	3*5*4 = 60
`int *A2[3][5]`			
`int (*A3)[3][5]`			
`int *(A4[3][5])`			
`int (*A5[3])[5]`			

Problem: Arrays

Consider accessing elements of `A`....

	Compiles	Bad Deref?	Size (bytes)
`int A1[3][5]`	Y	N	`3*5*(4) = 60`
`int *A2[3][5]`	Y	N	`3*5*(8) = 120`
`int (*A3)[3][5]`			
`int *(A4[3][5])`			
`int (*A5[3])[5]`			

Problem: Arrays

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

	Compiles	Bad Deref?	Size (bytes)
`int A1[3][5]`	Y	N	`3*5*(4) = 60`
`int *A2[3][5]`	Y	N	`3*5*(8) = 120`
`int (*A3)[3][5]`	Y	N	`1*8 = 8`
`int *(A4[3][5])`			
`int (*A5[3])[5]`			

Problem: Arrays

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

	Compiles	Bad Deref?	Size (bytes)
`int A1[3][5]`	Y	N	`3*5*(4) = 60`
`int *A2[3][5]`	Y	N	`3*5*(8) = 120`
`int (*A3)[3][5]`	Y	N	`1*8 = 8`
`int *(A4[3][5])`	Y	N	`3*5*(8) = 120`
`int (*A5[3])[5]`			

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

Problem: Arrays

Consider accessing elements of A....

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

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

Problem: Arrays

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

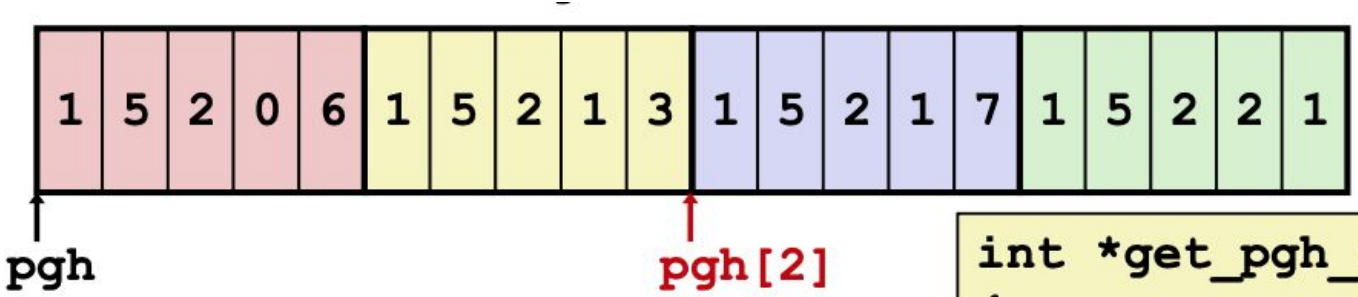
Problem: Arrays

Decl	An			*An			**An		
	Cmp	Bad	Size	Cmp	Bad	Size	Cmp	Bad	Size
`int A1[3][5]`	Y	N	60	Y	N	20	Y	N	4
`int *A2[3][5]`	Y	N	120	Y	N	40	Y	N	8
`int (*A3)[3][5]`	Y	N	8	Y	Y	60	Y	Y	20
`int *(A4[3][5])`	Y	N	120	Y	N	40	Y	N	8
`int (*A5[3])[5]`	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: Arrays

Sample assembly-type questions

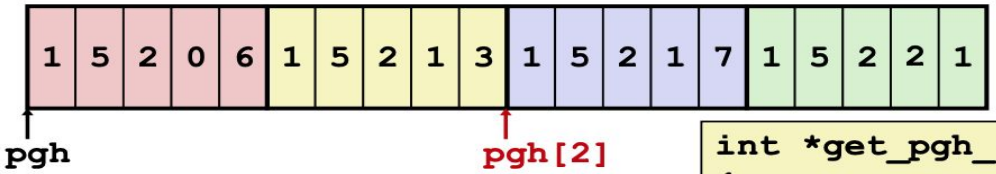



```
int *get_pgh_zip(int index)
{
  return pgh[index];
}
```

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

Problem: Arrays

Nested Array Row Access Code



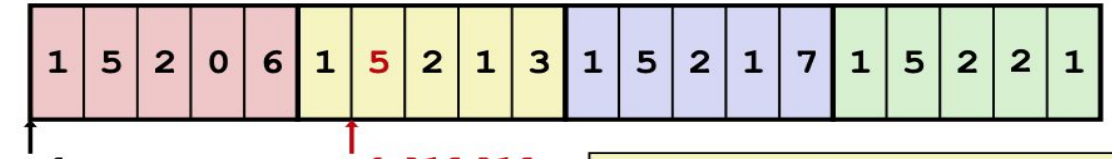

```
int *get_pgh_zip(int index)
{
  return pgh[index];
}
```

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

- **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: 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: `pgh + 20*index + 4*dig`
    `= pgh + 4*(5*index + dig)`

Problem: 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: 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: 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: 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: 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 => 1.1111

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

E = 1 => exponent = 1 + bias = 2

Problem: 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 => fraction bits are 1111

exponent bits are 10

Problem: 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 <= excess bit

Problem: 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 <= excess bit

Guard bit = 1

Round bit = 1

Round up! (add 1 to the fraction bits)

Problem: 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 <= excess bit

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

Problem: 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 <= Infinity!

Problem: 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: 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: 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 => 1.11

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

E = -1 => exponent = -1 + bias = 0

Problem: 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 => 1.11 <= (We assumed M was in the range [1.0, 2.0). Need to update the value of M)

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

E = -1 => exponent = -1 + bias = 0 <= denormalized

Problem: 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 => 0.111 <= M should be in the range [0.0, 1.0) for denormalized numbers so we divide it by 2

exp = 0

Problem: 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 => Fraction bits = 111

exp bits = 00

Result = 1 00 111

Problem: 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: 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

exp = 2 => E = exp - bias = 1 (normalized)

M = 1.101 (between 1 and 2 since it is normalised)

Result = 2*1.101 = 2*(13/8) = 13/4

Problem: 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

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 → 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	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	3	M
2	1010 0111	2	M
3	1101 1001	1	M
4	1011 1100	3	H
5	1011 1001	3	H

Bonus!

C. How many sets are there? 2 bits → 4 sets
How big is each cache line? 4 bits → 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