

# Midterm II Review

## ICS312 - Spring 2009 Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)

## What to Study for the Midterm?

- Material to review
  - Quizzes
  - Homework solutions
    - Including my assembly code if you had issues with your assignment
  - Lecture notes
    - Bit operations
    - Subprograms
    - Floating Point Arithmetic
  - Reading assignments in the textbook
    - Especially the examples

## What Questions to Expect

- Bit operations
  - Quiz-like questions
  - What output does this code produce?
  - Write a piece of code using bit operations to do something
- Subprograms
  - Quiz-like questions
  - Write a subprogram
  - What does the stack look like?
- FP Arithmetic
  - Quiz-like questions
  - Conversion
  - No FP assembly code

+ 1 or 2 "mystery programs"

## Bit Operations

- What does this code print out?

```

mov     ax, 0035Dh
sal     ah, 6
sar     ax, 2
shr     ah, 2
xor     ah, al
not     ah
sar     ah, 4
movsx   eax, ah
call    print_int
  
```

## Bit Operations

		EAX
mov	ax, 0035Dh	???????? ???? 0000011 01011101
sal	ah, 6	???????? ???? 11000000 01011101
sar	ax, 2	???????? ???? 11110000 00010111
shr	ah, 2	???????? ???? 00111100 00010111
xor	ah, al	???????? ???? 00101011 00010111
not	ah	???????? ???? 11010100 00010111
sar	ah, 4	???????? ???? 11111101 00010111
movsx	eax, ah	11111111 11111111 11111111 11111101
call	print_int	

FFFFFFFFFF

flip: 00000002     +1: 00000003  
the code prints out: -3

## The Carry bits and Shifts

- Remember that when you do a shift, the last bit shifted out ends up in the carry bit
- Remember that the **adc** instruction is very useful as it adds the carry bit to a register:
 

```

adc  eax, 12    ; eax += 12 + carry
adc  al, 0      ; al += 0 + carry
  
```
- Make sure you understand how to do HW#6 with bit operations
  - Ask me for my solution if needed
- Let's review a few simple things with bitmasks...

## Example #1

- Code to flip the  $n^{\text{th}}$  bit of EAX, counting from right to left with the rightmost bit, where  $n$  is stored in  $cl$

## Example #1

- Code to flip the  $n^{\text{th}}$  bit of EAX, counting from right to left with the rightmost bit, where  $n$  is stored in  $cl$

```
mov  ebx, 1
shl  ebx, cl ; only cl works here
xor  eax, ebx
```

## Example #2

- Create in  $eax$  a 32-bit bitmask that looks like 0's followed  $n$  1's, where  $n$  is stored in  $cl$

## Example #2

- Create in  $eax$  a 32-bit bitmask that looks like 0's followed  $n$  1's, where  $n$  is stored in  $cl$

```
mov  eax, 0FFFFFFFFh
shl  eax, cl
not  eax
```

## Example #3

- Create in  $eax$  a 32-bit bitmask that contains  $n$  1's, followed by  $32-2n$  0's, followed by  $n$  1's, where  $n$  is stored in  $cl$

## Example #3

- Create in  $eax$  a 32-bit bitmask that contains  $n$  1's, followed by  $32-2n$  0's, followed by  $n$  1's, where  $n$  is stored in  $cl$

```
mov  eax, 0FFFFFFFFh
shl  cl, 1 ; multiply cl by 2
shl  eax, cl
shr  cl, 1 ; divide cl by 2
ror  eax, cl
not  eax
```

## Example #4

- Count int ebx the number of odd-numbered bits set to zero in eax (the right most bit is bit 0)

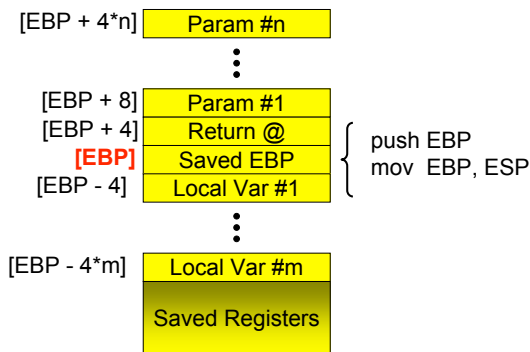
```

xor ebx, ebx
mov ecx, 16
not eax
begin_loop: shr eax, 2
            adc ebx, 0
            loop begin_loop
    
```

## Subprograms: The C Convention

- The Caller puts the arguments on the stack in reverse order
- The Caller executes the "call" instruction
  - which puts the return @ on the stack
- The Callee puts EBP on the stack
- The Callee sets EBP to ESP
  - To access parameters and local vars consistently even if ESP changes
- The Callee reserves space on the stack for local variables
- The Callee may save registers on the stack
- The Callee does its thing
- The Callee restores saved registers if any
- The Callee sets EAX to the return value is needed
- The Callee removes space for local variables
- The Callee restores EBP
- The Callee executes "ret"
  - Which pops the return @ off the stack
- The Caller removes the parameters from the stack

## Subprograms: The C Convention



## Write a function

- Write the following function in assembly so that no register value is destroyed by the call to the function, but for EAX of course (use the C convention and a "straight" translation without optimizations):

```

int f(int a, int b) {
    int z;
    z = a + b;
    return z+2;
}
    
```

## Write a function

```

int f(int a, int b) {
    int z;
    z = a + b;
    return z+2;
}
    
```

```

f:
push    ebp
mov     ebp, esp
sub     esp, 4
push   ebx
mov     ebx, [ebp+8]
add     ebx, [ebp+12]
mov     [ebp-4], ebx
mov     eax, [ebp-4]
add     eax, 2
pop     ebx
add     esp, 4
pop     ebp
ret
    
```

## Write a function

- Same Question, but saving ALL registers (paranoid)

```

int f(int a, int b) {
    int z;
    z = a + b;
    return z+2;
}
    
```

## Write a function

```
int f(int a, int b) {
    int z;
    z = a + b;
    return z+2;
}
```

```
segment .bss:
    retvalue dd
f:
    push    ebp
    mov     ebp, esp
    sub     esp, 4
    pusha
    mov     ebx, [ebp+8]
    add     ebx, [ebp+12]
    mov     [ebp-4], ebx
    add     ebx, 2
    mov     [retvalue], ebx
    popa
    mov     eax, [retvalue]
    add     esp, 4
    pop     ebp
    ret
```

## The Stack

- What does the stack look like right before the program below prints "hello" (main calls f(2)), assuming that no register needs to be saved but EBP

```
void f(int n) {
    int z;
    g(2, n+1);
    ...
}

void g(int x, int y) {
    if (x == 0) {
        printf("Hello\n");
    } else {
        g(x-1, y+1);
    }
}
```

## The Stack

```
void f(int n) {
    int z;
    g(2, n+1);
    ...
}

void g(int x, int y) {
    if (x == 0) {
        printf("Hello\n");
    } else {
        g(x-1, y+1);
    }
}
```

2
return @ (to main)
saved EBP (for main)
z
3
2
return @ (to f)
saved EBP (for f)
4
1
return @ (to g)
saved EBP (for g)
5
0
return @ (to g)
saved EBP (for g)

## FP Arithmetic

- Quiz-like questions about the IEEE encoding
- Conversion of 43.21 into binary (only 4 bits after the floating point)

$$\begin{aligned}
 43 &= 2 * 21 & + 1 \\
 21 &= 2 * 10 & + 1 \\
 10 &= 2 * 5 & + 0 \\
 5 &= 2 * 2 & + 1 \\
 2 &= 2 * 0 & + 0
 \end{aligned}$$



$$\begin{aligned}
 .21 * 2 &= 0.42 \\
 .42 * 2 &= 0.84 \\
 .84 * 2 &= 1.68 \\
 .68 * 2 &= 1.36
 \end{aligned}$$



Answer: **1011.0011**

Questions?