

# Midterm I 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
    - Up to and **NOT** including “bit operations”
  - Reading assignments in the textbook
    - Especially the examples

## What Questions to Expect

- Questions like in the homework assignments
  - 2's complement
  - OF and CF flags after arithmetic operations
  - Memory Layout
  - How does a program modify memory
- “Write a fragment of assembly code that does ....”
  - nothing different from what you've done in the homework assignments
- “Here is a program, tell me what it does”

## Numbers

- Let's just do 2's complement once more
- What's -194 in 2-byte hex?
- What's +36 in 4-byte hex?
- What's F3, interpreted as a signed number?
- What's F3, interpreted as an unsigned number?

## Signed / Unsigned

- The “add” and “sub” operations do the right thing as long as you're consistent in your interpretations of the operands and of the result
- There is a “mul” instruction for unsigned numbers, and an “imul” instruction for signed numbers
- There is a “div” instruction for unsigned numbers, and an “idiv” instruction for signed numbers

## The div instruction

- If src is a 32-bit quantity:
  - EDX:EAX is divided by src
  - quotient stored in EAX
  - remainder stored in EDX
- Don't forget to set EDX to zero!

## The imul instruction

Will not overflow (although the overflow bit may be set)

dst	src1	src2	action
	reg/mem8		AX = AL * src1
	reg/mem16		DX:AX = AX * src1
	reg/mem32		EDX:EAX = EAX * src1
reg16	reg/mem16		dst *= src1
reg32	reg/mem32		dst *= src1
reg16	immed8		dst *= immed8
reg32	immed8		dst *= immed8
reg16	immed16		dst *= immed16
reg32	immed32		dst *= immed32
reg16	reg/mem16	immed8	dst = src1*src2
reg32	reg/mem32	immed8	dst = src1*src2
reg16	reg/mem16	immed16	dst = src1*src2
reg32	reg/mem32	immed32	dst = src1*src2

## How to tell what will overflow?

- Figure out how many bits we're talking about
- Figure out the (decimal) range of acceptable values
  - Depending on whether we're thinking of signed or unsigned numbers
  - E.g., [-128; +127] for 1-byte signed
  - E.g., [0; 255] for 1-byte unsigned
- Figure out whether the result of an operations falls within the allowable range or not

## Size modification

- When moving a X-byte quantity to a Y-byte quantity, with X > Y, you just drop the extra bits on the left
  - May lead to consistent results
  - May lead to inconsistent results
- To increase size one must use
  - Movzx: adds zeros to the left
    - Good to extend the size of unsigned integers
  - Movsx: adds replicas of the sign-bit to the left
    - Good to extend the size of signed integers

## How to Detect Overflow in Programs?

UNSIGNED → CARRY FLAG

- jc, jnc

SIGNED → OVERFLOW FLAG

- jo, jno

## A Few Examples

- 1-byte: AF + 70
  - Is the Carry Bit set?
  - Is the Overflow Bit set?
  - What would print\_int print out?
- 2-byte: FF12 + 7FFE
  - Is the Carry Bit set?
  - Is the Overflow Bit set?
  - What would print\_int print out?
- 1-byte: AF + 12
  - Is the Carry Bit set?
  - Is the Overflow Bit set?
  - What would print\_int print out?

## If-then-Else

- The basis of an if-then-else:
 

```

cmp XXX
jXX thenblock
; else block
jmp endif
thenblock:
; then block
endif:
            
```

## If-then-else

- How about: `if ((eax == 0) && (ebx == 1))?`

```
    cmp  eax, 0
    jnz  elseblock
    cmp  ebx, 1
    jnz  elseblock
    ; thenblock
    jmp  endif
elseblock:
    ; elseblock
endif:
```

## If-then-else

- How about: `if ((eax == 0) || (ebx == 1))?`

```
    cmp  eax, 0
    jz   thenblock
    cmp  ebx, 1
    jz   thenblock
    ; elseblock
    jmp  endif
thenblock:
    ; thenblock
endif:
```

## Conditional Branches

cmp x, y			
signed		unsigned	
Instruction	branches if	Instruction	branches if
JE	x = y	JE	x = y
JNE	x != y	JNE	x != y
JL, JNGE	x < y	JB, JNAE	x < y
JLE, JNG	x <= y	JBE, JNA	x <= y
JG, JNLE	x > y	JA, JNBE	x > y
JGE, JNL	x >= y	JAE, JNB	x >= y

## Loops

- Doing: `for (i=3; i<10; i+=2) { stuff }`

```
    mov  ebx, 3
loop1: ; stuff
    add  ebx, 2
    cmp  ebx, 10
    jb   loop1
```

## The loop instruction

- Only if
  - You want ecx to be the loop index
  - You want the index to go from some positive value down to zero in increments of 1

```
    mov  ecx, 20
loop1: stuff
    loop loop1
```

## The Memory Layout

- Given a `.data` segment
  - What do bytes look like?
    - Little Endian / Big Endian
  - What do a bunch of instructions do to the data?
- Here is `_yet another_` example

```
L1      db      "abc"
L2      dd      0AABCCAAh
L3      time 4      dw 25
L4      db      "d", 0

mov  eax, L2
add  eax, 3
mov  word [eax], 23
mov  ebx, L3
mov  ebx, [ebx]
movsx eax, bh
mov  [L3], eax
```

## Mystery Program Example

```
L    resw 10
...
    mov     ebx, L
    add     ebx, 18
    mov     ecx, 0
loop1: movsx  word eax, [ebx]
    add     ecx, eax
    sub     ebx, 2
    cmp     ebx, L
    jnz    loop1
```

## Questions?