Midterm Review

ICS312

Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)

What to Expect?

- **Open note/computer**
- On Laulima (during lecture period, timed)
  - A single question with the whole exam as a PDF, a single answer as text
- Material to review
  - Practice Quizzes
  - Homework solutions
    - Request solutions via e-mail!
  - Lecture notes
  - Reading assignments in the textbook
    - Especially the examples

Material Covered

- All modules from “Getting Started” to “Bit Operations” (inclusive)

What Questions to Expect

- A few quiz-like questions
  - Study for them by pretending you're teaching the course
  - If you have to go through your notes to answer them all, you'll waste too much time
- Questions like in the homework assignments
  - 2's complement
  - Size modification
  - OF and CF flags after arithmetic operations
  - Memory Layout
  - How does a program modify memory?
  - How does one implement control structures
  - How does one use bitwise operations
- "Write a (small) 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 or fix it"

Numbers

- What’s -194 decimal in 2-byte hex?
- What’s +36 decimal in 4-byte hex?
- What’s F3 in decimal, interpreted as a signed number?
- What’s F3 in decimal, interpreted as an unsigned number?

Numbers

- What's -194 in 2-byte hex?
  - 194d = 160d + 34d = 12d*16d + 2d = 00C2h
  - flip: FF3D
  - add one: FF3E
- What's +36 in 4-byte hex?
  - 36d = 24h
- What's F3, interpreted as a signed number?
  - It's negative, so flip: 0C, add one: 0D
  - It's -13d
- What's F3, interpreted as an unsigned number?
  - 15d*16d + 3d = 243d

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

Signed / Unsigned

- The confusing thing:
  - The microprocessor has no notion of whether values are signed or unsigned
  - The programmer has that notion
- If you show me a piece of code written by somebody else, the only way I can tell whether the programmer thinks of numbers as signed or unsigned:
  - Are there any imul? idiv?
  - Does the programmer check to OF flag?
  - Which conditional branch instructions are used?
  - Are there movsx or movzx instructions?

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 Detect Overflow in Assembly Programs?

UNSIGNED → CARRY FLAG

- jc, jnc

SIGNED → OVERFLOW FLAG

- jo, jno

In both cases we talk of “overflow”!

CF and OF: set or not set?

- Is CF set?
  - Option #1: Do the hex addition and see if you have a left-over carry, in which case CF=1
    - That carry would require an additional bit
  - Option #2: Reason about the numbers as unsigned and determine whether there will be a carry, in which case CF=1, without computing the whole addition
- Is OF set?
  - Think of the two numbers as signed
  - If they are of different sign, then OF=0 no matter what
  - If they are of the same sign, then OF=1 only if the sign of the result makes no sense
    - Option #1: Do the hex addition and look at the sign of the result
    - Option #2: Estimate the sign of the result based on magnitudes

A Few Examples

- 1-byte: AF + 70
  - Is the Carry Bit set?
  - Is the Overflow Bit set?
- 2-byte: FF12 + 7FFE
  - Is the Carry Bit set?
  - Is the Overflow Bit set?
- 1-byte: AF + 84
  - Is the Carry Bit set?
  - Is the Overflow Bit set?

A Few Examples

- 1-byte: AF + 70
- Carry flag
  - The “human” method
    - AF is large positive, 70 is large positive, we “overflow”, CF is set
  - The “brute-force” method
    - AF + 70 = 11F, CF is set
- Overflow flag
  - AF is negative, 70 is positive, no overflow

A Few Examples

- 2-byte: FF12 + 7FFE
- CF flag
  - The “human” method:
    - We add two huge numbers together: CF is set
  - The “brute-force” method
    - FF12 + 7FFE = 17F10 and a carry
- OF flag
    - The two numbers are of opposite signs: OF is not set

A Few Examples

- 1-byte: AF + 84
- CF Flag
  - The “human” method
    - A + 8 > F: carry
  - The “microprocessor” method
    - AF + 84 =   133, a carry is generated
- OF flag
  - Both number are negative, so we may have overflow
  - The result is 33, which is positive, so OF is set

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 numerically consistent results
  - May lead to numerically 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

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

If-then-Else

- The basis of an if-then-else:

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

Example

- if ((eax == 0) && (ebx == 1))?

Example

- if ((eax == 0) && (ebx == 1))?

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

Other Example

- if ((eax == 0) || (ebx >= 1))?   (signed)

Other Example

- if ((eax == 0) || (ebx >= 1))?   (signed)

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

Loops

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

Loops

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

```
        mov  ebx, 3
loop1:  ; body
        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:  body
        loop   loop1
```

The Memory Layout

- Little Endian

```
L1        db        "a, "bc"
L2        dd        0AABBCCAAh
L3        times 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
```

The Memory Layout

- Little Endian

L1	db	“a”,”bc”
L2	dd	0AABBCCAAh
L3	times 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
```

61 62 63 AA CC BB AA E7 FF E7 FF E7 FF E7 FF 64 00

The Memory Layout

- Little Endian

```
L1      db        "a","bc"
L2      dd        0AABBCCAAh
L3      times 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
```

61 62 63 AA CC BB AA E7 FF E7 FF E7 FF E7 FF 64 00

The Memory Layout

- Little Endian

L1	db	“a”,”bc”
L2	dd	0AABBCCAAh
L3	times 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
```

61 62 63 AA CC BB AA E7 FF E7 FF E7 FF E7 FF 64 00

The Memory Layout

- Little Endian

L1	db	"a","bc"
L2	dd	0AABBCCAAh
L3	times 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
```

61 62 63 AA CC BB 17 00 FF E7 FF E7 FF E7 FF 64 00

The Memory Layout

- Little Endian

```
L1      db          "a","bc"
L2      dd          0AABBCCAAh
L3      times 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
```

61 62 63 AA CC BB 17 00 FF E7 FF E7 FF E7 FF 64 00

The Memory Layout

- Little Endian

```
L1    db        "a","bc"
L2    dd        0AABBCCAAh
L3    times 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
```

61 62 63 AA CC BB 17 00 FF E7 FF E7 FF E7 FF 64 00

ebx = FF E7 FF 00

The Memory Layout

- Little Endian

L1	db	“a”,”bc”
L2	dd	0AABBCCAAh
L3	times 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
```

61 62 63 AA CC BB 17 00 FF E7 FF E7 FF E7 FF 64 00

ebx = FF E7 FF 00
eax = FF FF FF FF

The Memory Layout

- Little Endian

```
L1    db        "a","bc"
L2    dd        0AABBCCAAh
L3    times 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
```

61 62 63 AA CC BB 17 FF FF FF FF E7 FF E7 FF 64 00

ebx = FF E7 FF 00
eax = FF FF FF FF

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

Instruction	Operands	EAX			
mov	ax, 0035Dh	????????	????????	00000011	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				

FFFFFFFD

flip: 00000002 +1: 00000003

the code prints out: -3

Counting bits

- The only real example we've seen is how to count bits (set to 1) in some register
- Should we look at this again?

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
```

- Let’s review a few simple things with bitmasks...

Example #1

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

Example #1

- Code to flip the n^{th} bit of EAX, counting from right to left from 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 by 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
```

Mystery Program

- In close-notes exams, I often have “mystery program” questions
- I don’t do this for open-notes/open-computer exams
- But it may still good practice for you to see if you can do the following mystery program problems…

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
```

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
```

computes the sum of the elements in an array of 10 2-byte values, which starts at address L

Yet another mystery program..

```
segment .data
        msg     db "Enter an integer: ",
0
        success db "Success", 0

segment .text

       . . .
again:
        mov     eax, msg
        call    print_string
        call    read_int
        and     al, 0
        cmp     eax, 0
        jnz     again

        mov     eax, success
        call    print_string
        call    print_nl
      . . .
```

Yet another mystery program..

```
segment .data
        msg     db "Enter an integer: ",
0
        success db "Success", 0

segment .text

       . . .
again:
        mov     eax, msg
        call    print_string
        call    read_int
        and     al, 0
        cmp     eax, 0
        jnz     again

        mov     eax, success
        call    print_string
        call    print_nl
      . . .
```

repeatedly asks the user for an integer until that integer is between 0 and 255, then prints “Success”.

Other Mystery Program

```
segment .bss
        L        resb   10

segment .text

        . . .
        mov      ebx, L
        add      ebx, 9

        mov      ecx, 0

for:
        mov      dl, [ebx]
        shr      dl, 1
        jc       nope
        inc      ecx
nope:
        dec      ebx
        cmp      ebx, L
        jnz      for

        mov      eax, ecx
        call     print_int
        call     print_nl
         . . .
```

Other Mystery Program

```
segment .bss
        L       resb   10

segment .text

        . . .
        mov     ebx, L
        add     ebx, 9

        mov     ecx, 0

for:
        mov     dl, [ebx]
        shr     dl, 1
        jc      nope
        inc     ecx
nope:
        dec     ebx
        cmp     ebx, L
        jnz     for

        mov     eax, ecx
        call    print_int
        call    print_nl
         . . .
```

prints the number of **even** values among the first 10 1-byte values at address L

Yet another mystery program..

```
segment .bss
        L       resd 10

segment .text
       . . .
        mov     ecx, 0
        mov     edx, 10
        mov     ebx, L
for:
        mov     eax, [ebx]
        shr     eax, 1
        jc      no
        shl     eax, 1
        cmp     eax, ecx
        jle     no
        mov     ecx, eax
no:
        add     ebx, 4
        dec     edx
        jnz     for

        mov     eax, ecx
        call    print_int
        call    print_nl
       . . .
```

Yet another mystery program..

```
segment .bss
        L       resd 10

segment .text
       . . .
        mov     ecx, 0
        mov     edx, 10
        mov     ebx, L
for:
        mov     eax, [ebx]
        shr     eax, 1
        jc      no
        shl     eax, 1
        cmp     eax, ecx
        jle     no
        mov     ecx, eax
no:
        add     ebx, 4
        dec     edx
        jnz     for

        mov     eax, ecx
        call    print_int
        call    print_nl
       . . .
```

prints the largest **even** values among the first 10 4-byte values at address L

Questions….