5. Arithmetic and Logic Functions

Background

Arithmetic, shift/rotate, and logical operations are included in every microprocessor instruction set. This lab examines the x86 capabilities in these areas.

Objectives

A. Addition/Subtraction of Extended Numbers

B. Multiplication/Division of unsigned numbers

C. Multiplication, Division of signed numbers

D. Use of Boolean logic operations.

E. Use of shift/rotate instructions.

Pre-Lab

A. Read the description of the mul, imul, div, idiv instructions, and shift/rotate instructions in Sections 7.1 through 7.4 of the Irvine Textbook. Read the descriptions of the Boolean operations in Section 6.1 of Irvine.

B. What is the 64-bit result of the following sum (write in HEX):
`10F701398034AB23 h + 25C28138D292FF7A h = _____`

C. Give the 16 bit HEX result of the following 8 bit UNSIGNED multiplication:
1A * F0 = ____ . Verify your result by converting all operands and result to their unsigned decimal equivalents.

D. Give the 16 bit HEX result of the following 8 bit SIGNED multiplication:
1A * F0 = ____ . Verify your result by converting all operands and result to their unsigned decimal equivalents.

E. Give the 8 bit Quotient and Remainder of the following 8 bit UNSIGNED division:
23AB / 9C = ____(quotient), _____ (remainder) . Verify your result by converting all operands and result to their unsigned decimal equivalents.

F. Give the 8 bit Quotient and Remainder of the following 8 bit UNSIGNED division:
23AB / 9C = ____(quotient), _____ (remainder) . Verify your result by converting all operands and result to their unsigned decimal equivalents.

G. Label the bits of AL as $B_7B_6B_5B_4B_3B_2B_1B_0$. If the instruction "ror al,4" is executed, what is the result in AL? (use $B_7 \ldots B_0$, '1', or '0' to describe each bit of result).

H. Label the bits of AL as $B_7B_6B_5B_4B_3B_2B_1B_0$. If the instruction "and al, AAh" is executed, what is the result in AL? (use $B_7 \ldots B_0$, '1', or '0' to describe each bit of result).

I. Label the bits of AL as $B_7B_6B_5B_4B_3B_2B_1B_0$. If the instruction "or al, AAh" is executed, what is the result in AL? (use $B_7 \ldots B_0$, '1', or '0' to describe each bit of result).

J. Label the bits of AL as $B_7B_6B_5B_4B_3B_2B_1B_0$. If the instruction "shl al, 4" is executed, what is the result in AL? (use $B_7 \ldots B_0$, '1', or '0' to describe each bit of result).

K. Label the bits of AL as $B_7B_6B_5B_4B_3B_2B_1B_0$. If the instruction "shr al, 4" is executed, what is the result in AL? (use $B_7 \ldots B_0$, '1', or '0' to describe each bit of result).

L. Label the bits of AL as $B_7B_6B_5B_4B_3B_2B_1B_0$. If the instruction "sar al, 4" is executed, what is the result in AL? (use $B_7 \ldots B_0$, '1', or '0' to describe each bit of result).

Lab

A. Adding and Subtracting Numbers

Previous labs have already used the ADD instruction. The form of the ADD instruction is:

```
ADD destination, source          ;dest operand = dest operand + source operand
```

The destination operand can be a register or in memory. The source operand can be a register, in memory or immediate.

We also have ADC, *which means to add the two operands plus the carry*. ADC comes in handy when adding multiple words. The ADC instruction has the form:

```
ADC destination, source          ;dest = dest + source + CF (carry flag)
```

The form of the two equivalent subtraction operations (subtract and subtract with borrow) are:

```
SUB dest, souce                  ;dest = dest - source
SBB dest, source                 ;dest = dest - source - CF
```

The subtraction operation follows 3 steps:

1) takes the 2's complement of the source

2) add it to the destination

3) inverts carry.

If the Carry Flag is set after the operation, then a larger number was subtracted from a smaller number, and a 'borrow' occurred which sets the carry flag.

Add with Carry and Subtract with Borrow are useful for doing extended integer arithmetic -- i.e., doing a 64-bit addition or subtraction using only 8-bit , or 16-bit or 32-bit operations. The program below adds two 64 bit numbers (A+B) and stores the result in SUM using only 8 bit operations:

```
.model small
.586
.stack 100h
.data
;opA and opB are two 64 bit numbers in little endian order
;little endian means the least significant  byte is stored first
opA db 23h,0ABh,34h,80h,39h,01h,0F7h,10h      ;10F701398034AB23 h
opB db 7Ah,0FFH, 92h, 0D2h, 38h, 81h,0C2h,25h ;25C28138D292FF7A h
sum  db 8 dup (?)
.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   cx,8
      mov   bx,offset opA
      mov   si,offset sum
      clc
      mov   al,00h
LP1:  mov   al,[bx]
      adc   al, [bx+8]
      Mov   [si],al
      Inc   bx
      Inc   si
      Loop  lp1
      Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

Lab Question 1: Assemble this program (exam1.asm) and trace through its execution with Codeview.

A. What is the 64-bit value (in hex) that gets stored at SUM? Verify that this is the same value that you calculated for 'B' in the pre-lab.

B. What does the instruction 'CLC' accomplish and why is it needed?

C. Why is the 'mov al,00h' instruction needed?

D. Comment each line x86 line of the program and include the assembled listing in your report.

Lab Question 2: Modify the program above so that 16-bit additions are used (Hint: one of the needed changes would be "adc ax, [bx+8]". In the assembled listing in your lab report, and describe the differences between the two programs. Verify that your new program produces the same result as the old program. Each x86 instruction line must have comment explaining its function in the program.

B. Multiplying and Dividing Numbers

In multiplication and division operations, the x86 microprocessor use the registers AX, AL, AH, EAX, DX and EDX as used as shown in the tables.

Multiplication Summary

	Multiplicand	Multiplier	Result
8 bits x 8 bits	AL	register or memory	AX (16 bits)
16 bits x 16 bits	AX	register or memory	DX:AX (32 bits)
32 bits x 32 bits	EAX	register or memory	EDX: EAX (64 bits)

Division Summary

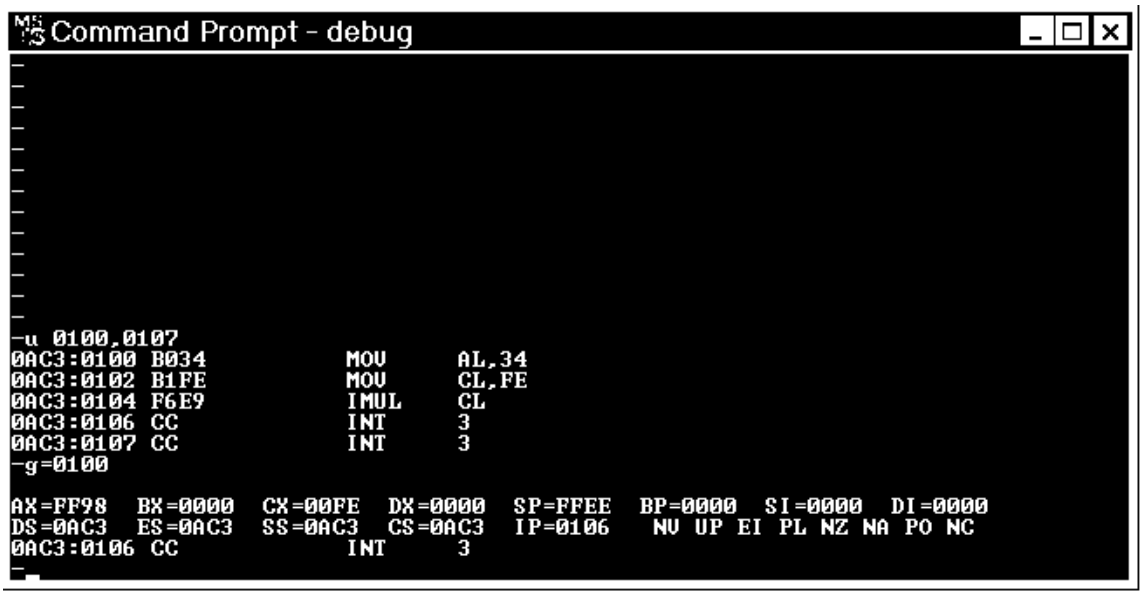
	Dividend (numerator)	Divisor (denominator)	Quotient	Remainder
16 bits / 8 bits	AX	register, memory (8-bit)	AL	AH
32 bits / 16 bits	DX:AX	register, memory (16-bit)	AX	DX
64 bits / 32 bits	EDX:EAX	register, memory (32 bits)	EAX	EDX

The operands can be considered as signed numbers or unsigned numbers. The unsigned multiplication and division operations are MUL, DIV. The signed multiplication/division operations are IMUL, IDIV.

For signed multiplication, if the two numbers have the same sign the result is always positive. If the operands are different signs then the result will be negative.

For signed division, if the signs of the dividend and divisor are the same, then the quotient sign is positive. If the signs of the dividend and divisor are different, then quotient sign is negative. The remainder sign is always the same sign as the dividend. You can always check your work via quotient*divisor + remainder = dividend)

Lab Question 3: Use either Codeview or Debug and verify your answers for Pre-lab questions C, D, E, F. For each one, capture a screen as shown below and include in your report. To capture a WINDOW, do ALT+PRINT_SCREEN while the window is selected. Then open the Paint program via Programs->Accessories->Paint and paste the image into the edit area. Save the image to disk as a monochrome BMP image -- use monochrome to reduce the file size.



The screen above shows a debug session that computes 34h (+52) x FEh (-2) = FF98 (-104) using signed multiplication

C. Logic Operations

The x86 instruction set includes the bit-wise logical operations of AND, OR, XOR.

The AND operation is useful for CLEARING particular bits in an operand ('0' AND anything = '0'). For example, the operation: AND AL, 0Fh will set bits B7-B4 to '0', and leave bits B3-B0 unaffected.

The OR operation is useful for SETTING particular bits in an operand ('1' OR anything = '1'). For example, the operation: OR AL, 0Fh will set bits B3-B0 to '1', and leave bits B7-B4 unaffected.

The XOR operation is useful complementing bits in an operand ('1' XOR anything = not(anything)). For example, the operand XOR AL, 0Fh will complement bits B3-B0 and leave bits B7-B4 unaffected. The XOR operation can also be used to clear a register to zero - the operation XOR AX, AX will set AX to zero (this requires less machine code than MOV AX,0000).

The program (exam2) below writes STRING1 to the console one character at time.

```
.model small
.586
.stack 100h
.data
string1 db "I aM A cOoL DUDz aND I roXXer","$"

.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   bx,offset string1
LP1:  mov   dl,[bx]
      Cmp   dl, '$'
      Je    exit
      Inc   bx
      ?????insert instruction here???
      mov   ah,02
      int   21h
      jmp   lp1
Exit: Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

Lab Question 4: Assemble and execute the above program. Note that string1 is printed to the screen.

A. Insert the instruction "and dl, 11011111B" where shown in the original progam (note that a binary value is used in order to make clearer what bit pattern is being applied. Assemble and re-execute the program. How is this different from before? Why? (you need to be looking at an ASCII table when you answer this question).

B. Insert the instruction "or dl, 00100000B" where shown in the original program (note that a binary value is used in order to make clearer what bit pattern is being applied. Assemble and re-execute the program. How is this different from before? Why?

C. Insert the instruction "xor dl, 00100000B" where shown in the original program (note that a binary value is used in order to make clearer what bit pattern is being applied. Assemble and re-execute the program. How is this different from before? Why?

You only need to include the assembled listing of one of A, B, or C programs in your lab report (choose one, each x86 instruction must have a comment explaining its function in the program). You must include the answers to A,B,C in your lab report.

D. Shift and Rotate Operations

Shift and Rotate instructions are used to move bits around in operands. Make sure you understand the basic shift/rotate instructions as explained in Section 7.1 of the Irvine text before you continue with this section.

The program below (exam3.asm) prints out the 8-bit value stored at NUM as a binary number to the screen.

```
.model small
.586
.stack 100h
.data
num  db  03Dh

.code
main  proc
      mov   ax,@data
      mov   ds,ax
      mov   bl,num
      mov   cx,8
LP1:  shr   bl,1
      Jc    is_one
      Mov   dl,30h
      Jmp   print
Is_one:
      Mov   dl,31h
Print:
      Mov   ah,2
      int   21h
      loop  lp1
Exit: Mov   ax, 4c00h
      Int   21h
Main  endp
End main
```

How does this program work?

1. The 'shr bl,1' shifts the BL register to the RIGHT by 1 position. The least significant bit is shifted into the Carry flag, and a '0' is shifted into the MSB position. If the LSB was a '1', then the carry flag will be set. If the LSB was a '0', then carry flag is cleared.
2. The "jc is_one' jumps to the instruction at label 'is_one' if the carry flag is set (JC means Jump on Carry). At label 'is_one', we load DL with the ASCII value for '1' and print this out to the screen. If the carry flag is not set, then the 'jc is_one' jump is not taken -- DL is loaded with the ASCII value for a '0' and then this is sent to the screen.

Lab Question 5: Assemble and execute the above program. You should notice a problem with the output - the 8-bit value is printed least significant bit to most significant bit. This is backwards from the way a binary number is normally printed which is most significant bit to least significant bit. Modify the program so that the number is printed most significant bit to least significant bit and verify its operation.

Lab Question 6: Modify the above program so that it uses a 16-bit value stored at NUM (ie. NUM DW 0A3FE h). Make sure the 16-bit number is printed most significant bit to least significant bit. Include the assembled listing of the program in your report and put a comment on every x86 instruction line.

Lab Report

Include the answers to all "**Lab Questions**" in your report.

Applying what you have learned

Demo to the TA the programs that you wrote for Lab Questions 2 (64-bit addition, 16-bits at a time) and 6 (printing a 16-bit binary number).