**Computer Organization & Assembly Language**

**Lab 8**

**Topics:**

1. Rotate and Shift keywords
2. iMul
3. iDiv
4. **Rotate and shift:**

|  |  |
| --- | --- |
| Mnemonic | Function |
| SHL, SHR | Logical shift left, right byte or word, by 1 or CL |
| SAL, SAR | Arithmetic shift left, right byte or word, by 1 or CL |
| ROL, ROR | Rotate left, right byte or word, by 1 or CL |
| RCL, RCR | Rotate left, right through carry byte or word, by 1 or CL |

* SHL operand1, operand2
  + Example
  + MOV AL, 11100000b
  + SHL AL, 1 ; AL = 11000000b
* SHR operand1, operand2
  + Example
  + MOV AL, 00000111b
  + SHR AL, 1 ; AL = 00000011b
* ROL operand1, operand2
  + shift all bits left, the bit that goes off is inserted to the right-most position.
  + Example
  + MOV AL, 1Ch ; AL = 10011100b
  + ROL AL, 1 ; AL = 00111001b
* ROR operand1, operand2
  + shift all bits right, the bit that goes off is inserted to the left-most position
  + Example
  + MOV AL, 1Ch ; AL = 00011100b
  + ROR AL, 1 ; AL = 00001110b

1. **Imul:**

The IMUL (signed multiply) instruction performs signed integer multiplication. Unlike the

MUL instruction, IMUL preserves the sign of the product. It does this by sign extending the highest bit of the lower half of the product into the upper bits of the product. The x86 instruction set supports three formats for the IMUL instruction: one operand, two operands, and three operands. In the one-operand format, the multiplier and multiplicand are the same size and the product is twice their size.

IMUL reg/mem8

IMUL reg/mem16

IMUL reg/mem32

.data

word1 SWORD 4

dword1 SDWORD 4

.code

imul bx,word1,-16

imul ebx,dword1,-16

imul ebx,dword1,-2000000000

1. **Idiv:**

Signed integer division is nearly identical to unsigned division, with one important difference:

The dividend must be sign-extended before the division takes place. Sign extension is the term

used for copying the highest bit of a number into all of the upper bits of its enclosing variable or

register. To show why this is necessary, let’s try leaving it out. The following code uses MOV to

assign 101 to AX, which is the lower half of EAX:

.data

wordVal SWORD -101 ; 009Bh

.code

mov eax,0 ; EAX = 00000000h

mov ax,wordVal ; EAX = 0000009Bh (+155)

mov bx,2 ; EBX is the divisor

idiv bx ; divide EAX by BX (signed operation)

**Tasks:**

1. Mov ax, 65535

Keep shifting right (dividing by 2) until ax reaches zero. Keep count of how many iterations it took for ax to reach 0. Store that in a variable.

1. Mov ax, 1

Keep shifting left (multiply by 2) until ax reaches its maximum limit (65536/0). Keep count of how many iterations it took for ax to reach 0. Store that in a variable.

1. Performed signed multiplication of the following:
   1. -22 \* 63
   2. -4 \* 56
   3. -7896 \* 2552
   4. -2554 \* -1221

Explain your result in comments in code. Explain if the final result is exceeding 16 bits. If yes, explain how the bits are stored in AX and DX.

1. Perform sign division for the following:
   1. -15 / 3
   2. -33 / 2
   3. 101 / -2
   4. 151 / -14

Explain your results in comments in code. The answers obtained, their negative/2’s complement form. Value of AL and AH of every part.

1. Perform SAR and SAL on the following, explain in comments how are their values getting set:
   1. 100010001b
   2. 00011000b
   3. 11111011b
   4. 00000010b
   5. 10101010b
2. Perform ROL and ROR on the following, explain in comments how are their values getting set:
   1. 10101011b
   2. 00110011b
   3. 10001000b
   4. 11110000b
   5. 00101101b
3. Perform RCL and RCR on the following, explain in comments how are their values getting set:
   1. 01011110b
   2. 11100011b
   3. 10101000b
   4. 00110011b
   5. 111001111b