Lecture Notes on Assembly Language - J. Vaughan

14. Multiplication

In multiplication and division, separate instructions exist for signed and unsigned arithmetic. In the case of multiplication, MUL deals with unsigned data and IMUL with signed data.

General format:

MUL/IMUL *register/memory*

The idea is that multiplicand is in some part of the eax register and (part of) the result is also placed there.

The operations available are byte*byte, word*word and dword*dword

Byte*Byte

Multiplicand is in AL and multiplier is a byte in memory or a register.

Product is placed in AX

Any data already in AH is ignored and overwritten by the product.

Word*Word

Multiplicand is in AX and multiplier is a word in memory or a register.

Product is placed in DX:AX

Any data already in DX is ignored and overwritten by the product.

Doubleword*Doubleword

Multiplicand is in EAX and multiplier is a doubleword in memory or a register.

Product is placed in EDX:EAX

Any data already in EDX is ignored and overwritten by the product.

The operation of MUL depends on the definition of the multiplier. Definitions using DB, DW and DD determine whether the operation is byte*byte, word*word or dword*dword. When the multiplier is in a register, the size of the register determines the operand sizes. Thus MUL BL, MUL DX and MUL ECX are byte*byte, word*word and dword*dword operations respectively.

For signed multiplication, IMUL is used. Remember the sizes of the operands, and that CBW or CWD will have to be used to sign-extend the multipliers if they are not of the length expected by IMUL.

Additional IMUL formats

16-bit operations

All operands are 16 bits long.

IMUL *register1, immediate*

IMUL *register1, register2/memory*

Register1 contains multiplicand. Second operand is the multiplier. Product goes in register1. A product that exceeds the register size causes the overflow and carry flags to be set.

32-bit operands

All operands are 32 bits long.

IMUL register, memory, immediate

Multiplicand is in memory. Multiplier is immediate data. Product goes in register.

IMUL register1, register2/memory

Register1 contains multiplicand. Second operand is the multiplier. Product goes in register1.

As for the 16-bit operations, a product that exceeds the register size causes the overflow and carry flags to be set.

Fast Multiplication

If the multiplier is a power of 2, it may be faster to shift the multiplicand left by the corresponding power.

Multiword Multiplication

Example

2435

* 12

29220

If you can only multiply 2-digit numbers, the multiplication has to be split up like this:

24	35
*12	*12
288	420

The separate products can then be added, remembering that the '288' is really in the '100s' place, so it has to have 2 zeros added in order to give its correct value. This is the same as shifting left by 2 decimal places.

28800

\+ 420

29220

The same logic can be applied to multiword binary multiplication, using binary shifts instead of adding decimal zeros.

Exercise

Consult the NASM manual to find out about shift operations. Write a routine to shift the DX:AX registers left by a number of bits specified in the CX register. Do the same for a right shift. Extend your routines to shift the EDX:EAX registers left or right by an amount specified in ECX. Note that shifts can affect the carry flag.

15. Division

In the case of division, DIV deals with unsigned data and IDIV with signed data.

General format:

DIV/IDIV *register/memory*

The idea is that (part of) the dividend is in some part of the eax register and the result is also placed there.

The operations available are byte into word, word into doubleword and doubleword into quadword.

Byte into Word

Dividend is in AX and divisor is a byte in memory or a register.

Quotient in AL, remainder in AH

Word into Doubleword

Dividend is in DX:AX and divisor is a word in memory or a register.

Quotient in AX, remainder in DX

Doubleword into Quadword

Dividend is in EDX:EAX and divisor is a doubleword in memory or a register.

Quotient in EAX, remainder in EDX

The operation of DIV depends on the definition of the divisor. Definitions using DB, DW and DD determine whether the operation is byte into word, word into doubleword or doubleword into quadword.

When the divisor is in a register, the size of the register determines the operand sizes. Thus DIV BL, DIV DX and DIV ECX are byte into word, word into doubleword and doubleword into quadword operations respectively.

For signed division, IDIV is used. Remember the sizes of the operands, and that CBW or CWD will have to be used to sign-extend the divisors if they are not of the length expected by IDIV.

Fast Division

If the multiplier is a power of 2, it may be faster to shift the multiplicand left by the corresponding power.

Division by Subtraction

Occasionally it may be faster to use the repeated-subtraction approach to division.

Overflows and Interrupts

DIV and IDIV operations can easily cause an overflow, with a resulting interrupt. The operations assume that the quotient is significantly smaller than the original dividend. Dividing by zero will always cause an interrupt. One rule of thumb is the following: If a divisor is a byte, it must be greater than the left byte of the dividend; if a divisor is a word, it must be greater than the left word of the dividend; if a divisor is a doubleword, it must be greater than the left doubleword of the dividend.