DIV Instruction

The following table shows the relationship between the dividend, divisor, quotient, and remainder for the DIV instruction:

Operand Size	Dividend	Divisor	Quotient	Remainder
8-bit	AX	reg/mem8	AL	AH
16-bit	DX:AX	reg/mem16	AX	DX
32-bit	EDX:EAX	reg/mem32	EAX	EDX

In 64-bit mode, the DIV instruction uses RDX:RAX as the dividend and permits the divisor to be a 64-bit register or memory operand. The quotient is stored in RAX, and the remainder is stored in RDX.

The table above shows the relationship between the dividend, divisor, quotient, and remainder for the DIV instruction.

- **Dividend** is the number being divided.
- **Divisor** is the number by which the dividend is being divided.
- **Quotient** is the result of dividing the dividend by the divisor.
- **Remainder** is the number that is left over after the dividend is divided by the divisor.

The table shows that the operand size of the dividend and divisor determines the operand size of the quotient and remainder.

For example, if the dividend and divisor are 8-bit integers, then the quotient and remainder will also be 8-bit integers.

The table also shows that the dividend and divisor can be stored in registers or memory.

For example, the dividend can be stored in the AX register, and the

divisor can be stored in the BL register.

Here is an example of how to use the DIV instruction to perform 8-bit unsigned division:

DIV Examples

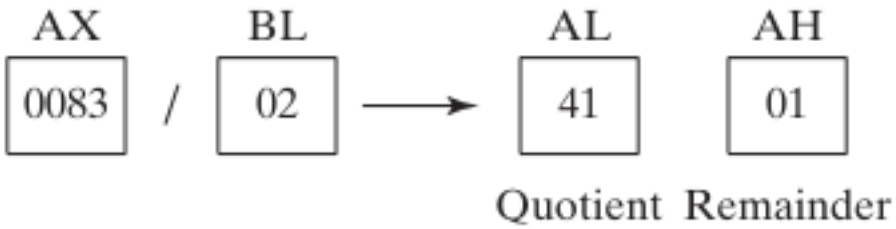
The following instructions perform 8-bit unsigned division (83h/2), producing a quotient of 41h and a remainder of 1:

```
746 ;dividend
747 mov ax, 0083h
748 ;divisor
749 mov bl, 2
750 ;divide
751 div bl
752 ;AL = 41h, AH = 01h
```

In this example, the dividend is stored in the AX register, and the divisor is stored in the BL register.

The DIV instruction divides the dividend by the divisor and stores the quotient in the AL register and the remainder in the AH register.

The following diagram illustrates the movement between registers:



DIV Example 2:

The following instructions perform 32-bit unsigned division using a memory operand as the divisor:

```
.data
    dividend QWORD 0000000800300020h
    divisor DWORD 00000100h
.code
    mov edx, DWORD PTR dividend + 4     ;high doubleword
    mov eax, DWORD PTR dividend         ;low doubleword
    div divisor
    ;EAX = 08003000h, EDX = 00000020h
```

Explanation:

The **.data directive** defines two variables: dividend and divisor. The dividend variable is a 64-bit integer (QWORD) that contains the dividend.

The **divisor variable is a 32-bit integer (DWORD)** that contains the divisor. The .code directive marks the beginning of the code section.

The **mov edx, DWORD PTR dividend + 4** instruction loads the high doubleword of the dividend into the EDX register.

The **mov eax, DWORD PTR** dividend instruction loads the low doubleword of the dividend into the EAX register.

The **div divisor instruction** divides the dividend in the EAX:EDX registers by the divisor in the divisor variable and stores the quotient in the EAX register and the remainder in the EDX register.

After the DIV instruction executes, the EAX register will contain the quotient (08003000h) and the EDX register will contain the remainder (00000020h).

In other words, the above code performs the following operation:

```
EAX:EDX = 0000000800300020h / 00000100h
```

The result is stored in the EAX:EDX registers, with the quotient in EAX and the remainder in EDX.

The **EAX:EDX registers** are a pair of 32-bit registers that can be used

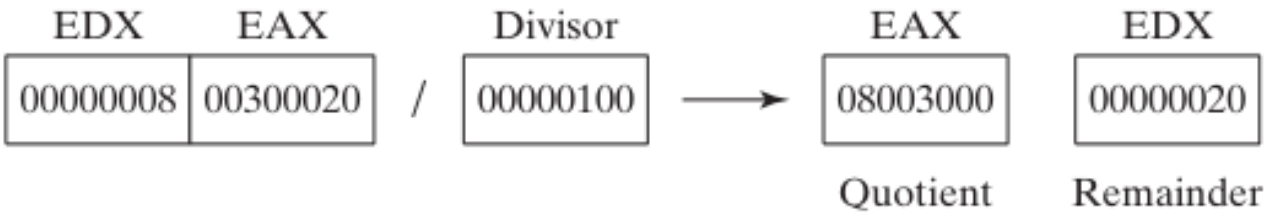
to store a 64-bit value. The EAX register stores the lower 32 bits of the value, and the EDX register stores the higher 32 bits of the value.

When you say that the result of a division operation is stored in the EAX:EDX registers, it means that the quotient of the division is stored in the EAX register and the remainder of the division is stored in the EDX register.

For example, if you divide the number 100 by the number 10, the quotient is 10 and the remainder is 0. The EAX register would contain the value 10 and the EDX register would contain the value 0.

Another way to think about it is that the EAX:EDX registers can be used to store a 64-bit integer. When you perform a division operation, the result is a 64-bit integer, which is then stored in the EAX:EDX registers.

The following diagram illustrates the movement between registers:



This image is related to the text you provided. The image shows a diagram of a sequence of numbers, with the following arrows and labels:

This diagram illustrates the **32-bit unsigned division operation** that is described in the text.

The dividend is 00300020h, the divisor is 00000100h, the quotient is 08003000h, and the remainder is 00000020h.

The EAX register contains the low doubleword of the dividend and the EDX register contains the high doubleword of the dividend.

The DIV instruction divides the dividend in the **EAX:EDX registers** by the divisor in the divisor variable. The quotient is stored in the EAX register and the remainder is stored in the EDX register.

EAX:EDX = 0000000800300020h / 00000100h

This equation represents the division operation that is being performed. The dividend is 0000000800300020h and the divisor is 00000100h. The result of the division is stored in the EAX:EDX registers.

DIV Example 3:

The following 64-bit division produces the quotient (0108000000003330h) in RAX and the remainder (0000000000000020h) in RDX:

```
787 .data
788     dividend_hi QWORD 0000000000000108h
789     dividend_lo QWORD 0000000033300020h
790     divisor QWORD 0000000000010000h
791 .code
792     mov rdx, dividend_hi
793     mov rax, dividend_lo
794     div divisor ;RAX = 0108000000003330
795     ;RDX = 0000000000000020
```

Explanation:

The .data directive defines three variables: dividend_hi, dividend_lo, and divisor.

The dividend_hi and dividend_lo variables contain the high and low doublewords of the dividend, respectively.

The divisor variable contains the divisor. The .code directive marks the beginning of the code section.

The mov rdx, dividend_hi instruction loads the high doubleword of the dividend into the RDX register.

The mov rax, dividend_lo instruction loads the low doubleword of the dividend into the RAX register.

dividend into the RAX register.

The div divisor instruction divides the dividend in the RAX:RDX registers by the divisor in the divisor variable and stores the quotient in the RAX register and the remainder in the RDX register.

After the DIV instruction executes, the RAX register will contain the quotient (0108000000003330h) and the RDX register will contain the remainder (0000000000000020h).

Why is each hexadecimal digit in the dividend shifted 4 positions to the right?

This is because the dividend is being divided by 64. In other words, the dividend is being shifted 6 bits to the right.

Each hexadecimal digit represents 4 bits, so each hexadecimal digit in the dividend will be shifted 4 positions to the right.

For example, the high doubleword of the dividend (0000000000000108h) is shifted 4 positions to the right to produce the following result:

0000000000000108h >> 4 = 0000000000000010h

The low doubleword of the dividend (0000000033300020h) is also shifted 4 positions to the right to produce the following result:

0000000033300020h >> 4 = 0000000000003330h

The quotient of the division operation is then stored in the RAX register and the remainder is stored in the RDX register.