Visão geral das instruções

Falaremos apenas do subconjunto de instruções necessários para os tópicos e programas discutidos no âmbito deste texto. Isso vai excluir algumas das instruções mais avançadas e instruções de modo restrito.

As instruções são apresentadas na seguinte ordem:

• Movimento de dados

• Instruções de conversão

• Instruções aritméticas

• Instruções lógicas

• Instruções de controle

1 Convenções de notação

Esta seção resume a notação usada neste texto, que é bastante comum na literatura técnica. Em geral, uma instrução consistirá na instrução ou operação própria (ou seja, add, sub, mul, etc.) e os operandos. Os operandos referem-se a onde os dados (a ser operado) vem de e / ou de onde o resultado será colocado.

1.1 Notação de operando

A tabela a seguir resume as convenções de notação usadas.

Operando	**Descrição**
<reg>	Operando registrador. O operando deve ser um registrador
<reg8>, <reg16>, <reg32>, <reg64>	Operando registrador com requisito de tamanho específico. Por exemplo, **reg8** significa um registro de tamanho de byte (por exemplo, al, bl, etc.) apenas e **reg32** significa um registrador de tamanho de palavra dupla (por exemplo, eax, ebx, etc.) apenas.
<dest>	Operando de destino. O operando pode ser um registrador ou memória. Por ser um operando de destino, o conteúdo será substituído com o novo resultado (com base em instrução específica).
<Rxdest>	Operando de registro de destino de ponto flutuante. O operando deve ser um registro de ponto flutuante. Por ser um destino operando, o conteúdo será substituído pelo novo resultado (com base em instrução específica).
<src>	Operando fonte (origem). O valor do operando permanece inalterado após a instrução.
<imm>	Valor imediato. Pode ser especificado em decimal, hex, octal, ou binário.
<mem>	Localização da memória. Pode ser um nome de variável ou referência

	indireta (ou seja, um endereço de memória).
<op> ou <operand>	Operando, registro ou memória.
<op8>, <op16>, <op32>, <op64>	Operando, registro ou memória, com tamanho específico. Por exemplo, **op8** significa um operando de tamanho byte apenas e **op32** significa um tamanho de um operando de double word apenas.
<label>	Um rótulo de programa

Por padrão, os valores imediatos são decimais ou base *10*. Podem ser usados valores imediatos hexadecimais ou de base *16*, mas devem ser precedidos por **0x** para indicar que o valor é hexadecimal. Por exemplo, 15_{10} pode ser inserido em hexadecimal como *0x0F (ou OFh)*.

2. Movimento de Dados

Normalmente, os dados devem ser movidos para um registro de CPU da RAM para que possam ser tratados. Assim que os cálculos forem concluídos, o resultado pode ser copiados colocados em uma variável. Existem várias fórmulas simples no programa exemplo que executa essas etapas. Esta operação básica de movimentação de dados é realizada com a instrução de movimento.
A forma geral da instrução de movimento é:

mov <dest>, <src> ; dest = src

~~mov [vetaux],[vet1] ;~~ não pode

```
mov bx, [vet1]      ;pode
mov [vetaux],bx     ;pode
```

O operando de origem é copiado para o operando de destino. O valor do operando de origem permanece inalterado. O operando de destino e origem deve ser do mesmo tamanho (ambos os bytes, ambas as palavras, etc.). O operando de destino não pode ser um imediato. Ambos os operandos não podem ser memória. Se uma operação de memória para memória for necessário, duas instruções devem ser usadas.

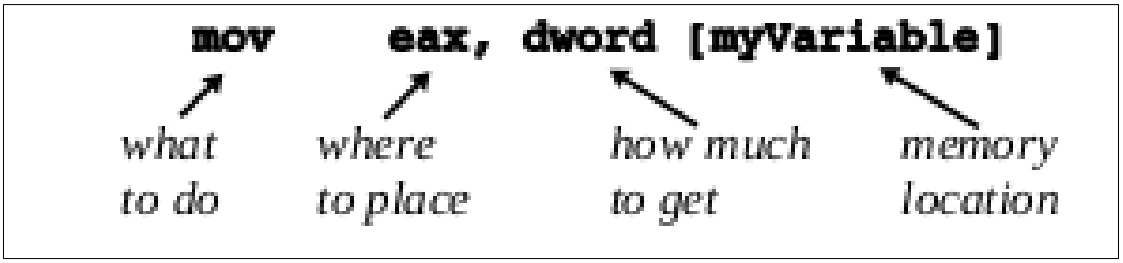
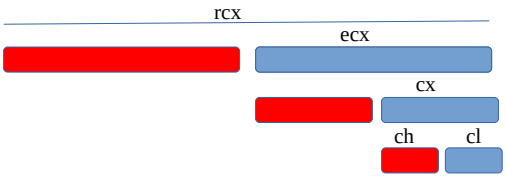


Figura 1. Instrução MOV

Quando o operando do registrador de destino tem tamanho double-word e o operando de origem tem tamanho quadword, a double-word de ordem superior do registrador quad-word é definida como zero. Isso se aplica apenas quando o operando de destino é um registrador inteiro double-word.
Especificamente, se as seguintes operações forem realizadas,

```
mov eax, 100 ; eax = 0x00000064
mov rcx, -1    ; rcx = 0xffffffffffffffff
mov ecx, eax ; ecx = 0x00000064
```



Inicialmente, o registro **rcx** deve ser definido como -1 (que são todos 0xF's). Quando o número positivo do registrador **eax** (100_{10}) é movido para o registrador **rcx**, a parte de ordem superior do registrador quadword **rcx** é definida como 0 sobrescrevendo os 1s da instrução anterior.

A instrução mov é resumida como segue:

Instrução	**Explicação**
mov <dest>, <src>	Copia o operando de origem para o operando de destino. Nota 1, ambos os operandos não podem ser memória. Nota 2, operandos de destino não podem ser imediatos. Nota 3, para um destino double-word recebe um operando origem, a parte de ordem superior do registrador quad word é definida como 0.
Exemplos:	**mov ax, 42** **mov cl, byte [bvar]** **mov dword [dVar], eax** **mov qword [qVar], rdx**

Por exemplo, supondo as seguintes declarações de dados:

```
dValue      dd      0
bNum        db      42
wNum        dd      5000
dNum        dw      73000
qNum        dw      73000000
bAns        db      0
wAns        dw      0
dAns        dd      0
qAns        dd      0
```

Vamos realizar as seguintes operações:

```
dValue = 27
bAns = bNum
wAns = wNum
dAns = dNum
qAns = qNum
```

Usamos as seguintes operações:

```
mov     dword [dValue], 27          ; dValue = 27

mov     al, byte [bNum]
mov     byte [bAns], al             ; bAns = bNum

mov     ax, word [wNum]
mov     word [wAns], ax             ; wAns = wNum

mov     eax, dword [dNum]
mov     dword [dAns], eax           ; dAns = dNum

mov     rax, qword [qNum]
mov     qword [qAns], rax           ; qAns = qNum
```

Para algumas instruções acima, a especificação de tipo explícita (por exemplo, byte, word, double-word, qword) pode ser omitida, pois o outro operando definirá claramente o tamanho. No texto, ele será incluído por questões de consistência e boas práticas de programação.

3. Endereços e valores

A única maneira de acessar a memória é com os colchetes ([]). A omissão dos colchetes não acessará a memória e, em vez disso, obterá o endereço do item. Por exemplo:

mov rax, qword [var1] ; valor de var1 em rax (ou mov rax, [var1])
mov rax, var1 ; endereço de var1 em rax

3.1 Instrução LEA

Visto que omitir os colchetes não é um erro, o montador não gerará mensagens de erro ou avisos. Isso pode causar confusão.

Além disso, o endereço de uma variável pode ser obtido com o endereço efetivo de carga, ou instrução **lea**. A instrução de endereço efetivo de carga é resumida da seguinte forma:

Instrução	**Explicação**
lea <reg64>, <mem>	Carrega o endereço de <mem> em reg64.
Exemplos:	**lea rcx, byte [bvar] ; equivalente a mov rcx, bvar** **lea rsi, dword [dVar]**

4. Instruções de conversão

Às vezes, é necessário converter de um tamanho para outro. Por exemplo, um **byte** pode precisar ser convertido em uma **dword** para alguns cálculos em uma fórmula. O processo usado para conversões depende do tamanho e tipo do operando. As seções a seguir resumem como as conversões são realizadas.

4.1 Conversão de contração

As conversões de contração convertem um tipo maior para um tipo menor (ou seja, word para byte ou dword para word).
Nenhuma instrução especial é necessária para restringir as conversões. A parte inferior da localização da memória ou registro pode ser acessada diretamente. Por exemplo, se o valor de 50 (0x32) for colocado no registro **rax**, o registro **al** pode ser acessado diretamente para obter o valor da seguinte forma:

```
mov rax, 50
mov byte [bVal], al
```

Este exemplo é razoável, pois o valor de 50 caberá em um valor de byte. No entanto, se o valor de 500 (0x1f4) for colocado no registro rax, o registro al ainda pode ser acessado.

```
mov rax, 500 ; não cabe em um byte, precisa de dois bytes
mov byte [bVal], al ; bval = 0xf4 = 244
```

Neste exemplo, a variável bVal conterá 0xf4 que pode levar a resultados incorretos. O programador é responsável por garantir que as conversões de redução sejam realizadas de forma adequada. Ao contrário de um compilador, nenhum aviso ou mensagem de erro será gerado.

4.2 Conversão de expansão

As conversões de expansão são de um tipo menor para um tipo maior (por exemplo, byte para word ou word para double-word). Como o tamanho está sendo expandido, os bits de ordem superior devem ser definidos com base no sinal do valor original. Como tal, o tipo de dados, com ou sem sinal, deve ser conhecido e o processo ou instruções apropriados devem ser usados.

4.2.1 Conversão de expansão sem sinal

Para conversões de expansão sem sinal, a parte superior da localização da memória ou registro deve ser definida como zero. Como um valor sem sinal só pode ser positivo, os bits de ordem superior só podem ser zero. Por exemplo, para converter o valor de byte de 50 no registrador al, para um valor de quadword em rbx, as seguintes operações podem ser realizadas.

```
mov al, 50
mov rbx, 0
mov bl, al
```

Como o registro rbx foi definido como 0 e os 8 bits mais baixos foram para o valor de al (50 neste exemplo), todo o registro rbx de 64 bits agora é 50.

Este processo geral pode ser executado na memória ou em outros registros. É responsabilidade do programador garantir que os valores sejam apropriados para os tamanhos de dados que estão sendo usados.
Uma conversão sem sinal de um tamanho menor para um tamanho maior também pode ser realizada com uma instrução de movimento especial, da seguinte maneira:

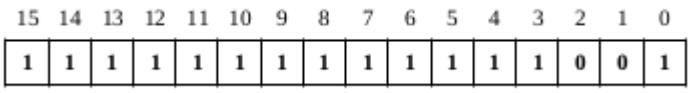
movzx <dest>, <orig>

Isto preencherá os bits de ordem superior do registrador destino com zeros. Um resumo das instruções que executam a conversão de alargamento sem sinal são as seguintes:

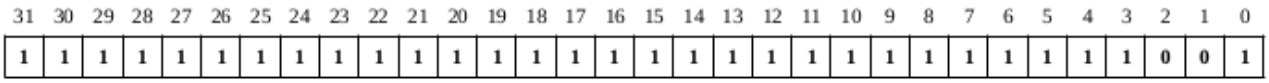
Instrução	**Explicação**
movzx <dest>, <src> **movzx <reg16>,<op8>** **movzx <reg32>,<op8>** **movzx <reg32>,<op16>** **movzx <reg64>,<op8>** **movzx <reg64>,<op16>** **movzx <reg64>,<op32>**	Conversão de largura sem sinal. Nota 1, ambos os operandos não podem ser memória. Nota 2, operandos de destino não podem ser um imediato. Nota 3, valores imediatos não permitidos.
Exemplos:	**movzx cx, byte [bVar]** **movzx dx,al** **movzx ebx, word [wVar]** **movzx ebx, cx** **movzx rbx, cl** **movzx rbx, cx**

4.2.2 Conversão com sinal

Para conversões de expansão com sinal, os bits de ordem superior devem ser definidos como 0 ou 1, dependendo se o valor original era positivo ou negativo.
Isso é executado por uma operação de extensão de sinal. Especificamente, o bit de ordem superior do valor original indica se o valor é positivo (com 0) ou negativo (com 1). O bit de ordem superior do valor original é estendido para os bits superiores do novo valor ampliado.
Por exemplo, dado que o registro **ax** está definido como -7 (0xfff9), os bits seriam definidos da seguinte forma:



Como o valor é negativo, o bit de ordem superior (bit 15) é 1. Para converter o valor da palavra no registrador **ax** em um valor de palavra dupla no registrador **eax**, o bit de ordem superior (1 neste exemplo) é estendido ou copiado em toda a palavra de ordem superior (bits 31-16), resultando no seguinte:



Há uma série de instruções dedicadas usadas para converter valores com sinal em um registrador **A** de um tamanho menor para um de tamanho maior. Essas instruções funcionam apenas no registrador A, às vezes usando o registrador **D** para o resultado. Por exemplo, a instrução **cwd** converterá um valor com sinal no registrador **ax** em um valor de double-word nos registradores **dx** (parte de ordem superior) e **ax** (parte de ordem inferior). Normalmente, é por convenção escrito como **dx: ax**. A instrução **cwde** converterá um valor com sinal no registrador **ax** em um valor dword no registrador **eax**.
Uma conversão com sinal mais generalizada de um tamanho menor para um tamanho maior também pode ser realizada com algumas instruções de movimento especiais, como segue:

Instrução	**Explicação**
cbw	Converte byte em **al** para word em **ax**. Note que só funciona do registrador **al** para **ax**
Exemplos:	**cbw**
cwd	Converte word em **ax** para double-word em **dx:ax**. Note que só funciona do registrador **ax** para **dx:ax**
Exemplos:	**cwd**
cwde	Converte word em **ax** para double-word em **eax**. Note que só funciona do registrador **ax** para **eax**
Exemplos:	**cwde**
cdq	Converte word em **eax** em quadword em **edx:eax**. Note que só funciona do registrador **eax** para **edx:ax**
Exemplos:	**cdq**
cdqe	Converte double-word em **eax** para quadword em **rax**. Note que só funciona do registrador **eax** para **rax**
Exemplos:	**cdqe**
cqo	Converte quadword em r**ax** para quadword em **rdx:rax**. Note que só funciona do registrador **rax** para **rdx:rax**
Exemplos:	**cqo**
movsx <dest>, <src> **movsx <reg16>,<op8>** **movsx <reg32>,<op8>** **movsx <reg32>,<op16>** **movsx <reg64>,<op8>** **movsx <reg64>,<op16>** **movsxd <reg64>,<op32>**	Conversão de expansão com sinal (via expansão de sinal). Nota 1, ambos os operandos não podem ser memória. Nota 2, operandos de destino não podem ser um imediato. Nota 3, valores imediatos não permitidos. Nota 4, instrução especial (movsxd) necessária para expansão com sinal de 32 a 64 bits.
Exemplos:	**movsx cx, byte [bVar]** **movsx dx, al** **movsx ebx, word [wVar]** **movsx ebx, cx** **movsxd rbx, dword [dVar]**

5. Instruções de operação aritmética

5.1 Adição

A forma geral de adição é

add <dest>, <src>; <dest> = <dest>+<src>

Especificamente, os operandos de origem e destino são adicionados e o resultado é colocado no operando de destino (sobrescrevendo o conteúdo anterior). O valor do operando de origem permanece inalterado. O operando de destino e de origem devem ter o mesmo tamanho (ambos os bytes, ambas as palavras, etc.). O operando de destino não pode ser imediato. Ambos os operandos não podem ser memória. Se uma operação de adição de memória para memória for necessária, duas instruções devem ser usadas.
Para exemplo,sejam as declarações abaixo:

```
bNum1       db      42
bNum2       db      73
bAns        db      0

wNum1       dw      4321
wNum2       dw      1234
wAns        dw      0

dNum1       dd      42000
dNum2       dd      73000
dAns        dd      0

qNum1       dq      42000000
qNum2       dq      73000000
qAns        dq      0
```

As seguintes instruções podem ser usadas:

```
; bAns = bNum1 + bNum2
mov     al, byte [bNum1]
add     al, byte [bNum2]
mov     byte [bAns], al

; wAns = wNum1 + wNum2
mov     ax, word [wNum1]
add     ax, word [wNum2]
mov     word [wAns], ax
; dAns = dNum1 + dNum2
mov     eax, dword [dNum1]
add     eax, dword [dNum2]
mov     dword [dAns], eax

; qAns = qNum1 + qNum2
mov     rax, qword [qNum1]
add     rax, qword [qNum2]
mov     qword [qAns], rax
```

Para algumas instruções, incluindo estas acima, a especificação de tipo explícita (por exemplo, byte, word, dword, qword) pode ser omitida, pois o outro operando definirá claramente o tamanho. Ele está incluído para consistência e boas práticas de programação.
Além da instrução **add** básica temos a instrução que adiciona um ao operando especificado.

inc <operand> ; <operand> = <operand>+1

O resultado é exatamente o mesmo que usar a instrução **add** (e adicionar um). Ao usar um operando de memória, a especificação de tipo explícita (por exemplo, byte, palavra, dword, qword) é necessária para definir claramente o tamanho.
Exemplo:

```
inc     rax                     ; rax = rax + 1
inc     byte [bNum]             ; bNum = bNum + 1
inc     word [wNum]             ; wNum = wNum + 1
inc     dword [dNum]            ; dNum = dNum + 1
inc     qword [qNum]            ; qNum = qNum + 1
```

A instrução de adição opera da mesma forma em dados com e sem sinal. É responsabilidade do programador garantir que os tipos e tamanhos de dados sejam apropriados para as operações realizadas.

5.1.1 Adição com transporte

A adição com transporte (***"vai 1"***) é uma instrução especial de adição que incluirá um transporte de uma operação de adição anterior. Isso é útil ao adicionar números muito grandes, especificamente números maiores que o tamanho do registro da máquina.

Usar um transporte adicional é uma operação padrão. Por exemplo, considere o seguinte:

```
  17
+ 25
----
  42
```

Como você deve se lembrar, os dígitos menos significativos (7 e 5) são adicionados primeiro. O resultado de 12 é anotado como 2 com 1 transporte. Os dígitos mais significativos (1 e 2) são adicionados junto com o transporte anterior (1 neste exemplo) resultando em 4.
Como tal, são necessárias duas operações de adição. Como não há transporte possível com a parte menos significativa, uma instrução de adição regular é usada. A segunda operação de adição precisaria incluir um possível transporte da operação anterior e deve ser feita com uma adição com instrução de transporte. Além disso, a adição com transporte deve seguir imediatamente a operação de adição inicial para garantir que o registrador rFlag não seja alterado por uma instrução não relacionada (portanto, possivelmente alterando o bit de transporte).
Para programas em assembly, a quadword menos significativa (LSQ) é adicionada com a instrução add e, em seguida, imediatamente a quadword mais significativa (MSQ) é adicionada com o adc, que adicionará as palavras quádruplas e incluirá um transporte da operação de adição anterior.
A forma geral é:

adc <dest>,<src>; <dest> = <dest> + <src> + <carryBit>

Especificamente, os operandos de origem e destino junto com o bit de transporte são adicionados e o resultado é colocado no operando de destino (sobrescrevendo o valor anterior). O carry é parte do registrador rFlag. O valor do operando de origem permanece inalterado.
O operando de destino e de origem devem ter o mesmo tamanho (ambos os bytes, ambas as palavras, etc.). O operando de destino não pode ser imediato. Ambos os operandos não podem ser memória. Se uma operação de adição de memória para memória for necessária, duas instruções devem ser usadas.
Por exemplo considere a declaração abaixo:

dquad1 ddq 0x1A000000000000000
dquad2 ddq 0x2C000000000000000
dqSum ddq 0

Cada uma das variáveis **dquad1**, **dquad2** e **dqSum** tem 128 bits e, portanto, excederá o tamanho do registro de 64 bits da máquina. No entanto, dois registros de 64 bits podem ser usados para cada um dos valores de 128 bits. Isso requer duas instruções de movimentação, uma para cada registro de 64 bits. Por exemplo,

mov rax, qword [dquad1]
mov rdx, qword [dquad1+8]

O primeiro movimento para o registrador **rax** acessa os primeiros 64 bits da variável de 128 bits. O segundo movimento para o registro **rdx** acessa os próximos 64 bits da variável de 128 bits. Isso é feito usando o endereço inicial da variável, **dquad1,** e adicionando 8 bytes, pulando assim os primeiros 64 bits (ou 8 bytes) e acessando os próximos 64 bits.
Se os LSQs forem adicionados e, em seguida, os MSQs forem adicionados incluindo qualquer transporte, o resultado de 128 bits pode ser obtido corretamente. Por exemplo,

```
mov     rax, qword [dquad1]
mov     rdx, qword [dquad1+8]

add     rax, qword [dquad2]
adc     rdx, qword [dquad2+8]

mov     qword [dqsum], rax
mov     qword [dqsum+8], rdx
```

Inicialmente, o LSQ de **dquad1** é colocado em **rax** e o MSQ é colocado em rdx. Em seguida, a instrução **add** adicionará o rax de 64 bits com o LSQ de dquad2 e, neste exemplo, fornecerá um transporte de 1 com o resultado em rax. Em seguida, o **rdx** é adicionado com o MSQ de **dquad2** junto com o transporte por meio da instrução **adc** e o resultado colocado em **rdx**.

5.2 Subtração

A forma geral de subtração de inteiros é como segue:
sub <dest>, <src>; <dest> = <dest> - <src>

Especificamente, o operando de origem é subtraído do operando de destino e o resultado é colocado no operando de destino (sobrescrevendo o valor anterior). O valor do operando de origem permanece inalterado. O operando de destino e de origem devem ter o mesmo tamanho (ambos os bytes, ambos words, etc.). O operando de destino não pode ser imediato.

Ambos os operandos não podem ser memória. Se uma operação de subtração de memória para memória for necessária, duas instruções devem ser usadas. Em termos de operação é semelhante à operação de adição.
Além da instrução de subtração básica, há uma instrução de decremento que subtrairá um do operando especificado. A forma geral da instrução de decremento é a seguinte:

dec <operand> ; <operand> = <operand> -1

As demais regras são semelhantes à operação **inc.**

5.3 Multiplicação

A instrução **mult** multiplica dois operandos inteiros. Matematicamente, existem regras especiais para lidar com a multiplicação de valores com sinais. Como tal, instruções diferentes são usadas para multiplicação sem sinal (**mul**) e multiplicação com sinal (**imul**).
A multiplicação normalmente produz resultados de tamanho duplo. Ou seja, a multiplicação de dois valores de *n* bits produz um resultado de *2 n* bits. Multiplicar dois números de 8 bits produzirá um resultado de 16 bits. Da mesma forma, a multiplicação de dois números de 16 bits produzirá um resultado de 32 bits, a multiplicação de dois números de 32 bits produzirá um resultado de 64 bits e a multiplicação de dois números de 64 bits produzirá um resultado de 128 bits.
Existem muitas variantes para a instrução multiplicação. Para a multiplicação com sinal, alguns formulários truncarão o resultado para o tamanho dos operandos originais. É responsabilidade do programador garantir que os valores usados funcionarão para as instruções específicas selecionadas.

5.3.1 Multiplicação sem sinal

A forma geral de multiplicação é:

mul <src>; Acumulador*<src>

Onde o operando de origem deve ser um registrador ou local de memória. Um operando imediato não é permitido.
Para a instrução de multiplicação de operando único, o registro A (al / ax / eax / rax) deve ser usado para um dos operandos (**al** para 8 bits, **ax** para 16 bits, **eax** para 32 bits e **rax** para 64 -bits).
O outro operando pode ser um registrador ou variável, mas não imediato.
Além disso, o resultado será colocado nos registradores **A** e possivelmente **D**, com base nos tamanhos que estão sendo multiplicados. A tabela a seguir mostra as várias opções para multiplicações não assinadas de byte, word, double-word e quadword.

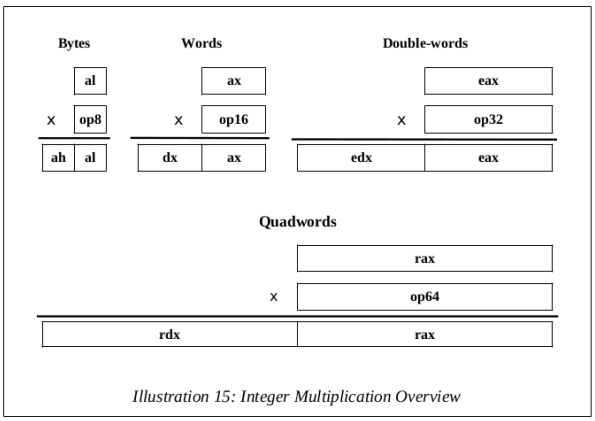


Illustration 15: Integer Multiplication Overview