CMSC 131 Introduction to Computer Organization and Machine-Level Programming
Handout 3: Basic Assembly Arithmetic Instructions

Learning Outcomes

At the end of this session, the student should be able to:

1. understand basic integer arithmetic operations in Assembly; and
2. develop Assembly programs implementing basic arithmetic operations.

Content

Assembly Operands

In Assembly, arithmetic instructions require one or two operands.
An operand can be:

- Register (`rax, ebx, cx, dh`)
- Immediate value (`2, 10, 0x1A`)
    - These operands cannot be used as destination operands.
- Memory - square brackets denotes the value of the variable
    - `[myVar]` - value of `myVar`
    - `myVar` - address of `myVar`

Integer Arithmetic Instructions

The integer arithmetic instructions perform arithmetic operations such as addition, subtraction, multiplication, and division on integer values. The following sections present the basic integer arithmetic operations.

Example 1: Prepare some memory operands in our sample code.

```
global _start

section .data
    SYS_EXIT equ 60
    bNum1 db 42
    bNum2 db 73
    bAns db 0

    wNum1 dw 4321
    wNum2 dw 1234
    wAns dw 0

    qNum1 dq 42000000
```

```
    qNum2 dq 73000000
    qAns dq 0

section .text
_start:

    ;INSERT CODE SNIPPET HERE

    ;exit code
    mov rax, SYS_EXIT
    xor rdi, rdi
    syscall
```

Addition

The general form of the integer addition instruction is:

```
add <dest>, <src>
```

where operation performs *<dest> = <dest> + <src>*

Notes:

- Only the value of the destination operand, not the source operand, is changed.
- The destination and source operand must be of the same size.
- Operands cannot be both memory. If a memory to memory addition operation is required, two instructions must be used.

Example 1A: Modify text section in example 1

```
    mov rax, 4
    add rax, 42
    add rax, qword[qNum2]

    ;memory to memory addition
    ;bAns = bNum1 + bNum2
    mov rax, 0
    mov al, byte[bNum2]
    add al, byte[bNum1]
    mov byte[bAns], al

    ;exit code
```

Increment

The general form of the integer increment instruction is:

```
inc <dest>
```

where operation performs *<dest> = <dest> + 1*

Example 1B: Modify text section in example 1

```
inc rax             ; rax = rax + 1
inc byte [bNum1]     ; bNum1 = bNum1 + 1

;exit code
```

Subtraction

The general form of the integer subtraction instruction is:

```
sub <dest>, <src>
```

where operation performs *<dest> = <dest> – <src>*

Notes:

- Only the value of the destination operand, not the source operand, is changed.
- The destination and source operand must be of the same size.
- Operands cannot be both memory. If a memory to memory addition operation is required, two instructions must be used.

Example 1C: Modify text section in example 1

```
sub qword [qNum1], 300

;wAns = wNum1 - wNum2
mov bx, word [wNum1]
sub bx, word [wNum2]
mov word [wAns], bx

;exit code
```

Decrement

The general form of the integer decrement instruction is:

```
dec <dest>
```

where operation performs *<dest> = <dest> – 1*

Example 1D: Modify text section in example 1

```
dec rbx             ; rbx = rbx - 1
dec word [wNum1]    ; wNum = wNum - 1

;exit code
```

Multiplication

The general form of the integer multiplication instruction is:

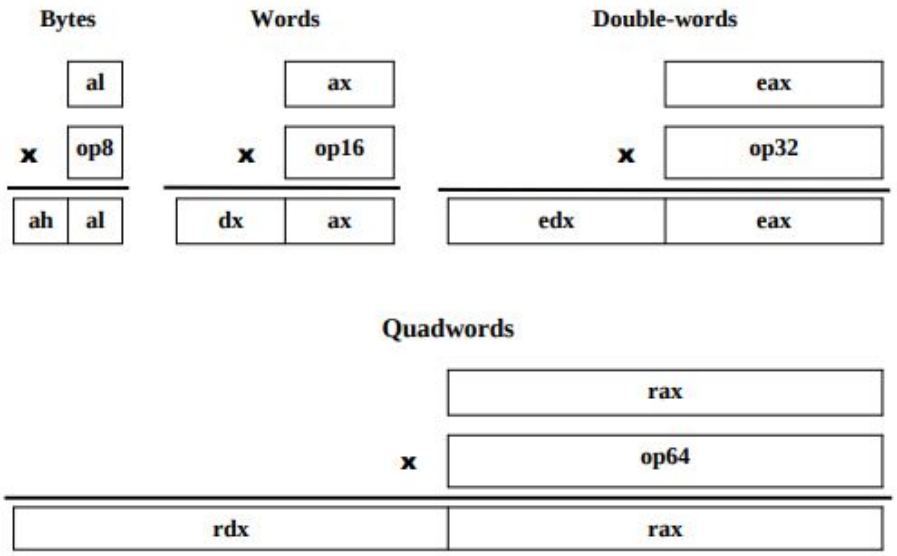
```
mul <src>
```

where operation multiplies the A register (`al, ax, eax, or rax`) and the `<src>`. The result will be placed in the A and possibly D (`dx, edx, or rdx`) registers, based on the sizes being multiplied.

Notes:

- If <src> is byte-size, then **`ax = al * <bsrc>`**
- If <src> is word-size, then **`dx:ax = ax * <wsrc>`**
- If <src> is doubleword-size, then **`edx:eax = eax * <dsrc>`**
- If <src> is quadword-size, then **`rdx:rax = rax * <qsrc>`**
- The <src> can be a memory location or a register, but not an immediate.

A simple illustration on how multiplication in Assembly works is shown below.



Example 2: Create a new file mul.asm

```
global _start

section .data
    SYS_EXIT equ 60
    bNumA db 42
    bNumB db 73
    wAns dw 0

    wNumA dw 4321
    wNumB dw 1234
    dAns dd 0

    dNumA dd 42000
```

```
    dNumB dd 73000
    qAns dq 0

section .text
_start:

    ;wAns = bNumA * bNumB
    mov al, byte [bNumA]
    mul byte [bNumB]            ; result in ax
    mov word [wAns], ax

    ;dAns = wNumA * wNumB
    mov ax, word [wNumA]
    mul word [wNumB]            ; result in dx:ax
    mov word [dAns], ax
    mov word [dAns+2], dx

    ;qAns = dNumA * dNumB
    mov eax, dword [dNumA]
    mul dword [dNumB]           ; result in edx:eax
    mov dword [qAns], eax
    mov dword [qAns+4], edx

    mov rax, SYS_EXIT
    xor rdi, rdi
    syscall
```

Division

$$\text{Recall that } \frac{\textit{dividend}}{\textit{divisor}} = \textit{quotient}$$

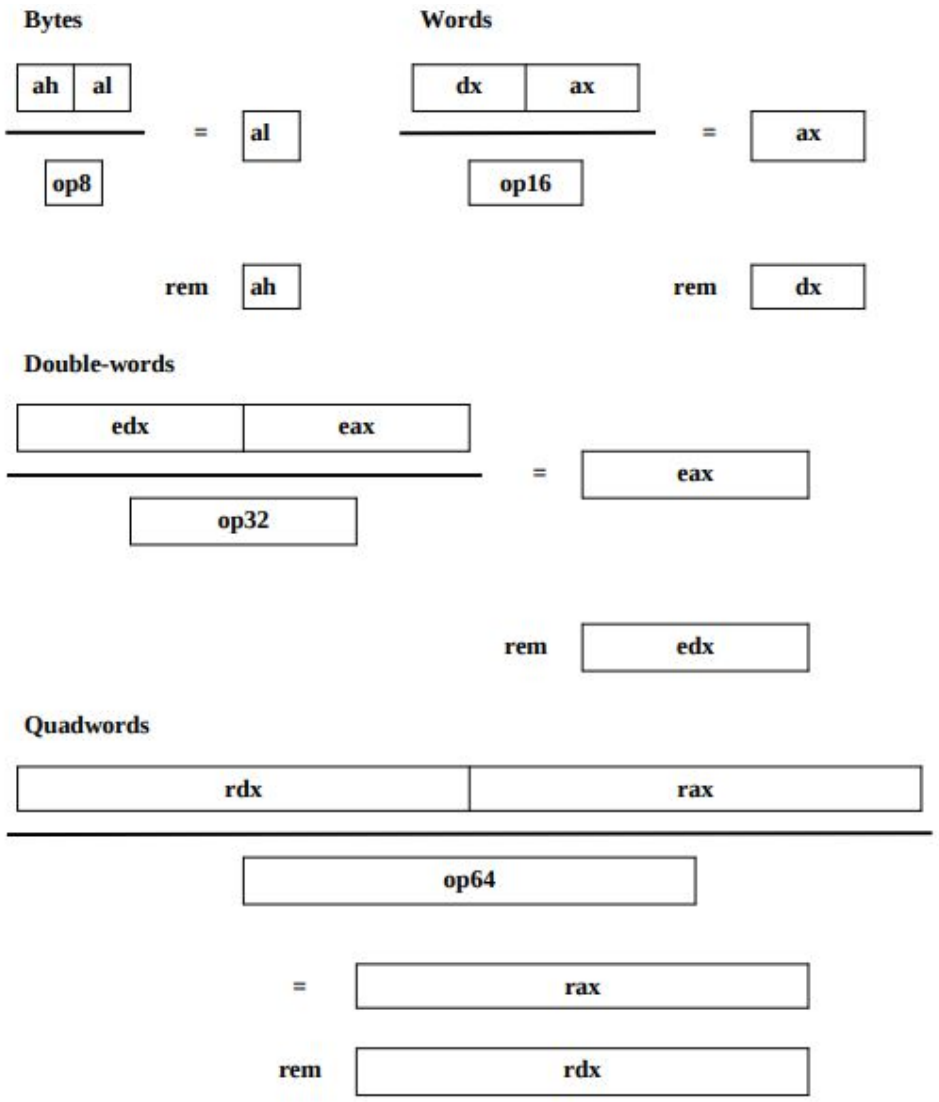
Division requires that the dividend must be a larger size than the divisor. In order to divide by an 8-bit divisor, the dividend must be 16-bits (i.e., the larger size). Similarly, a 16-bit divisor requires a 32-bit dividend. And, a 32-bit divisor requires a 64-bit dividend.

The general form of the integer division instruction is:

```
div <src>
```

where operation divides `A`, and possibly `D` registers (`ax`, `dx:ax`, `edx:eax`, or `rdx:rax`) times the `<src>`. The result will be placed in the A register (`al/ax/eax/rax`) and the remainder in either the `ah`, `dx`, `edx`, or `rdx` register.

A simple illustration on how division in Assembly works is shown below.



Example 3: Create a new file div.asm

```
global _start

section .data
      SYS_EXIT equ 60
      bNumA db 63
      bAns db 0
      bRem db 0

      wNumA dw 4321
      wNumB dw 1234
      wAns dw 0
      wRem dw 0
```

```
      qNumA dq 73000000
      qNumB dq 42000000
      qAns dq 0
      qRem dq 0

section .text
_start:
      ;bAns1 = bNumA / 3
      mov ax, 0
      mov al, byte [bNumA]
      mov bl, 3
      div bl                  ;al = ax / 3
      mov byte [bAns], al     ;bAns = ax / 3
      mov byte [bRem], ah     ;bRem = ax % 3

      ;wAns2 = wNumA / wNumB
      mov dx, 0
      mov ax, word [wNumA]
      div word [wNumB]        ;ax = dx:ax / wNumB
      mov word [wAns], ax     ;wAns = dx:ax / wNumB
      mov word [wRem], dx     ;wRem = dx:ax % wNumB
```

```
      mov rdx, 0
      mov rax, qword[qNumA]
      div qword[qNumB]        ;rax = rdx:rax / qNumB
      mov qword[qAns], rax    ;qAns = rdx:rax / qNumB
      mov qword[qRem], rdx    ;qAns = rdx:rax % qNumB

      mov rax, SYS_EXIT
      xor rdi, rdi
      syscall
```

References

Jorgensen, Ed. 2019. x86-64 Assembly Language Programming with Ubuntu. Version 1.1.40.