Assembly Language Arithmetic Operations

Exercise

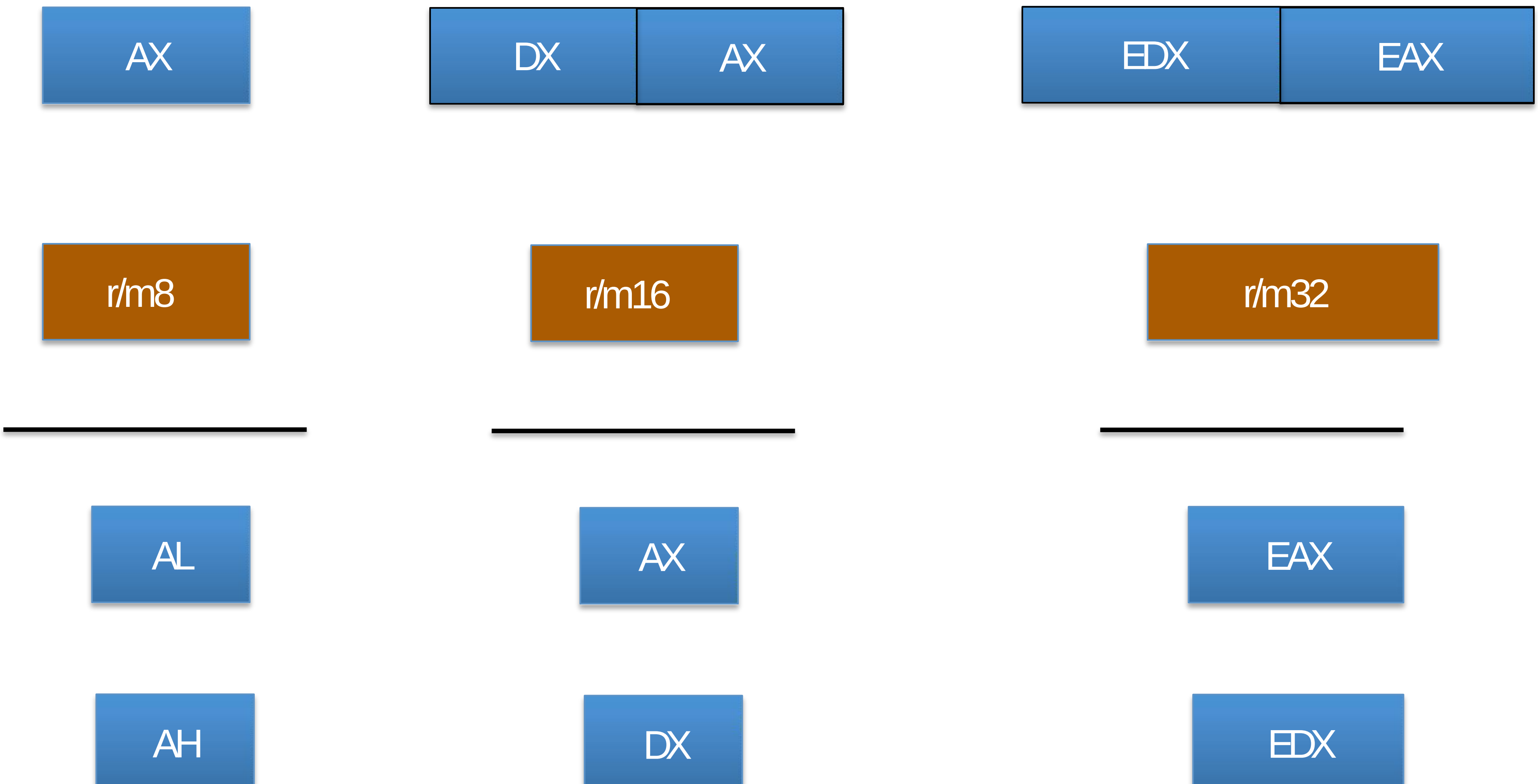
- Multiply and Divide Instructions
- GDB to trace execution and register values

Unsigned Multiply (MUL)

AL	AX	EAX
r/m8	r/m16	r/m32
AX	DX	AX	EDX	EAX

OF = 1 and CF = 1 if upper half of result is non--zero

Unsigned Divide (DIV)



Web Resources

- *Assembler References*
  - https://sourceware.org/binutils/docs/as/ (GAS Reference)
  - https://msdn.microsoft.com/en-us/library/afzk3475.aspx (MASM Reference)
  - http://www.nasm.us/doc/nasmdoc0.html (NASM Reference)
- *x86 Instruction Set Listings*
  - http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html (Volume 1, Chapter 5 Instruction Set Summary, Section 5.1)
  - http://www.felixcloutier.com/x86/
  - http://x86.renejeschke.de
  - https://en.wikipedia.org/wiki/X86_instruction_listings
  - http://www.nasm.us/doc/nasmdocb.html

Operand Definitions

Table 4.1 Operand definitions

Operand	Description
L	A literal (immediate) value (e.g., 42).
M	A memory (variable) operand (e.g., numOfStudents).
R	A register (e.g., *eax*).

Specific sizes in bits

- **L8**
- **M16**
- **M32/R32**
- **etc.**

Data Movement

GAS

```
MOVS $L/M/%R, M/%R            # MOVL $10, sum
```

MASM

```
MOV M/R, L/M/R                ; MOV eax, sum
```

NASM

```
MOV SIZE [M]/R, L/[M]/R       ; MOV DWORD [sum], edx
```

Rules

- Operands must be the same size
- Operands can’t both be memory
- IP can not be the destination

Data Movement

GAS

```
XCHGS M/%R, M/%R                # XCHGL %eax, sum
```

MASM

```
XCHG M/R, M/R                   ; XCHG eax, sum
```

NASM

```
XCHG SIZE [M]/R, [M]/R          ; XCHG DWORD [sum], edx
```

Increment/Decrement

GAS

```
INCS M/%R           # INCL sum
DECS M/%R           # DECL %eax
```

MASM

```
INC M/R             ; INC sum
DEC M/R             ; DEC eax
```

NASM

```
INC SIZE [M]/R      ; INC DWORD [sum]
DEC SIZE [M]/R      ; DEC eax
```

Add/Subtract

GAS

```
# src, dest
ADDS $L/M/%R, M/%R        # ADDW $50, sum
SUBS $L/M/%R, M/%R        # SUBL val, %eax
```

MASM

```
; dest, src
ADD M/R, L/M/R            ; ADD sum, 50
SUB M/R, L/M/R            ; SUB eax, val
```

NASM

```
; dest, src
ADD SIZE [M]/R, L/[M]/R       ; ADD WORD [sum], 50
SUB SIZE [M]/R, L/[M]/R       ; SUB eax, [val]
```

Negate

GAS

```
NEGS M/%R                  # NEGL %eax
```

MASM

```
NEG M/R                    ; NEG sum
```

NASM

```
NEG SIZE [M]/R             ; NEG BYTE [sum]
```

- Reverses the sign of a value
- Two's Complement

- The MUL instruction performs *unsigned* integer multiplication. MUL accepts a single operand, the multiplier; but what about the multiplicand and the result? The multiplicand is a value that is stored in the accumulator register based on the multiplier size (8, 16, 32, and 64-bit), as shown in the following table Also, because MUL only accepts a single operand, the programmer does not specify the result destination, it is implicit.

Unsigned Multiplication

Table 4.2 Unsigned multiplication operands

Multiplier	Multiplicand	Product
M8/R8	*al*	*ax*
M16/R16	*ax*	*dx:ax*
M32/R32	*eax*	*edx:eax*
M64/R64	*rax*	*rdx:rax*

Example 4.1 Unsigned 16-bit multiplication

```
mov ax, 64      ; Store the multiplicand for 16-bit multiplication
mov bx, 8096    ; Store the multiplier in a 16-bit register so when
                ; we multiply, AX is used as the multiplicand and
                ; a 32-bit result is produced
mul bx          ; bx * ax = 518144
                ;         = 00000000000001111110100000000000
                ;      dx = 0000000000000111, ax = 1110100000000000
```

- Both the multiplicand and product locations are implied based on the multiplier size. Consider the example of multiplying 8096 by 64 (example in previous slide). The largest value is 8096, which uses 13 bits of storage; so we are forced to store the value in at least a 16-bit location. A good chance exists that the product of two 16-bit values could exceed 16 bits of storage. Consequently, the product of a 16-bit multiplication will be stored in a 32-bit destination: *dx:ax* (the high 16 bits of the product in *dx* and the low 16 bits of the product in *ax*). The product of 8096 * 64 is 518144, which requires 19 bits of storage. The result fits in *dx:ax*, but would have overflowed a 16-bit destination.

Unsigned Multiplication

GAS

```
MULS M/%R                # MULW %ax
```

MASM

```
MUL M/R                  ; MUL ax
```

NASM

```
MUL SIZE [M]/R           ; MUL DWORD [sum]
```

- Similar to MUL, division has both unsigned and signed versions. The DIV instruction performs *unsigned* integer division and produces a two-part result: the quotient and the remainder.

- When using DIV, the programmer only specifies the divisor (bottom number in standard division notation). The dividend (top number) must be pre-loaded into the correct register based on the size of the divisor, as shown in Table in following slide. Also, the sign of the dividend value must extend into the high-bit register. For example, when performing signed division, storing a negative value in *dx:ax* requires the sign bit (1) to be extended from *ax* through *dx*, so that the value retains the sign. Sign extension is discussed later in this chapter.

Unsigned Division

Table 4.3 Unsigned division operands

Divisor	Dividend	Quotient	Remainder
M8/R8	*ax*	*al*	*ah*
M16/R16	*dx:ax*	*ax*	*dx*
M32/R32	*edx:eax*	*eax*	*edx*
M64/R64	*rdx:rax*	*rax*	*rdx*

Example 4.2 Unsigned 32-bit division

```
mov edx, 0      ; Load EDX:EAX with the dividend, 32
mov eax, 32
mov ecx, 3      ; Load ECX with the divisor, 3
div ecx         ; 32 / 3 = 10r2
                ; eax = 0000000Ah, edx = 00000002h
```

- Let us examine how 32-bit unsigned division might work to solve the problem 32 / 3 (Example in previous slide). We must pre-load the dividend into *edx:eax*, but because the positive value 32 does not extend past the 32 bits of *eax*, we need to ensure that *edx* contains 0. In other words, we need to extend the sign bit (0) for positive. Next, we choose a 32-bit register, *ecx*, to hold the divisor. After performing the division, *eax* contains the quotient 10 and *edx* contains the remainder of 2

Unsigned Division

GAS

```
DIVS M/%R        # DIVW %ax
```

MASM

```
DIV M/R          ; DIV ax
```

NASM

```
DIV SIZE [M]/R       ; DIV DWORD [sum]
```

Unsigned Division

GAS

```
DIVS M/%R        # DIVW %ax
```

MASM

```
DIV M/R          ; DIV ax
```

NASM

```
DIV SIZE [M]/R        ; DIV DWORD [sum]
```

Signed Multiplication

GAS

```
# One-operand - follows Table 4.2
IMULS M/%R                          # IMULL %ebx

# Two-operand - src, dst
IMULS $L/M/%R, %R                   # IMULW val, %ax

# Three-operand - imm, src, dst
IMULS $L, M/%R, %R                  # IMULL $10, val, %eax
```

MASM

```
# One-operand - follows Table 4.2
IMUL M/R                            ; IMUL ebx

# Two-operand - dst, src
IMUL R, L/M/R                       ; IMUL ax, val

# Three-operand - dst, src, imm
IMUL R, M/R, L                      ; IMUL eax, val, 10
```

NASM

```
# One-operand - follows Table 4.2
IMUL SIZE [M]/R                     ; IMUL DWORD [sum]

# Two-operand - dst, src
IMUL R, L/[M]/R                     ; IMUL ax, [val]

# Three-operand - dst, src, imm
IMUL R, [M]/R, L                    ; IMUL eax, [val], 10
```

Signed Division

GAS

```
IDIVS M/%R        # IDIVW %ax
```

MASM

```
IDIV M/R          ; IDIV ax
```

NASM

```
IDIV SIZE [M]/R       ; IDIV DWORD [sum]
```