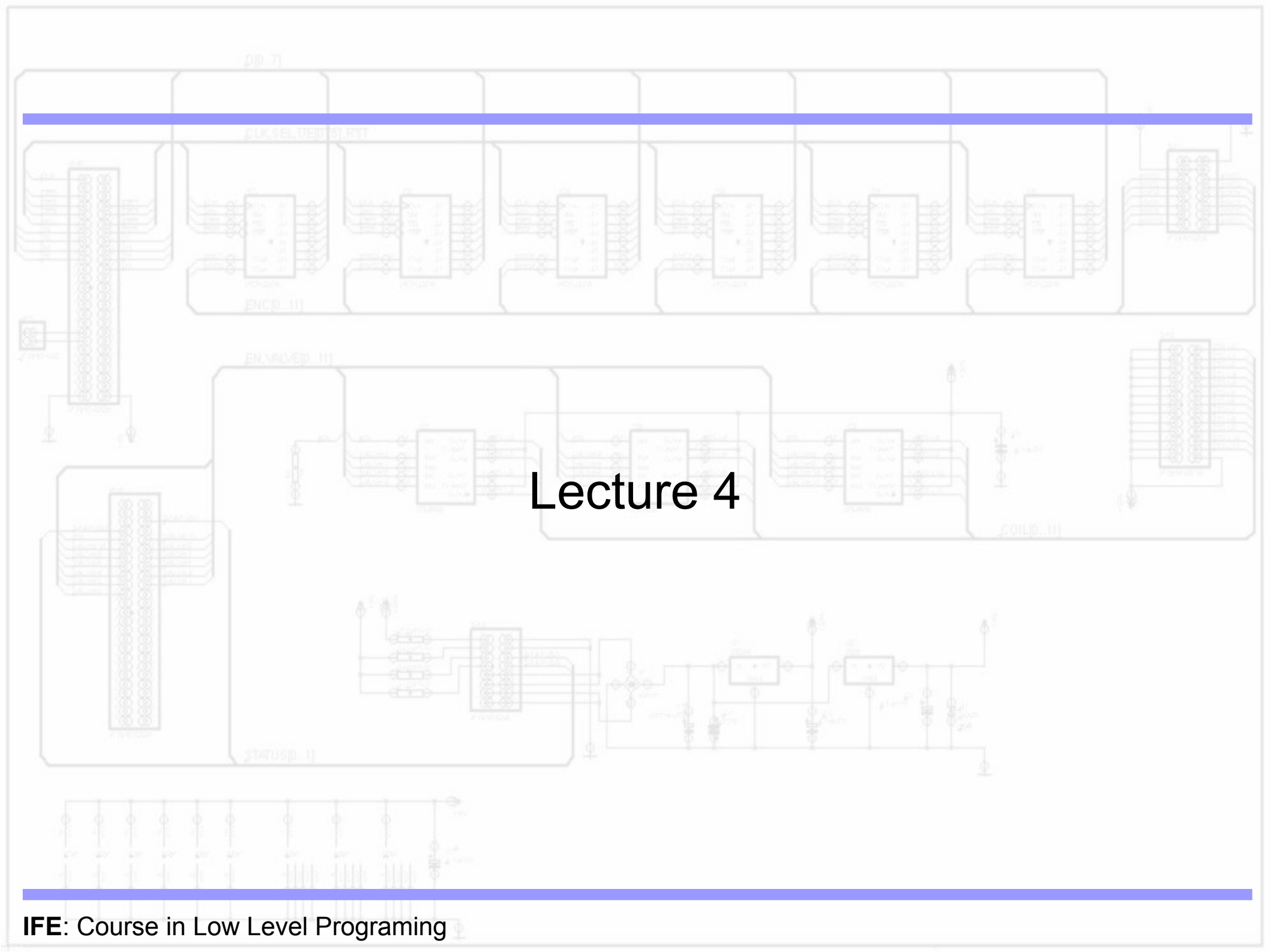

Lecture 4

Instruction Set of Intel x86 Microprocessors

Groups of instructions:

1. Data movement instructions
2. Arithmetic operations
3. Logical operations
4. String handling instructions
5. Jumps and conditional jumps
6. Loops
7. State changing instructions
8. Privileged instructions
9. SIMD instructions
10. Other groups

Instruction Set of Intel x86 Microprocessors

List of symbols and abbreviations

seg – 16-bit segment part of memory address,
offs – 16-bit offset part of memory address,
reg8 – 8-bit register (AH, AL, BH, BL, etc.),
reg16 – 16-bit register (AX, BX, SP, BP, etc.),
sreg – segment register (CS, DS, ES, etc.),
mem8 – 8-bit memory location,
mem16 – 16-bit memory location,
imm8 – immediate 8-bit value,
imm16 – immediate 8-bit value.
byte – one byte,
word – one 16-bit word,
dword – one 32-bit word, data type available starting with IA-32 Architecture.

OF – overflow flag (range exceeded for 2`s complement numbers),
DF – direction flag,
IF – interrupt flag,
SF – sign flag,
ZF – zero flag,
PF – parity flag,
CF – carry flag (range exceeded for natural binary numbers).

Instruction Set of Intel x86 Microprocessors

Data movement instructions

Transmission between memory and registers

MOV – transmission of one byte, word, dword,
PUSH – sends one word (dword) to the top of the stack,
POP – takes one word (dword) from the top of the stack,
XCHG – in place exchange of byte, word or dword,
XLAT(B)– takes byte from table using look-up translation.

Input/Output communication instructions

IN – takes byte, word or dword from input device port,
OUT – sends byte, word or dword to the port of output device.

Address loading instructions

LEA – loads an effective address to register,
LDS – loads an effective address to specified register and DS,
LES – loads an effective address to specified register and ES.
LSS – loads an effective address to specified register and SS.

Flags transmission instructions

PUSHF – sends the contents of FLAGS register to the top of the stack,
POPF – takes the contents of FLAGS register from the top of the stack.

Instruction Set of Intel x86 Microprocessors

Transmission of data between memory and registers

MOV (from *move)*

Moves data from right to the left argument.
Arguments can be represented by registers, memory locations, immediate values, segment registers.
Affects no flags.

Available variants:

MOV r/m8,r8	MOV r/m16,r16	MOV r/m32,r32
MOV r8,r/m8	MOV r16,r/m16	MOV r32,r/m32
MOV r16,sreg	MOV sreg, r16	
MOV r/m8,imm8	MOV r/m16,imm16	MOV r/m32,imm32

mxx – stands for memory location of appropriate width **xx.** It can be addressed as follows:
seg:{[(E)BX,(E)BP,(E)SP]}{[(E)DI,(E)SI]}{imm8,16,32}.

Notice.
For (E)BX by default seg is taken from DS,
for (E)BP or (E)SP seg must be taken from SS.

Instruction Set of Intel x86 Microprocessors

EXAMPLES

```
(...)
MOV AL, AH
MOV CX,AX
(…)

(...)
a db ?
b dw 0A00h

MOV a,AL
MOV BX,b
MOV BX,DS:b
MOV BX,DS:[offset b]
MOV AX,seg b
MOV DS,AX
MOV BX,offset b
MOV DS:[BX],WORD PTR 2510h
MOV DI,0
MOV DS:[BX][DI+1],BYTE PTR 13h
(…)
```

Instruction Set of Intel x86 Microprocessors

EXAMPLES cont.

```
(...)
tab db 100h dup(?)

(…)
MOV AX,seg tab
MOV DS,AX
MOV BX,offset tab
MOV DI,0
MOV AL,DS:[BX][DI]
(…)
MOV AL,tab[DI]          ; convenient  way of addressing elements in tables
(…)                     ; in C we would have tab[index]
```

Instruction Set of Intel x86 Microprocessors

PUSH

Moves word (dword) given as the argument to the top of the stack that is memory location indicated by SS:[SP].
Argument can be given in the form of memory location or register.
Additionally SP is decreased by 2 (respectively by 4 with dword argument).
Affects no flags.

Available variants:

PUSH r/m16 PUSH r/m32 ; for IA-32 Architecture
PUSH sreg

EXAMPLES

```
(...)
PUSH AX
PUSH DS
(…)
a dw 0123h

PUSH a
(…)
```

Instruction Set of Intel x86 Microprocessors

POP

Passes word (dword) from the top of the stack (i.e. SS:[SP]) to the given argument.
Argument can be given in the form of memory location or register.
Additionally SP is increased by 2 (respectively by 4 with dword argument).
Affects no flags.

Available variants:

```
POP r/m16          POP r/m32    ; for IA-32 Architecture
POP sreg
```

EXAMPLES

```
    (...)
    POP DS
    POP AX
    (…)
    a dw 0123h

    MOV AX,0010h
    PUSH AX
    POP a
    (…)
```

Instruction Set of Intel x86 Microprocessors

XCHG (from *exchange)*

Moves data from right to the left argument and from left to the right one. We can say that the contents of two arguments are exchanged in place.
Arguments can be represented by registers or memory locations.
Affects no flags.

Available variants:

XCHG r/m8,r8 XCHG r/m16,r16 XCHG r/m32,r32

EXAMPLES

```
(…)
a dw 1000h
b dw 0010h

MOV AX,a
XCHG AX,b
MOV a,AX
(…)
```

Instruction Set of Intel x86 Microprocessors

XLAT/XLATB

Instruction sets AL to memory byte from location DS:[(E)BX+AL].
Arguments can be represented by registers or memory locations.
Affects no flags.

Available variants:

```
XLAT m8      ; Notice. Argument is available in assembly only for documentation purposes
XLATB
```

EXAMPLES

```
    (…)
    tab db 100h dup(?)
    index db 10h

    MOV AL,index
    MOV BX, offset tab
    XLAT tab
    (…)
```

Instruction Set of Intel x86 Microprocessors

IN (from *input*)

Instruction takes byte or word (dword) from given port to the accumulator register.
Argument can be provided in the form of DX register or immediate value.
Affects no flags.

Available variants:

```
IN AL,DX      IN AX,DX      IN EAX,DX
IN AL,imm8    IN AX,imm8    IN EAX,imm8
```

EXAMPLES

```
    (…)
    port dw 0060h

    MOV DX,port
    IN AX,DX
    (…)

    (…)
    IN AL,60h
    (…)
```

Instruction Set of Intel x86 Microprocessors

OUT (from *output*)

Instruction sends byte or word (dword) from accumulator register to the output port.
Argument can be provided in the form of DX register or immediate value.
Affects no flags.

Available variants:

OUT DX,AL	OUT DX,AX	OUT DX,EAX
OUT imm8,AL	OUT imm8,AX	OUT imm8,EAX

EXAMPLES

```
    (…)
    port dw 03C7h

    MOV DX,port
    MOV Al,00h
    OUT DX,AL
    MOV AL,0FFh
    MOV DX,03C8h
    OUT DX,AL
    OUT DX,AL
    OUT 03C8h,AL
    (…)
```

Instruction Set of Intel x86 Microprocessors

LEA (from *load effective address*)

Computes effective address of the second operand and stores it in the first operand. Argument can be provided in the form of memory offset. This instruction is used when the effective address must be calculated during program execution. Otherwise programmer should use offset directive.
Affects no flags.

Available variants:
LEA r16,m LEA r32,m

EXAMPLES

```
(…)
tab db 100h dup(?)

LEA tab[BX][DI]
(…)
```

Instruction Set of Intel x86 Microprocessors

LDS/LES/LSS (from *load segment register*)

Instruction loads logical address in the form of segment:offset from memory to DS (ES) register and any non-segment register provided as an argument.
LDS/LES takes two arguments: non-segment register and memory location.
Affects no flags.

Available variants:
LDS r16,m16:16 LDS r32,m16:32

Starting from IA-32 Architecture there are available additional segment registers FS, GS and hence adequate variants of that instruction, i.e. LFS, LGS.
It should be noted that upper word is loaded into segment register and the lower word goes into argument register.

EXAMPLES

```
(…)
addresses dd 10h dup(?)
pointer dw 5678h, 0000h

LDS BX,addresses[DI]
LES SI,pointer
(…)
```

Instruction Set of Intel x86 Microprocessors

PUSHF

Moves contents of FLAGS register to the top of the stack.
Instruction takes no arguments. Additionally SP is decreased by 2 (respectively by 4 in 32-bit operational mode).
Affects no flags.

Available variants:

PUSHF

EXAMPLES

```
(...)
PUSHF
ADD AX,BX
POPF
(…)
```

Instruction Set of Intel x86 Microprocessors

POPF

Takes word (dword) from the top of the stack and loads it into FLAGS register.
Instruction takes no arguments. Additionally SP is increased by 2 (respectively by 4 in 32-bit operational mode).
Affects no flags.

Available variants:

POPF

EXAMPLES

```
(...)
PUSHF
ADD AX,BX
POPF
(…)
```

Instruction Set of Intel x86 Microprocessors

Arithmetic operations

ADD – adds two arguments,
SUB – subtracts one argument from the other,
INC – increments by 1 given argument,
DEC – decrements by 1 given argument,
NEG – change of sign for 2`s complement numbers,
CMP – compares two arguments.

MUL – multiplication of two arguments (unsigned numbers),
DIV – division of two arguments (unsigned numbers).

IMUL – multiplication of signed numbers,
IDIV – division of signed numbers.

Instruction Set of Intel x86 Microprocessors

ADD (from *add byte, word or double word)*

Instruction adds the contents of the first operand to the second one and the result is stored in the first operand.
Arguments can be provided in the form of registers, memory locations and immediate values.
Affects: OF, SF, ZF, PF, CF.

Available variants:

ADD r/m8,r8	ADD r/m16,r16	ADD r/m32,r32
ADD r8,r/m8	ADD r16,r/m16	ADD r32,r/m32
ADD r/m8,imm8	ADD r/m16,imm16	ADD r/m32,imm32

EXAMPLES

```
(…)
a db 10h
b db 20h
c db ?

MOV AL,a
ADD AL,b
MOV c,AL
(…)
```

Instruction Set of Intel x86 Microprocessors

EXAMPLES cont.

After execution we should obtain as a result number 30h in variable c. Such number is positive. Hence SF=0 and ZF=0. While operating both on natural and 2`s complement numbers there is no range exceed. Therefore OF and CF flags are also equal zero. Integer 30h has even number of 1s (0s) in binary notation and that's why PF equals 1.

```
(…)
MOV AL,100
ADD AL,AL
(…)
```

In this example result is correct in binary form but incorrect in 2`s complement notation (we add two positive numbers but result is negative). Hence CF=1 and SF=1. Moreover: OF=0, PF=0 (200d=11001000b), ZF=0.

```
(…)
MOV AL,200
ADD AL,56
(…)
```

The range is exceeded for natural numbers (CF=1). Moreover ZF=1, PF=1. However the result is correct for 2`s complement (OF=0).

Instruction Set of Intel x86 Microprocessors

SUB (from *subtract byte, word or double word*)

Instruction subtracts the content of the second argument from the first one and stores the result in the first argument.
Arguments can be provided in the form of registers, memory locations and immediate values.
Affects: OF, SF, ZF, PF, CF.

Available variants:

SUB r/m8,r8	SUB r/m16,r16	SUB r/m32,r32
SUB r8,r/m8	SUB r16,r/m16	SUB r32,r/m32
SUB r/m8,imm8	SUB r/m16,imm16	SUB r/m32,imm32

EXAMPLES

```
(…)
a db 0Ah
b db 10h

MOV AL,a
SUB AL,b        ; Flags: OF=0, CF=1, ZF=0, PF=1.
(…)
```

Instruction Set of Intel x86 Microprocessors

INC (from *increment by 1 byte, word or double word*)

Instruction increments by 1 (adds one) the content of its argument.
Argument can be provided in the form of register or memory location.
Affects: OF, SF, ZF, PF.

Available variants:

INC r/m8 INC r/m16 INC r/m32

EXAMPLES

```
(…)
a db 0Ah

INC AL
INC a
(…)
```

Notice. The INC instruction does not affect CF flag.

Instruction Set of Intel x86 Microprocessors

DEC (from *decrement by 1 byte, word or double word*)

Instruction decrements by 1 (subtructs one) the content of its argument.
Argument can be provided in the form of register or memory location.
Affects: OF, SF, ZF, PF.

Available variants:

DEC r/m8 DEC r/m16 DEC r/m32

EXAMPLES

```
(…)
a dw 000Ah

DEC BX
DEC a
(…)
```

Notice. The DEC instruction does not affect CF flag.

Instruction Set of Intel x86 Microprocessors

NEG (from *negate byte, word or double word*)

Instruction subtracts the content of its argument from 0. Such operation is equivalent to sign change of a number in 2`s complement code.
Argument can be provided in the form of register or memory location.
Affects: OF, SF, ZF, PF, CF.

Available variants:

NEG r/m8 NEG r/m16 NEG r/m32

EXAMPLES

```
(…)
a dw 000Ah

NEG a
MOV AX, 1234h
ADD AX,a
(…)
```

Instruction Set of Intel x86 Microprocessors

CMP (from *compare byte, word or double word)*

The CMP instruction subtracts its second argument from the first one, flags are affected according to the result but result itself is not stored in any of registers.
Arguments can be provided in the form of registers, memory locations and immediate values.
Affects: OF, SF, ZF, PF, CF.

Available variants:

CMP r/m8,r8	CMP r/m16,r16	CMP r/m32,r32
CMP r8,r/m8	CMP r16,r/m16	CMP r32,r/m32
CMP r/m8,imm8	CMP r/m16,imm16	CMP r/m32,imm32

EXAMPLES

```
(…)
a db 10h
b db 20h
MOV AL,b
CMP a,AL
(…)
```

Notice. The CMP instruction followed by conditional jumps constitutes IF instruction known from high level languages.

Instruction Set of Intel x86 Microprocessors

MUL (from *multiply byte, word or double word unsigned)*

Instruction multiplies two unsigned arguments. The first argument is by default hold in AL, AX or EAX register. The size of register depends on the size of argument provided explicitly (AL for 8-bit, AX for 16-bit and EAX for 32-bit argument). The result is stored in AX, DX:AX or EDX:EAX register(s) respectively. Argument can be provided in the form of register or memory location.
Affects: OF, SF, ZF, PF, CF (OF=CF=0 if upper part of result equals 0, otherwise CF=OF=1, the contents of remaining flags are undefined).

Available variants:

MUL r/m8 MUL r/m16 MUL r/m32

EXAMPLES

```
(…)
a db 10h
b db 20h
c dw ?
MOV AL,a
MUL b
MOV c,AX
(…)
```

Instruction Set of Intel x86 Microprocessors

DIV (from *divide byte, word or double word unsigned)*

Instruction divides two unsigned arguments. The first argument is by default hold in AX, DX:AX or EDX:EAX register(s). The size of first argument depends on the size of argument provided explicitly (AX for 8-bit, DX:AX for 16-bit and EDX:EAX for 32-bit argument). The result is stored in AX, DX:AX or EDX:EAX register(s) respectively in the form of quotient (lower part – AL, AX, EAX) and reminder (upper part – AH, DX, EDX). The argument can be provided in the form of register or memory location.
Affects: OF, SF, ZF, PF, CF (the contents of those flags are undefined).

Available variants:

DIV r/m8 DIV r/m16 DIV r/m32

EXAMPLES

```
(…)
a db 10h
b db 20h
c db ?
MOV AL,a
DIV b
MOV c,AL
(…)
```

Instruction Set of Intel x86 Microprocessors

The IMUL and IDIV instructions are counterparts of MUL and DIV that operate on signed numbers, that is numbers in 2`s complement code.