Instruction set

categories:

- ⇒ Data transfer group
- ⇒ Arithmetic group
- ⇒ Bit manipulation group
- ⇒ string Instruction group
- ⇒ program transfer instruction group
- ⇒ process control instruction group

Data Transfer group

- ⇒ General purpose
- ⇒ Input output
- ⇒ Address object
- ⇒ Flag Transfer

Data Transfer GROUP

- ⇒ General purpose

=> LAR

=> MOV

=> BSWAP

=> MOV CRN

=> POPA

=> MOV DRN

=> PUSHA

=> MOV TRN

=> POPAD

=> PUSH

=> POPD

=> POP

=> PUSHAD

=> XCHG

=> PUSHD

=> XLAT

=> MOVSX

=> MOVZX

1. MOV

=> MOV: copy a word or a byte or doubleword or quadword to from register memory

=> The MOV instruction cannot:

=> set the value of the CS and IP registers.

=> copy value of one segment register to another segment register (should copy to general register first).

=> Algorithm:

=> Operand1 = Operand2

MOVS

=> move string data

MOVSX

=> Modifies flags: None

=> Copies the value of the source operand to the destination register with the sign

extended.

=> The source register or memory is 8 16 bits, while the destination register or memory is
16 32 bits

MOVZX

=> Modifies flags: None

=> Copies the value of the source operand to the destination register with the zero
extended.

=> The source register or memory is 8 16 bits, while the destination register or memory
is 16 32 bits

PUSH

=> Usage: PUSH src

=> PUSH immed 80188 only

=> Modifies flags: None

=> Decrements SP by the size of the operand (two or four, byte values are

sign

extended and transfers one word from source to the stack top (SS:SP).

PUSHA PUSHAD

=> usage: PUSHA

=> PUSHAD (386)

=> Modifies flags: None

=> Pushes all general purpose registers onto the stack in the following order: (E)AX, (E)CX, (E)DX, (E)BX, (E)SP, (E)BP, (E)SI, (E)DI. The value of SP is the value before the actual push of SP.

POP

=> usage: POP dest

=> Modifies flags: None

=> Transfers word at the current stack top (SS:SP) to the destination then increments SP by two to point to

the new stack top. CS is not a valid destination.

POPA POPAD

=> usage: POPA

=> POPAD (386+)

=> Modifies flags: None

=> POPs the top 8 words off the stack into the 8 general purpose 16 32 bit registers. Registers are POPped in the following order: (E)DI, (E)SI, (E)BP, (E)SP, (E)DX, (E)CX and (E)AX. The (E)SP value POPped from the stack is actually discarded.

XCHG

=> usage: XCHG dest,src

=> Modifies flags: None

=> Exchanges contents of source and destination.

XCHG

XLAT XLATB

=> usage: XLAT translation-table

=> XLATB cmasm 5.X)

=> Modifies flags: None

=> Replaces the byte in AL with byte from a user table addressed by BX. The original

value of AL is the index into the translate table. The best way to describe this is

MOV AL,[BX AL]

LAR - LOad Access Rights C286

Protected

=> usage: LAR dest,src

=> Modifies Flags: ZF

=> The high byte of the of the destination register is overwritten by the value

of the access rights byte and the low order byte is zeroed depending on the selection in the source

Operand. The zero flag is set if the load
Operation is successful.

BSWAP: Byte swap (486)

=> Usage: BSWAP reg32

=> Modifies flags: none

=> Changes the byte order of a 32 bit register from
big endian to little endian or vice versa. Result left
in destination register is undefined if the Operand is
a 16 bit register.

Input output Instruction

=> In

=> OUT

=> INS

=> OUTS

IN

=> usage: IN accum, PORT

=> Modifies flags: None

=> A byte, WORD or dWORD is read from "PORT" and placed in AL, AX or EAX respectively. If the PORT number is in the range of 0-255 it can be specified as an immediate, Otherwise the PORT number must be specified in DX. Valid PORT ranges on the PC are 0-1024, though values through 65535 may be specified and recognized by third party vendors and PS 2's.

INS: Input string from PORT (80188)

=> usage: INS dest, PORT

=> INSB

=> INSW

=> INSD (386)

=> Modifies flags: None

=> Loads data from PORT to the destination ES:(E)DI (even if a

destination operand
is supplied. (E)DI is
adjusted by the size of the operand and
increased if the
Direction flag is cleared and
decreased if the Direction flag is set. For
INSB,
INSW, INSD no operands are
allowed and the size is determined by the
mnemonic.

OUT

⇒ usage: OUT port,accum
⇒ Modifies flags: None
⇒ Transfers byte in AL, word in AX
or dword in EAX to the specified
hardware port
address. If the port number is in
the range of 0-255 it can be
specified as an
immediate. If greater than 255
then the port number must be
specified in DX. since
the PC only decodes 10 bits of

the PORT address, values over 1023 can only be decoded by third party vendor equipment and also map to the PORT range 0-1023.

OUTS

=> usage: OUTS PORT,SRC
=> OUTSB
=> OUTSW
=> OUTSD (386)
=> Modifies flags: None
=> Transfers a byte, word or doubleword from "SRC" to the hardware PORT specified in DX. For instructions with no operands the "SRC is located at DS:SI and SI is incremented or decremented by the size of the operand or the size dictated by the instruction format. When the Direction Flag is set SI is decremented, when clear, SI is incremented. If the PORT

number is in the range of 0-255 it can be specified as an
immediate. If greater than 255 then the port number must be specified in DX. since
the pc only decodes 10 bits of the port address, values over 1023 can only be decoded by third party vendor equipment and also map to the port range 0-1023.

Address object

=> These instructions manipulate the addresses of the variables, rather than the contents of the variables.

=> LEA

=> LDS

=> LES

=> LFS

=> LGS

=> LSS

LEA: LOad Effective Address

=> Usage: LEA dest,src

=> Modifies Flags: None

=> Transfers Offset address of "src" to the destination register.

LDS: Load Pointer with DS

LES: Load Pointer using ES

LFS,LGS,LSS

=> LES: Load Pointer using ES

=> LGS: Load Pointer using GS

=> LSS: Load Pointer using SS

Flag Transfer Instructions

=> LAHF: Load AH Register from flags

=> SAHF: Store AH Register in flags

=> PUSHF PUSHFD: PUSH flags on to stack

=> POPF POPFD: POP flags off

stack POP Eflags off stack

LAHF: Load AH register from flags

SAHF: Store AH register in flags

PUSHF PUSHFD: PUSH flags Eflags on to stack

POPF POPFD: POP flags off stack POP Eflags off stack

CMPXCHG: Compare and Exchange

Arithmetic Instructions

=> Addition
=> Subtraction
=> Multiplication
=> Division

Arithmetic Instructions: Addition

=> ADD: Add byte OR WORd OR doUbleWORd OR qUadWORd

=> ADC: Add byte OR WORd OR doUbleWORd OR qUadWORd with carry

=> INC: Increment byte OR WORd OR doUbleWORd OR qUadWORd by 1

=> AAA: ASCII adjUst for addition

=> DAA: Decimal adjUst for addition

ADD: Add byte OR WORd

ADC: Add byte OR WORd with carry

INC: Increment byte OR WORd by 1

AAA: ASCII adjUst for addition

DAA: Decimal adjUst for addition

Arithmetic Instructions: subtraction

⇒ SUB

⇒ SBB

⇒ DEC

⇒ NEG

⇒ AAS

⇒ DAS

⇒ CMP

⇒ CMPXCHG: Compare and exchange

⇒ CMPXCH8B: Compare and exchange 8 byte

SUB: Subtract byte OR word OR doubleword OR quadword

SBB: Subtract byte OR word OR doubleword OR quadword with borrow

DEC: Decrement byte OR word OR doubleword OR quadword by 1

NEG: Negate byte OR word OR doubleword OR quadword

AAS: ASCII adjust after subtraction

DAS: Decimal adjust after subtraction

CMP: Compare byte OR WORD OR DOUBLE WORD OR QUADWORD

Arithmetic Instructions:

Multiplication

=> MUL

=> IMUL

=> AAM

MUL: Multiply byte OR WORD OR DOUBLEWORD OR QUADWORD unsigned

Multiplies AL, AX, EAX OR RAX by a source operand. If the source is 8 bits, it is multiplied by AL and the product is stored in AX. If the source is 16 bits, it is

multiplied by AX and
the product is stored in
DX:AX. If the source is
32 bits, it is multiplied
by EAX and the
product is stored in
EDX:EAX. If the source
is 64 bits, it is multiplied
by RAX and the
product is stored in RDX:RAX.

IMUL: Multiply byte or word
(signed)

performs a signed integer
multiplication on AL, AX,
EAX or RAX. If the
multiplier is 8 bits, the
multiplicand is AL and the
product is AX. If the
multiplier is 16 bits, the
multiplicand is AX and the
product is DX:AX. If the
multiplier is 32 bits, the
multiplicand is EAX and the
product is EDX:EAX. The

carry and overflow flags
are set if a 16-bit product
extends into AH, or a 32-bit
product extends into DX, or
a 64-bit product extends
into EDX.

AAM: ASCII Adjust after
multiplication

Arithmetic Instructions: Division

=> DIV

=> IDIV

=> AAD

=> CBW

=> CWD

=> CWDE: convert word to double word
extended

=> CDQ

DIV: unsigned Divide

performs either 8-, 16-,
32-bit or 64-bit unsigned

integer division. If the
divisor is 8 bits, the
dividend is AX, the
quotient is AL, and the
remainder is AH. If the
divisor is 16 bits, the
dividend is DX:AX, the
quotient is AX, and the
remainder is DX. If the
divisor is 32 bits, the
dividend is EDX:EAX,
the quotient is EAX, and
the remainder is EDX. If
the divisor is 64 bits, the
dividend is RDX:RAX,
the quotient is RAX, and
the remainder is RDX.

IDIV: signed Divide

⇒ performs a signed integer
division operation on
EDX:EAX, DX:AX, or AX. If
the divisor is 8 bits, the
dividend is AX, the quotient

is AL, and the remainder is AH. If the divisor is 16 bits, the dividend is DX:AX, the quotient is AX, and the remainder is DX. If the divisor is 32 bits, the dividend is EDX:EAX, the quotient is EAX, and the remainder is EDX. usually the IDIV operation is prefaced by either CBW or CWD to sign-extend the dividend.

AAD: ASCII Adjust before Division

CBW: convert byte into word

CWD: convert word to double

CDQ: convert doubleword to quadword

Bit manipulation Instruction

=> Logical
=> Shifts
=> Rotates

Logical Instructions

=> NOT
=> AND
=> OR
=> XOR
=> TEST
=> BT
=> BTS
=> BTR
=> BTC
=> BSF
=> BSR
=> SETcc

NOT: NOt byte OR WORd OR dOUble

WORd

AND: ANd byte OR WORd OR dOUble

WORd

OR: Inclusive OR byte OR WORd OR

double word

XOR: Exclusive OR byte OR word OR double word

TEST: Test byte OR word OR double word

word

BT: Bit Test

BTS: Bit Test and Set

BTS: Bit Test and Reset

BTC: Bit Test and Complement

BSF: Bit Scan FORWARD

BSR: Bit Scan Reverse

Shift Instructions

=> SHL

⇒ SAL

⇒ SHR

⇒ SAR

⇒ SHLD

⇒ SHRD

SHL: Shift logical left byte OR word OR double word

SAL: Shift arithmetic left byte OR word OR double word

SHR: Shift logical right byte OR word OR double word

SAR: Shift arithmetic right byte OR word OR double word

SHLD: Double-precision shift left (386)

SHLR: Double-precision shift right(386)

Rotate Instructions

⇒ ROL

⇒ ROR

⇒ RCL

⇒ RCR

ROL: Rotate Left

ROR: Rotate Right

RCL: Rotate through carry Left

RCR: Rotate through carry Right

String Instructions GROUP

⇒ MOVSB MOVS$_W$ MOVSD

⇒ CMPSB CMPS$_W$ CMPSD

⇒ SCASB SCAS$_W$ SCASD

⇒

LOADSB LOADS$_W$ LOADSD

⇒ STOSB STOS$_W$ STOSD

⇒ REP

⇒ REPE REPZ

⇒ REPNE REPNZ

Program control (OR Branch) GROUP Instruction

=> unconditional Transfer
=> conditional Transfer
=> Iteration control
=> Interrupts

Process control Instructions

=> Flag operations
=> NO operation
=> External synchronization

Flag operations

=> CLC
=> STC
=> CMC
=> STD

⇒ CLD

⇒ STI

⇒ CLI

CLC: clear carry flag

STC: set carry flag

CMC: complementary carry flag

CLD: clear Direction flag

STD: set Direction flag

CLI: clear Interrupt flag

STI: set Interrupt flag

NO operation

External synchronization

⇒ HLT

⇒ WAIT

=> ESC

=> LOCK

=> BOUND

=> ENTER

=> LEAVE

HLT: Hlt until interrupt or reset

WAIT: wait till Interrupt

ESC: Escape

LOCK: Lock BUS during next Instruction

BOUND: Check against array bound

ENTER

LEAVE

unconditional transfer

=> CALL

⇒ RET

⇒ RETF

⇒ JMP

⇒ BOUND

⇒ ENTER

⇒ LEAVE

CAL: call a PROcedure

RET: Return from PROcedure

RETF: Return from far PROcedure

JXX: JUmP INstrUctiONs (CONditiONal UNCONditiONal)

JXX: JUmP INstrUctiONs CONt..)

Iteration control Instructions

⇒ LOOP

⇒ LOOPE LOOPZ

⇒ LOOPNE LOOPNZ

LOOP

LOOPE

LOOPZ

LOOPNE

LOOPNZ

INTERRUPTS

⇒ INTO

⇒ IRET

INT: INTERRUPT

INTO: INTERRUPT OVERFLOW

IRET: INTERRUPT RETURN

PROTECTED MODE SPECIFIC INSTRUCTIONS

=> ARPL

=> CLTS

=> LAR

=> LLDT

=> LGDT

VERR VERW

=> LIDT

=> LMSW

=> SGDT

=> SIDT

=> SLDT

=> SMSW

=> STR

=>

=> LTR

=> LSL