Complete 8086 instruction set

	CMPSB				MOV		
AAA	CMPSW	JAE	JNBE	JPO	MOVSB	RCR	SCASB
AAD	CWD	JB	JNC	JS	MOVSW	REP	SCASW
AAM	DAA	JBE	JNE	JZ	MUL	REPE	SHL
AAS	DAS	JC	JNG	LAHF	NEG	REPNE	SHR
ADC	DEC	JCXZ	JNGE	LDS	NOP	REPNZ	STC
ADD	DIV	JE	JNL	LEA	NOT	REPZ	STD
AND	HLT	JG	JNLE	LES	OR	RET	STI
CALL	IDIV	JGE	JNO	LODSB	OUT	RETF	STOSB
CBW	IMUL	JL	JNP	LODSW	POP	ROL	STOSW
CLC	IN	JLE	JNS	LOOP	POPA	ROR	SUB
CLD	INC	JMP	JNZ	LOOPE	POPF	SAHF	TEST
CLI	INT	JNA	JO	LOOPNE	PUSH	SAL	XCHG
CMC	INTO	JNAE	JP	LOOPNZ	PUSHA	SAR	XLATB
CMP	IRET	JNB	JPE	LOOPZ	PUSHF	SBB	XOR
	JA				RCL		

Operand types:

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
SREG: DS, ES, SS, and only as second operand: CS.
memory: [BX], [BX+SI+7], variable, etc...
immediate: 5, -24, 3Fh, 10001101b, etc...

8086 CPU Emirlerinin Bazıları

intsruction	*operands*
MOV/ XOR/ AND/ OR/ ADC/ ADD/ CMP	REG, memory memory, REG REG, REG memory, immediate REG, immediate
OUT/IN	DX, AL DX, AX
SHL/ SHR/ ROL/ ROR RCL/RCR	memory, immediate REG, immediate memory, CL REG, CL
JMP/ JZ/JNZ /LOOP	label
PUSH / POP	REG SREG memory
XLAT	

Instructions in alphabetical order:

Instruction	Operands	Description
ADC	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Add with Carry. Algorithm: operand1 = operand1 + operand2 + CF C Z S O P A r r r r r r
ADD	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Add. Algorithm: operand1 = operand1 + operand2 C Z S O P A r r r r r r
AND	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical AND between all bits of two operands. Result is stored in operand1. These rules apply: C Z S O P 0 r r 0 r
CALL	procedure name label 4-byte address	Transfers control to procedure, return address is (IP) is pushed to stack. *4-byte address* may be entered in this form: 1234h:5678h, first value is a segment second value is an offset (this is a far call, so CS is also pushed to stack). C Z S O P A unchanged

CBW	No operands	Convert byte into word.
		
		Algorithm:
		
		if high bit of AL = 1 then:
		AH = 255 (0FFh)
		
		else
		AH = 0
		
		C Z S O P A
		unchanged

CLC	No operands	Clear Carry flag. Algorithm: CF = 0 C 0
CLD	No operands	Clear Direction flag. SI and DI will be incremented by chain instructions: CMPSB, CMPSW, LODSB, LODSW, MOVSB, MOVSW, STOSB, STOSW. Algorithm: DF = 0 D 0
CLI	No operands	Clear Interrupt enable flag. This disables hardware interrupts. Algorithm: IF = 0 I 0
CMC	No operands	Complement Carry flag. Inverts value of CF. Algorithm: if CF = 1 then CF = 0 if CF = 0 then CF = 1 C r
CMP	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Compare. Algorithm: operand1 - operand2 result is not stored anywhere, flags are set (OF, SF, ZF, AF, PF, CF) according to result. C Z S O P A r r r r r r
CWD	No operands	Convert Word to Double word. Algorithm: if high bit of AX = 1 then:

		DX = 65535 (0FFFFh) else DX = 0 C Z S O P A unchanged
DEC	REG memory	Decrement. Algorithm: operand = operand - 1 Z S O P A r r r r r CF - unchanged!
DIV	REG memory	Unsigned divide. Algorithm: when operand is a **byte**: AL = AX / operand AH = remainder (modulus) when operand is a **word**: AX = (DX AX) / operand DX = remainder (modulus) C Z S O P A ? ? ? ? ? ?
HLT	No operands	Halt the System. C Z S O P A unchanged
IDIV	REG memory	Signed divide. Algorithm: when operand is a **byte**: AL = AX / operand AH = remainder (modulus) when operand is a **word**: AX = (DX AX) / operand DX = remainder (modulus) C Z S O P A ? ? ? ? ? ?
IMUL	REG memory	Signed multiply. Algorithm: when operand is a **byte**:

		AX = AL * operand. when operand is a **word**: (DX AX) = AX * operand. C Z S O P A r ? ? r ? ? CF=OF=0 when result fits into operand of IMUL.
IN	AL, im.byte AL, DX AX, im.byte AX, DX	Input from port into **AL** or **AX**. Second operand is a port number. If required to access port number over 255 - **DX** register should be used. C Z S O P A unchanged
INC	REG memory	Increment. Algorithm: operand = operand + 1 Z S O P A r r r r r CF - unchanged!
INT	immediate byte	Interrupt numbered by immediate byte (0..255). Transfer control to interrupt procedure Algorithm: Push to stack: flags register CS IP

LOOP	label	Decrease CX, jump to label if CX not zero.
		
		Algorithm:
		CX = CX - 1
		if CX <> 0 then
		jump
		else
		no jump, continue
		Example:
		
		C Z S O P A
		unchanged

MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate SREG, memory memory, SREG REG, SREG SREG, REG	Copy operand2 to operand1. 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). copy immediate value to segment register (should copy to general register first). Algorithm: operand1 = operand2

		C Z S O P A unchanged
MUL	REG memory	Unsigned multiply. Algorithm: when operand is a byte: AX = AL * operand. when operand is a word: (DX AX) = AX * operand. C Z S O P A r ? ? r ? ? CF=OF=0 when high section of the result is zero.
NEG	REG memory	Negate. Makes operand negative (two's complement). Algorithm: Invert all bits of the operand Add 1 to inverted operand C Z S O P A r r r r r r
NOT	REG memory	Invert each bit of the operand. Algorithm: if bit is 1 turn it to 0. if bit is 0 turn it to 1. C Z S O P A unchanged
OR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical OR between all bits of two operands. Result is stored in first operand. C Z S O P A 0 r r 0 r ?
OUT	DX, AL DX, AX	Output from AL or AX to port. First operand is a port number. If required to access port number over 255 - DX register should be used. C Z S O P A unchanged
ROL	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 left. The number of rotates is set by operand2. Algorithm: shift all bits left, the bit that goes off is set to CF and the same bit is inserted to the right-most position.

		 OF=0 if first operand keeps original sign.
ROR	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 right. The number of rotates is set by operand2. Algorithm: shift all bits right, the bit that goes off is set to CF and the same bit is inserted to the left-most position.  OF=0 if first operand keeps original sign.
RCL	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 left through Carry Flag. The number of rotates is set by operand2. When immediate is greater then 1, assembler generates several RCL xx, 1 instructions because 8086 has machine code only for this instruction (the same principle works for all other shift/rotate instructions). Algorithm: shift all bits left, the bit that goes off is set to CF and previous value of CF is inserted to the right-most position.  OF=0 if first operand keeps original sign.
RCR	memory, immediate REG, immediate memory, CL REG, CL	Rotate operand1 right through Carry Flag. The number of rotates is set by operand2. Algorithm: shift all bits right, the bit that goes off is set to CF and previous value of CF is inserted to the left-most position.  OF=0 if first operand keeps original sign.
SHL	memory, immediate REG, immediate memory, CL REG, CL	Shift operand1 Left. The number of shifts is set by operand2. Algorithm: Shift all bits left, the bit that goes off is set to CF. Zero bit is inserted to the right-most position.  OF=0 if first operand keeps original sign.
SHR	memory, immediate REG, immediate memory, CL REG, CL	Shift operand1 Right. The number of shifts is set by operand2. Algorithm: Shift all bits right, the bit that goes off is set to CF. Zero bit is inserted to the left-most position.

		C O r r OF=0 if first operand keeps original sign.
SUB	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Subtract. Algorithm: operand1 = operand1 - operand2 C Z S O P A r r r r r r
SBB	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Subtract with Borrow. Algorithm: operand1 = operand1 - operand2 - CF Example: C Z S O P A r r r r r r
XLATB	No operands	Translate byte from table. Copy value of memory byte at DS:[BX + unsigned AL] to AL register. Algorithm: AL = DS:[BX + unsigned AL] C Z S O P A unchanged
XOR	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand. C Z S O P A 0 r r 0 r ?

8086 Instruction Set Summary

The following is a brief summary of the 8086 instruction set:

Data Transfer Instructions

MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	Load effective address
LDS, LES	Load pointer using data segment, extra segment
PUSH, POP	Push word onto stack, pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

Logical Instructions

NOT	Logical NOT of byte or word (one's complement)
AND	Logical AND of byte or word
OR	Logical OR of byte or word
XOR	Logical exclusive-OR of byte or word
TEST	Test byte or word (AND without storing)

Shift and Rotate Instructions

SHL, SHR	Logical shift left, right byte or word by 1 or CL
SAL, SAR	Arithmetic shift left, right byte or word by 1 or CL
ROL, ROR	Rotate left, right byte or word by 1 or CL
RCL, RCR	Rotate left, right through carry byte or word by 1 or CL

Arithmetic Instructions

ADD, SUB	Add, subtract byte or word
ADC, SBB	Add, subtract byte or word and carry (borrow)
INC, DEC	Increment, decrement byte or word
NEG	Negate byte or word (two's complement)
CMP	Compare byte or word (subtract without storing)
MUL, DIV	Multiply, divide byte or word (unsigned)
IMUL, IDIV	Integer multiply, divide byte or word (signed)
CBW, CWD	Convert byte to word, word to double word (useful before multiply/divide)
AAA, AAS, AAM, AAD	ASCII adjust for addition, subtraction, multiplication, division (ASCII codes 30-39)
DAA, DAS	Decimal adjust for addition, subtraction (binary coded decimal numbers)

Transfer Instructions

JMP	Unconditional jump
JA (JNBE)	Jump if above (not below or equal)
JAE (JNB)	Jump if above or equal (not below)
JB (JNAE)	Jump if below (not above or equal)
JBE (JNA)	Jump if below or equal (not above)
JE (JZ)	Jump if equal (zero)
JG (JNLE)	Jump if greater (not less or equal)
JGE (JNL)	Jump if greater or equal (not less)

8259 Programmable Interrupt Controller (PIC)

ICW1

7	6	5	4	3	2	1	0
0	0	0	1	LTIM	0	SNGL	IC4

LTIM: 0=Edge Triggering 1=Level Triggering
SNGL: 0=Cascaded PICs 1=Master Only
IC4: 0=No ICW4 1=ICW4 Required

ICW2

7	6	5	4	3	2	1	0
Off7	Off6	Off5	Off4	Off3	0	0	0

Off7..Off3: Programmable Interrupt Vector Offset

ICW3 (Master)

7	6	5	4	3	2	1	0
S7	S6	S5	S4	S3	S2	S1	S0

S7..S0: 0=IR Line is Connected to Peripheral Device or is Free
1=IR Line is Connected to Slave PIC

ICW4

7	6	5	4	3	2	1	0
0	0	0	SFNM	BUF	M/S	AEOI	µPM

SFNM: 0=No Special Fully Nested Mode 1=Special Fully Nested Mode
BUF: 0=No Buffered Mode 1=Buffered Mode
M/S: 0=Slave PIC 1=Master PIC
AEOI: 0=Manual EOI 1=Automatic EOI
µPM: 0=MCS-80/85 Mode 1=8086/88 Mode

OCW1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M7	M6	M5	M4	M3	M2	M1	M0

INTERRUPT MASK
1 = MASK SET
0 = MASK RESET

8253/54 Programmable Interval Timer (PIT)

Kontrol kelimesinin formatı

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RW_1	RW_0	M_2	M_1	M_0	BCD

SC - Select counter

SC_1	SC_0	
0	0	Select counter 0
0	1	Select counter 1
1	0	Select counter 2
1	1	Illegal for 8253 Read -Back command for 8254 (See Read operations)

M - Mode

M_2	M_1	M_0	
0	0	0	Mode 0
0	0	1	Mode 1
x	1	0	Mode 2
x	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW - Read /Write

RW_1	RW_0	
0	0	Counter latch command (See Read operations)
0	1	Read / Write least significant byte only
1	0	Read / Write most significant byte only
1	1	Read / write least significant byte first, then most significant byte

BCD :

0	Binary counter 16 - bits
1	Binary coded decimal (BCD) Counter (4 Decades)