

## 86ASM.FTH Programmer's Manual

First Edition, July 2021  
by Lubomir Rintel

The 86ASM.FTH is a FORTH package providing an assembler for the Intel 8086 processor. It provides words corresponding to the human readable mnemonics of the instructions and operands that append corresponding machine code to the dictionary.

This document describes how to use the package. It is not a complete description of how to write assembly programs for Intel 8086, or a reference manual. Instead, it focuses on illustrating the specifics of a FORTH-based assembler with examples that assume familiarity with using other Intel 8086 assemblers.

It is highly recommended to supplement this document with a Intel 8086 Datasheet,<sup>1</sup> User's Manual<sup>2</sup> or an equivalent document (any good PC Assembly book will do).

### 1. Instruction format

As is typical for a FORTH assembler, assembling an instruction starts with words that push operands onto stack, followed by a word that takes them off stack and places them in a correct format with an opcode into the dictionary.

The instructions consume zero to two operands. In case of multiple operands, the destination or result of an operation comes second, at the bottom of the stack.

The general form is this:

Operand 1 (Source)	Operand 2 (Target)	Modifier	Instruction
34 \$	[SI]	SHORT	MOV
%AX	%BX		XCHG
%ES			PUSH
			HALT

---

<sup>1</sup> 8086 16-bit HMOS Microprocessor 8086 8086-2 8086-1, Intel Corporation, 1990

<sup>2</sup> 8086/8088 User's Manual — Programmer's and Hardware Reference, Intel Corporation, 1989, ISBN 1555120814

## 2. Operands

The operands specify the value, type and width of data used by the instruction. They are optionally preceded by a displacement or an immediate value and, also optionally, followed by a width modifier.

These are the supported operands and modifiers to them:

Operand Type	Arguments	Words
16-bit register		%AX %CX %DX %BX %SP %BP %SI %DI
8-bit register		%AL %CL %DL %BL %AH %CH %DH %BH
Segment register		%ES %CS %SS %DS
Memory with displacement	<i>displacement</i>	+ [BX+SI] + [BX+DI] + [BP+SI] + [BP+DI] + [SI] + [DI] + [BP] + [BX]
Memory indexed		[BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] [BP] [BX]
Memory pointer	<i>address</i>	PTR
Immediate value	<i>number</i>	\$
Segment value	<i>number</i>	SEG
16-bit modifier	<i>oper and</i>	LONG
8-bit modifier	<i>oper and</i>	SHORT

The width information is sometimes implicit — the AX register always carries 16-bit values, BL carries 8-bit values, immediate value of 0x1234 is known to be 16-bit. In other cases, such as an immediate value of 0x12 or memory pointer, their width is unspecified. In that case it can be either inferred from the another operand or forced with modifiers. If it is still unknown, it typically defaults to 16-bit wide.

Some examples to illustrate how are the operands specified:

%AX	12 + [BP]	MOV	Move contents of AX to address 12 + BP 16-bit inferred from the first operand
34 \$	[SI]	MOV	Move value 0x0012 into 16 bits at DI 16-bit default
34 \$	[SI]	SHORT MOV	Move value 0x12 into 8 bits at DI 8-bit forced by the modifier
1234	PTR	PUSH	Push 16-bit value at address 0x1234 onto stack 16-bit forced by the instruction
		HALT	Halt execution until an interrupt comes No operands

### 3. Instructions

This section presents examples of use for all supported instructions along with the machine code they generate. They are roughly in the same order as used by the Intel 8086 Processor Datasheet and grouped together where it makes sense.

The mnemonics used by the GNU Binutils' assembler and disassembler are present for convenient cross-referencing with documentation for an arguably more popular assembler.

Some mnemonics are different from what is used by Intel in order to avoid collisions with words in various FORTH implementations. This makes the assembler usable with FORTH implementations without vocabularies. Adding alias words is trivial in case user wishes to use more familiar mnemonics.

All numbers below are in base 16 (hex).

#### 3.1. MOV

Various ways to move data between registers and memory.

Assembled	Source	Target	Instruction	GNU Assembler
c7 05 34 12	1234 \$	[DI]	MO V	movw \$0x1234,(%di)
c6 04 34	34 \$	[SI]	SHOR T MOV	movb \$0x34,(%si)
c7 40 05 34 12	1234 \$	5+[BX+SI]	MO V	movw \$0x1234,0x5(%bx,%si)
89 0d	%CX	[DI]	MO V	mov %cx,(%di)
8b 0d	[DI]	%CX	MO V	mov (%di),%cx
89 d9	%BX	%CX	MO V	mov %bx,%cx
89 cb	%CX	%BX	MO V	mov %cx,%bx
89 0e 12 00	%CX	12 PTR	MO V	mov %cx,0x12
8b 0e 12 00	12 PTR	%CX	MOV	mov 0x12,%cx
c6 40 05 12	12 \$	5+[BX+SI]	SHOR T MOV	movb \$0x12,0x5(%bx,%si)
c7 04 34 00	34 \$	[SI]	LONG MO V	movw \$0x34,(%si)
c7 05 34 12	1234 \$	[DI]	LONG MO V	movw \$0x1234,(%di)
c7 40 05 34 12	1234 \$	5+[BX+SI]	LONG MO V	movw \$0x1234,0x5(%bx,%si)
c7 04 34 00	34 \$	[SI]	MO V	movw \$0x34,(%si)
88 0d	%CL	[DI]	MO V	mov %cl,(%di)
8a 0d	[DI]	%CL	MO V	mov (%di),%cl
88 d9	%BL	%CL	MO V	mov %bl,%cl
88 cb	%CL	%BL	MO V	mov %cl,%bl
88 0e 12 00	%CL	12 PTR	MO V	mov %cl,0x12
8a 0e 12 00	12 PTR	%CL	MOV	mov 0x12,%cl
a3 34 12	1234 PTR	%AX	MO V	mov %ax,0x1234
a1 34 12	%AX	1234 PTR	MO V	mov 0x1234,%ax
a2 34 12	1234 PTR	%AL	MO V	mov %al,0x1234
a0 34 12	%AL	1234 PTR	MO V	mov 0x1234,%al
bb 34 12	1234 \$	%BX	MO V	mov \$0x1234,%bx
b3 12	12 \$	%BL	MO V	mov \$0x12,%bl

### 3.2. PUSH, POP

Move 16-bit values to and from the top of the stack.

Assembled	Operand	Instruction	GNU Assembler
53	%BX	PUSH	push %bx
1e	%DS	PUSH	push %ds
ff b5 34 12	1234 +[DI]	PUSH	pushw 0x1234(%di)
ff 30	[BX+SI]	PUSH	pushw (%bx,%si)
ff 36 34 12	1234 PTR	PUSH	pushw 0x1234
ff 75 12	12 +[DI]	PUSH	pushw 0x12(%di)
ff 35	[DI]	PUSH	pushw (%di)
ff 76 00	[BP]	PUSH	pushw 0x0(%bp)
5b	%BX	POP	pop %bx
1f	%DS	POP	pop %ds
8f 85 34 12	1234 +[DI]	POP	popw 0x1234(%di)
8f 00	[BX+SI]	POP	popw (%bx,%si)
8f 06 34 12	1234 PTR	POP	popw 0x1234
8f 45 12	12 +[DI]	POP	popw 0x12(%di)
8f 05	[DI]	POP	popw (%di)
8f 46 00	[BP]	POP	popw 0x0(%bp)

### 3.3. XCHG, NOP

Exchange data between various registers and memory locations. Analogous to MOV, except for that immediate values don't make sense.

There's no real no-operation instruction on an 8086 — the NOP word is merely an alias to XCHG that has no effect.

Assembled	Operands	Instruction	GNU Assembler
87 d0	%DX %AX	XCHG	xchg %dx,%ax
92	%AX %DX	XCHG	xchg %ax,%dx
87 db	%BX %BX	XCHG	xchg %bx,%bx
87 9d 34 12	1234 +[DI] %BX	XCHG	xchg %bx,0x1234(%di)
87 18	[BX+SI] %BX	XCHG	xchg %bx,(%bx,%si)
87 1e 34 12	1234 PTR %BX	XCHG	xchg %bx,0x1234
87 1d	[DI] %BX	XCHG	xchg %bx,(%di)
87 5d 12	12 +[DI] %BX	XCHG	xchg %bx,0x12(%di)
87 5e 00	[BP] %BX	XCHG	xchg %bx,0x0(%bp)
87 9d 34 12	%BX 1234 +[DI]	XCHG	xchg %bx,0x1234(%di)
87 18	%BX [BX+SI]	XCHG	xchg %bx,(%bx,%si)
87 1d	%BX [DI]	XCHG	xchg %bx,(%di)
87 5d 12	%BX 12 +[DI]	XCHG	xchg %bx,0x12(%di)
87 5e 00	%BX [BP]	XCHG	xchg %bx,0x0(%bp)
87 1e 34 12	%BX 1234 PTR	XCHG	xchg %bx,0x1234
90	%AX %AX	XCHG	nop
90		NOP	nop

### 3.4. IN, OUT

Move data from and to the I/O space. Always uses the accumulator register — the choice between AX and AL determines the width of the data transfer.

Assembled	Source	Target	Instruction	GNU Assembler
e5 30	30 \$	%AX	IN	in \$0x30,%ax
ed	%DX	%AX	IN	in (%dx),%ax
e4 30	30 \$	%AL	IN	in \$0x30,%al
ec	%DX	%AL	IN	in (%dx),%al
e7 30	%AX	30 \$	OUT	out %ax,\$0x30
ef	%AX	%DX	OUT	out %ax,(%dx)
e6 30	%AL	30 \$	OUT	out %al,\$0x30
ee	%AL	%DX	OUT	out %al,(%dx)

### 3.5. XLAT

Assembled	Instruction	GNU Assembler
d7	XLAT	xlat %ds:(%bx)

### 3.6. LEA, LDS, LES

Assembled	Source	Target	Instruction	GNU Assembler
8d 0e 34 12	1234 PTR	%CX	LEA	lea 0x1234,%cx
8d 05	[DI]	%AX	LEA	lea (%di),%ax
c5 05	[DI]	%AX	LDS	lds (%di),%ax
c4 0e 34 12	1234 PTR	%CX	LES	les 0x1234,%cx

### 3.7. LAHF, SAHF, PUSHF, POPF

Assembled	Instruction	GNU Assembler
9f	LAHF	lahf
9e	SAHF	sahf
9c	PUSHF	pushf
9d	POPF	popf

### 3.8. ADD, ADC, SUB, SSB

Addition and subtraction, optionally with a carry or borrow.

Assembled	Source	Target	Instruction	GNU Assembler
05 12 00	12 \$	%AX	ADD	add \$0x12,%ax
05 34 12	1234 \$	%AX	ADD	add \$0x1234,%ax
04 12	12 \$	%AL	ADD	add \$0x12,%al
81 c3 12 00	12 \$	%BX	ADD	add \$0x12,%bx
81 c3 34 12	1234 \$	%BX	ADD	add \$0x1234,%bx
80 c3 12	12 \$	%BL	ADD	add \$0x12,%bl
01 cb	%CX	%BX	ADD	add %cx,%bx
15 12 00	12 \$	%AX	ADC	adc \$0x12,%ax
15 34 12	1234 \$	%AX	ADC	adc \$0x1234,%ax
14 12	12 \$	%AL	ADC	adc \$0x12,%al
81 d3 12 00	12 \$	%BX	ADC	adc \$0x12,%bx
81 d3 34 12	1234 \$	%BX	ADC	adc \$0x1234,%bx
80 d3 12	12 \$	%BL	ADC	adc \$0x12,%bl
11 cb	%CX	%BX	ADC	adc %cx,%bx
2d 12 00	12 \$	%AX	SUB	sub \$0x12,%ax
2d 34 12	1234 \$	%AX	SUB	sub \$0x1234,%ax
2c 12	12 \$	%AL	SUB	sub \$0x12,%al
81 eb 12 00	12 \$	%BX	SUB	sub \$0x12,%bx
81 eb 34 12	1234 \$	%BX	SUB	sub \$0x1234,%bx
80 eb 12	12 \$	%BL	SUB	sub \$0x12,%bl
29 cb	%CX	%BX	SUB	sub %cx,%bx
1d 12 00	12 \$	%AX	SSB	sbb \$0x12,%ax
1d 34 12	1234 \$	%AX	SSB	sbb \$0x1234,%ax
1c 12	12 \$	%AL	SSB	sbb \$0x12,%al
81 db 12 00	12 \$	%BX	SSB	sbb \$0x12,%bx
81 db 34 12	1234 \$	%BX	SSB	sbb \$0x1234,%bx
80 db 12	12 \$	%BL	SSB	sbb \$0x12,%bl
19 cb	%CX	%BX	SSB	sbb %cx,%bx

### 3.9. MUL, IMUL, DIV, IDIV

Signed or unsigned multiplication and division.

Assembled	Operand	Instruction	GNU Assembler
f7 e3	%BX	MUL	mul %bx
f6 e1	%CL	MUL	mul %cl
f7 25	[DI]	MUL	mulw (%di)
f6 a5 34 12	1234+[DI]	SHOR T MUL	mulb 0x1234(%di)
f7 eb	%BX	IMUL	imul %bx
f6 e9	%CL	IMUL	imul %cl
f7 2d	[DI]	IMUL	imulw (%di)
f6 ad 34 12	1234+[DI]	SHOR T IMUL	imulb 0x1234(%di)
f7 f3	%BX	DIV	div %bx
f6 f1	%CL	DIV	div %cl
f7 35	[DI]	DIV	divw (%di)
f6 b5 34 12	1234+[DI]	SHOR T DIV	divb 0x1234(%di)
f7 fb	%BX	IDIV	idiv %bx
f6 f9	%CL	IDIV	idiv %cl
f7 3d	[DI]	IDIV	idivw (%di)
f6 bd 34 12	1234+[DI]	SHOR T IDIV	idivb 0x1234(%di)

### 3.10. INC, DEC, NEG, BNOT

Instructions that make simple changes to the single operand they take.

BNOT mnemonic is used instead of NOT to avoid collision with CALL word in Open Firmware.

Assembled	Operand	Instruction	GNU Assembler
43	%BX	INC	inc %bx
fe c1	%CL	INC	inc %cl
ff 05	[DI]	INC	incw (%di)
fe 85 34 12	1234+[DI]	SHOR T INC	incb 0x1234(%di)
fe 00	[BX+SI]	SHORT INC	incb (%bx,%si)
4b	%BX	DEC	dec %bx
fe cb	%BL	DEC	dec %bl
ff 0d	[DI]	DEC	decw (%di)
fe 0d	[DI]	SHORT DEC	decb (%di)
f7 db	%BX	NEG	neg %bx
f6 db	%BL	NEG	neg %bl
f7 1d	[DI]	NEG	negw (%di)
f6 1d	[DI]	SHORT NEG	negb (%di)
f7 15	[DI]	BNOT	notw (%di)
f6 d3	%BL	BNO T	not %bl
f7 d3	%BX	BNO T	not %bx
f6 d1	%CL	BNO T	not %cl
f6 95 34 12	1234+[DI]	SHOR T BNOT	notb 0x1234(%di)
f6 10	[BX+SI]	SHORT BNOT	notb (%bx,%si)

**3.11. AAA, BAA, AAS, DAS, AAM, AAD**

Various Binary Coded Decimal handling utility instructions.

Assembled	Instruction	GNU Assembler
37	AAA	aaa
27	BAA	daa
3f	AAS	aas
2f	DAS	das
d4 0a	AAM	aam \$0xa
d5 0a	AAD	aad \$0xa

**3.12. CMP**

Compare, set bits in FLAGS register.

Assembled	Source	Target	Instruction	GNU Assembler
3d 12 00	12 \$	%AX	CMP	cmp \$0x12,%ax
3d 34 12	1234 \$	%AX	CMP	cmp \$0x1234,%ax
3c 12	12 \$	%AL	CMP	cmp \$0x12,%al
81 fb 12 00	12 \$	%BX	CMP	cmp \$0x12,%bx
81 fb 34 12	1234 \$	%BX	CMP	cmp \$0x1234,%bx
80 fb 12	12 \$	%BL	CMP	cmp \$0x12,%bl
39 cb	%CX	%BX	CMP	cmp %cx,%bx

**3.13. CBW, CWD**

Assembled	Instruction	GNU Assembler
98	CBW	cbtw
99	CWD	cwtd



**3.14. SHL, SAL, SHR, SAR, ROL, ROR, RCL, RCR**

Various shifts and rotations. Arithmetic and logical alike.

Assembled	Count	Target	Instruction	GNU Assembler
d1 e3	1 \$	%BX	SHL	shl %bx
d0 e3	1 \$	%BL	SHL	shl %bl
d3 60 05	%CL	5 +[BX+SI]	SHL	shlw %cl,0x5(%bx,%si)
d0 60 05	1 \$	5 +[BX+SI]	SHOR T SHL	shlb 0x5(%bx,%si)
d1 eb	1 \$	%BX	SHR	shr %bx
d0 eb	1 \$	%BL	SHR	shr %bl
d3 68 05	%CL	5 +[BX+SI]	SHR	shrw %cl,0x5(%bx,%si)
d0 68 05	1 \$	5 +[BX+SI]	SHOR T SHR	shrb 0x5(%bx,%si)
d1 fb	1 \$	%BX	SAR	sar %bx
d0 fb	1 \$	%BL	SAR	sar %bl
d3 78 05	%CL	5 +[BX+SI]	SAR	sarw %cl,0x5(%bx,%si)
d0 78 05	1 \$	5 +[BX+SI]	SHOR T SAR	sarb 0x5(%bx,%si)
d1 c3	1 \$	%BX	R OL	rol %bx
d0 c3	1 \$	%BL	R OL	rol %bl
d3 40 05	%CL	5 +[BX+SI]	ROL	rolw %cl,0x5(%bx,%si)
d0 40 05	1 \$	5 +[BX+SI]	SHOR T ROL	rolb 0x5(%bx,%si)
d1 cb	1 \$	%BX	R OR	ror %bx
d0 cb	1 \$	%BL	R OR	ror %bl
d3 48 05	%CL	5 +[BX+SI]	ROR	rorw %cl,0x5(%bx,%si)
d0 48 05	1 \$	5 +[BX+SI]	SHOR T ROR	rorb 0x5(%bx,%si)
d1 d3	1 \$	%BX	RCL	rcl %bx
d0 d3	1 \$	%BL	RCL	rcl %bl
d3 50 05	%CL	5 +[BX+SI]	RCL	rclw %cl,0x5(%bx,%si)
d0 50 05	1 \$	5 +[BX+SI]	SHOR T RCL	rclb 0x5(%bx,%si)
d1 db	1 \$	%BX	RCR	rcr %bx
d0 db	1 \$	%BL	RCR	rcr %bl
d3 58 05	%CL	5 +[BX+SI]	RCR	rcrw %cl,0x5(%bx,%si)
d0 58 05	1 \$	5 +[BX+SI]	SHOR T RCR	rcrb 0x5(%bx,%si)

### 3.15. BAND, TEST, BOR, BXOR

Various logical functions. TEST only modifies the FLAGS, not the Target operand.

BAND, BOR and BXOR mnemonics are used instead of AND, OR and XOR to avoid collision with standard words.

Assembled	Operand	Target	Instruction	GNU Assembler
25 12 00	12 \$	%AX	B AND	and \$0x12,%ax
25 34 12	1234 \$	%AX	B AND	and \$0x1234,%ax
24 12	12 \$	%AL	B AND	and \$0x12,%al
81 e3 12 00	12 \$	%BX	B AND	and \$0x12,%bx
81 e3 34 12	1234 \$	%BX	B AND	and \$0x1234,%bx
80 e3 12	12 \$	%BL	B AND	and \$0x12,%bl
21 cb	%CX	%BX	B AND	and %cx,%bx
a9 12 00	12 \$	%AX	TEST	test \$0x12,%ax
a9 34 12	1234 \$	%AX	TEST	test \$0x1234,%ax
a8 12	12 \$	%AL	TEST	test \$0x12,%al
f7 c3 12 00	12 \$	%BX	TEST	test \$0x12,%bx
f7 c3 34 12	1234 \$	%BX	TEST	test \$0x1234,%bx
f6 c3 12	12 \$	%BL	TEST	test \$0x12,%bl
85 cb	%CX	%BX	TEST	test %cx,%bx
0d 12 00	12 \$	%AX	BOR	or \$0x12,%ax
0d 34 12	1234 \$	%AX	BOR	or \$0x1234,%ax
0c 12	12 \$	%AL	BOR	or \$0x12,%al
81 cb 12 00	12 \$	%BX	BOR	or \$0x12,%bx
81 cb 34 12	1234 \$	%BX	BOR	or \$0x1234,%bx
80 cb 12	12 \$	%BL	BOR	or \$0x12,%bl
09 cb	%CX	%BX	BOR	or %cx,%bx
35 12 00	12 \$	%AX	BXOR	xor \$0x12,%ax
35 34 12	1234 \$	%AX	BXOR	xor \$0x1234,%ax
34 12	12 \$	%AL	BXOR	xor \$0x12,%al
81 f3 12 00	12 \$	%BX	BXOR	xor \$0x12,%bx
81 f3 34 12	1234 \$	%BX	BXOR	xor \$0x1234,%bx
80 f3 12	12 \$	%BL	BXOR	xor \$0x12,%bl
31 cb	%CX	%BX	BXOR	xor %cx,%bx

**3.16. REPNZ:, REPZ:**

Instruction prefix that conditionally repeat the following instruction. Typically used with data movement instructions for string operations.

Assembled	Prefix	Instruction	GNU Assembler
f2 90	REPNZ:	NOP	repnz nop
f3 90	REPZ:	NOP	pause

**3.17. MOVSB, MOVSW, CMPSB, CMPSW, SCASB, SCASW, LODSB, LODSW, STOSB, STOSW**

Data movement. Often used with the REPNZ: and REPZ: prefixes.

Assembled	Instruction	GNU Assembler
a4	MOVSB	movsb %ds:(%si),%es:(%di)
a5	MOVSW	movsw %ds:(%si),%es:(%di)
a6	CMPSB	cmpsb %es:(%di),%ds:(%si)
a7	CMPSW	cmpsw %es:(%di),%ds:(%si)
ae	SCASB	scas %es:(%di),%al
af	SCASW	scas %es:(%di),%ax
ac	LODSB	lods %ds:(%si),%al
ad	LODSW	lods %ds:(%si),%ax
aa	STOSB	stos %al,%es:(%di)
ab	STOSW	stos %ax,%es:(%di)

**3.18. CAL, JMP**

Unconditional branch or function entry.

CAL mnemonic is used instead of CALL to avoid collision with CALL word in GForth.

Assembled	Source	Target	Instruction	GNU Assembler
ff 50 05		5+[BX+SI]	CAL	call *0x5(%bx,%si)
ff 50 05		5+[BX+SI]	SHOR T CAL	call *0x5(%bx,%si)
ff 58 05		5+[BX+SI]	LONG CAL	lcall *0x5(%bx,%si)
ff 17		[BX]	CAL	call *(%bx)
ff 16 12 00		12 PTR	CAL	call *0x12
ff 16 34 12		1234 PTR	CAL	call *0x1234
ff 16 12 00		12 PTR	SHOR T CAL	call *0x12
ff 16 34 12		1234 PTR	SHOR T CAL	call *0x1234
ff 1e 34 12		1234 PTR	LONG CAL	lcall *0x1234
e8 34 12		1234 \$	CAL	call 0x1237
9a 34 12 78 56	1234 \$	5678 SEG	CAL	lcall \$0x5678,\$0x1234
9a 34 12 78 56	1234 \$	5678 SEG	LONG CAL	lcall \$0x5678,\$0x1234
ff 60 05		5+[BX+SI]	JMP	jmp *0x5(%bx,%si)
ff 60 05		5+[BX+SI]	SHOR T JMP	jmp *0x5(%bx,%si)
ff 68 05		5+[BX+SI]	LONG JMP	ljmp *0x5(%bx,%si)
ff 27		[BX]	JMP	jmp *(%bx)
ff 26 12 00		12 PTR	JMP	jmp *0x12
ff 26 34 12		1234 PTR	JMP	jmp *0x1234
ff 26 12 00		12 PTR	SHOR T JMP	jmp *0x12
ff 26 34 12		1234 PTR	SHOR T JMP	jmp *0x1234
ff 2e 34 12		1234 PTR	LONG JMP	ljmp *0x1234
eb 12		12 \$	JMP	jmp 0x14
e9 34 12		1234 \$	JMP	jmp 0x1237
ea 34 12 78 56	1234 \$	5678 SEG	JMP	ljmp \$0x5678,\$0x1234
ea 34 12 78 56	1234 \$	5678 SEG	LONG JMP	ljmp \$0x5678,\$0x1234

**3.19. RET, RETF**

Function return, near (just the 16-bit offset) or far (segment followed by offset, both 16-bit wide).

Assembled	Operand	Instruction	GNU Assembler
c2 34 12	1234	+RET	ret \$0x1234
c3		RET	ret
ca 34 12	1234	+RETF	lret \$0x1234
cb		RETF	lret

**3.20. JE, JL, JLE, JNAE, JBE, JP, JO, JS, JNE, JNL, JNLE, JNB, JNBE, JNP, JNO, JNS**

Conditional branches. JZ and JB mnemonics would collide with words defined in GForth. Use JE and JNAE instead.

Assembled	Operand	Instruction	GNU Assembler
74 12	12 \$	JE	je 0x14
7c 12	12 \$	JL	jl 0x14
7c 12	12 \$	JNGE	jl 0x14
7e 12	12 \$	JLE	jle 0x14
7e 12	12 \$	JNG	jle 0x14
72 12	12 \$	JNAE	jb 0x14
76 12	12 \$	JBE	jbe 0x14
76 12	12 \$	JNA	jbe 0x14
7a 12	12 \$	JP	jp 0x14
7a 12	12 \$	JPE	jp 0x14
70 12	12 \$	JO	jo 0x14
78 12	12 \$	JS	js 0x14
75 12	12 \$	JNE	jne 0x14
75 12	12 \$	JNZ	jne 0x14
7d 12	12 \$	JNL	jge 0x14
7d 12	12 \$	JGE	jge 0x14
7f 12	12 \$	JNLE	jg 0x14
7f 12	12 \$	JG	jg 0x14
73 12	12 \$	JNB	jae 0x14
73 12	12 \$	JAE	jae 0x14
77 12	12 \$	JNBE	ja 0x14
77 12	12 \$	JA	ja 0x14
7b 12	12 \$	JNP	jnp 0x14
7b 12	12 \$	JPO	jnp 0x14
71 12	12 \$	JNO	jno 0x14
79 12	12 \$	JNS	jns 0x14

**3.21. LOOPI, LOOPZ, LOOPNZ, JCXZ**

Branches conditional on CX value; possibly modifying it along the way. LOOPI mnemonic is used instead of LOOP to avoid collision with a standard word.

Assembled	Operand	Instruction	GNU Assembler
e2 12	12 \$	LOOPI	loop 0x14
e1 12	12 \$	LOOPZ	loope 0x14
e1 12	12 \$	LOOPE	loope 0x14
e0 12	12 \$	LOOPNZ	loopne 0x14
e0 12	12 \$	LOOPNE	loopne 0x14
e3 12	12 \$	JCXZ	jcxz 0x14

**3.22. INT, INTO**

Invoke software interrupts.

Assembled	Operand	Instruction	GNU Assembler
cd 05	5	INT	int \$0x5
cc	3	INT	int3
ce		INTO	into

**3.23. IRET**

Return from interrupt.

Assembled	Instruction	GNU Assembler
cf	IRET	iret

**3.24. CLC, CMC, STC, CLD, STD, CLI, STI**

Modify various bits in FLAGS.

Assembled	Instruction	GNU Assembler
f8	CLC	clc
f5	CMC	cmc
f9	STC	stc
fc	CLD	cld
fd	STD	std
fa	CLI	cli
fb	STI	sti

**3.25. HLT, WAIT**

Suspend execution.

Assembled	Instruction	GNU Assembler
f4	HLT	hlt
9b	WAIT	fwait

**3.26. ESC**

Call into coprocessor, such as 8087 or 8089.

It is really not advisable to use this word directly. To make reasonable use of a coprocessor, you'd define the words for actual coprocessor operations using ESC.

Assembled	Operand	Instruction	GNU Assembler
d9 fa	%DX	0F ESC	fsqrt

**3.27. LOCK:**

Instruction prefix that locks the bus during the duration of the following instruction.

Assembled	Prefix	Instruction	GNU Assembler
f0 90	LOCK:	NOP	lock nop

**3.28. SEG:**

Instruction prefix that makes the following instruction use a non-default segment register.

Assembled	Operand	Prefix	Instruction	GNU Assembler
26 90	%ES	SEG:	NOP	es nop

Copyright © 2021 Lubomir Rintel <lkundrak@v3.sk>

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

A copy of the license is available at <<https://www.gnu.org/licenses/fdl-1.1.txt>>