# CHAPTER 24. 8086 ASSEMBLER

The source code of the assembler is in CPU8086.BLK, screens 3 to 17, and KERNEL86.BLK, screen 80.

The assembler in FORTH allows the user to define words which will be executed at the raw machine speed and make use of all the resource in the host computer hardware system. It is often used to optimize a system or application program by recoding the critical or most often executed words to improve the performance of the program. A high level word can be substituted by a machine code word if the interface to the system, most notably the stack effect is kept the same.

The assembler is invoked by the defining word CODE which creates a header in the dictionary and makes the code field point to the parameter field. Machine instructions can then be compiled into the parameter field by a set of machine code words with mnemonic names similar to those used in regular assembler of the host processor. These machine code words are executed by the text interpreter and the net effect is to add new machine instructions to the parameter field so that when the new CODE word is executed, these machine instructions are executed in sequence. A CODE word must be terminated by the inner interpreter NEXT or its derivative to return control to the calling word, and the code ending word END-CODE or C;, which makes the new word available for execution or compilation.

## 24.1. ASSEMBLY TOOLS

A set of tool words is needed to assemble a machine code word in its most primitive form. If the user knows the host processor well enough, he can hand code a routine and generate a Forth CODE word without using the assembler.

| | | |
|---|---|---|
| VARIABLE AVOC | | A variable holding the old context vocabulary during assembling. |
| : CODE | ( --- ) | The defining word that starts the assembling process to build a machine code word. |
| | CREATE | Create the name field, link field, and code field for a new entry in the dictionary. |
| | HIDE | Smudge the header to hide the new word before it is completed. |
| | HERE DUP 2- ! | Store the pfa into code field. This is required for indirectly threaded code. |
| | CONTEXT @ AVOC ! | Save the old context vocabulary. |
| | ASSEMBLER | Use the ASSEMBLER as the context vocabulary to assemble machine code. |
| | ; | |
| : LABEL | ( --- ) | Mark the start of a subroutine. Return its address when the label is invoked. |
| | CREATE | Create the header. |
| | ASSEMBLER | Select context vocabulary. |
| | ; | |

```
232 CONSTANT DOES-OP          Op-code for CALL instruction used in DOES>.
3 CONSTANT DOES-SIZE          A CALL instruction consumes 3 bytes.

: DOES?       ( ip --- ip+3 f )   Return a true flag with the IP moved over the CALL instruction.
        DUP DOES-SIZE +           IP incremented by DOES-SIZE.
        SWAP C@                   Code at IP.
        DOES-OP =                 True if the opcode is CALL.
        ;

ASSEMBLER DEFINITIONS         All the assembler tool words are to be collected in this vocabulary.

: END-CODE          ( --- )       Terminate a code definition, make it available and also
                                  restore context vocabulary.
        AVOC @ CONTEXT !          Restore the old context vocabulary.
        REVEAL                    Un-smudge the header, making the word available to the text
                                  interpreter.
        ;

: C;                ( --- )
        END-CODE                  Synonym for END-CODE.
        ;
```

The following set of deferred words are used in assembling machine instructions or constructing structures in the code definitions. They are defined as deferred words so that they can be shared by the assembler and the meta-compiler.

```
DEFER C,            ( byte --- )  Assemble a machine code byte to the dictionary.

FORTH ' C, ASSEMBLER IS C,        Vector it to the C, in FORTH vocabulary.

DEFER ,             ( n --- )     Assemble a cell to the dictionary.

FORTH ' ,  ASSEMBLER IS ,         Vector to the , (comma)  in FORTH vocabulary.
```

C, and , are the primitive words used in the Forth assembler. Using them one can even generate code definitions without using an assembler. However, the machine code and operands have to be hand coded and stored into the dictionary by , and C, .

```
DEFER HERE          ( --- addr )  Return a pointer to the top of dictionary, the address of the
                                  next code to be assembled.

FORTH ' HERE ASSEMBLER IS HERE    Allow us to assembler code anywhere in memory.

DEFER ?>MARK                      Set up forward branch with error checking.
DEFER ?>RESOLVE                   Resolve a forward branch with error checking.
DEFER ?<MARK                      Set up a backward branch with error checking.
DEFER ?<RESOLVE                   Resolve a backward branch with error checking.
```

## 24.2. 8086 REGISTER DEFINITIONS

Registers are used as operands, which are inserted into the register field in a machine instruction. They are thus defined as constants. The register constants are defined in the following format to facilitate the building of machine instructions which uses the corresponding register, as shown in Fig. 24.1. The lower byte may become a byte instruction or the second instruction byte to specify addressing mode. The upper byte with the mode field is used only during assembly.

```
OCTAL                       8086 codes are best represented in octal due to the 3 bit fields.

: REG       ( mode reg# --- )   Use the addressing mode and the register number to generate
                                the proper register field to be defined as register constants.
        11 * SWAP               Fill both reg and r/m fields.
        1000 * OR               The addressing mode field.
        CONSTANT                Defined as a constant to be inserted into register accessing code.
        ;

: REGS      ( n mode --- )      Define a set of register words which differ only in the
                                register number.
        SWAP 0 DO               Scan through n registers.
             DUP I REG          Define each register as a constant.
        LOOP  DROP              Discard mode.
        ;
```

Using the powerful REGS, we can define all the registers with all possible addressing modes as distinct Forth words returning the appropriate register constants to be used by machine code assembler words to construct machine instructions.

```
10 0 REGS                   Define operands addressing only bytes in the registers.
        AL CL DL BL AH CH DH BH

10 1 REGS                   Define operands addressing word registers.
        AX CX DX BX SP BP SI DI

10 2 REGS                   Define indexed addressing operands.
        [BX+SI] [BX+DI] [BP+SI] [BP+DI] [SI] [DI] [BP] [BX]

4 2 REGS                    Duplicated definitions.
        [SI+BX] [DI+BX] [SI+BP] [DI+BP]

4 3 REGS                    Segment register addressing operands.
        ES CS SS DS

3 4 REGS                    Immediate addressing operands.
        # #) S#)
```

Four registers are of special interests to the Forth system because they are used as registers in the virtual Forth machine. They are assigned generic names more appropriate in the manipulation of the Forth

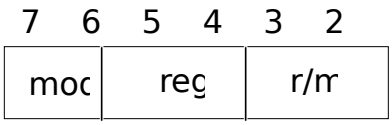
machine:

```
BP      CONSTANT  RP      The return stack pointer.
[BP]    CONSTANT  [RP]    Indirect addressing mode.
SI      CONSTANT  IP      The interpretive pointer.
[SI]    CONSTANT  [IP]
BX      CONSTANT  W       The current word pointer.
[BX]    CONSTANT  [W]
```

The data stack pointer uses the SP register which already has the correct name; therefore, it does not need a new name.

**Figure 24.1 Register addressing mode constant.**

**Addressing Mode Byte**




**Register Modes and Mnemonics**

| | Register | | | | Mode |
|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 |
| 0 | AL | AX | [BX+SI] | ES | # Immediate |
| 1 | CL | CX | [BX+DI] | CS | #) Indirect |
| 2 | DL | DX | [BP+SI] | SS | S#) Inter- |
| 3 | BL | BX | [BP+DI] | DS | segment |
| 4 | AH | SP | [SI] | | |
| 5 | CH | BP | [DI] | | |
| 6 | DH | SI | [BP] | | |
| 7 | BH | DI | [BX] | | |

## 24.3. ADDRESSING MODE OPERATORS

```
: MD                ( mode --- )    Define words which will test various addressing modes.
        CREATE                      Make header.
        1000 * ,                    Compile a template of addressing mode.
        DOES>                       ( mode-field --- f )
        @                           Get the template.
        SWAP 7000 AND               Mask over the mode field.
        = 0<>                       Return true if the mode matches.
        ;

0 MD R8?      ( operand --- f )     Is it in byte register mode?
1 MD R16?     ( operand --- f )     Is it in word register mode?
2 MD MEM?     ( operand --- f )     Is it in memory addressing mode?
3 MD SEG?     ( operand --- f )     Is it in segment addressing mode?
4 MD #?       ( operand --- f )     Is it in immediate addressing mode?

: REG?( operand --- f )             Test for either byte or word addressing mode.
        7000 AND                    Mask the mode field.
        2000 <                      Byte or cell.
        < 0<>                       Return the flag.
        ;

: BIG?        ( n --- f )           Test the size of address offset. Return true if n>255.
        ABS                         Absolute offset.
        -200 AND                    Examine upper byte.
        0<>                         True if not zero.
        ;

: RLOW              ( n1 --- n2 )   Retain only the low r/m field.
        7 AND ;

: RMID              ( n1 --- n2 )   Retain only the middle register field.
        70 AND      ;

VARIABLE SIZE                       A flag.  True for 16 bit and false for 8 bit number.

: BYTE              ( --- )         Reset SIZE to indicate byte operations.
        SIZE OFF    ;

: OP,               ( n opcode --- )  OR the operand and the opcode and assemble the machine code.
        OR C,  ;

: W,                ( opcode operand --- ) Assemble opcode with the W field set if operand indicates a
                                           word register.
        R16? 1 AND                  Set W field according to word mode.
        OP,                         Assemble.
        ;
```

```
: SIZE,         ( opcode --- )      Assemble the opcode with W field determined by SIZE.
        SIZE @ 1 AND                Set W field using SIZE.
        OP,                         Assemble.
        ;

: ,/C,                  ( n f --- ) Assemble a cell if f is true. Otherwise assemble a byte.
        IF ,                        f is true.  Assemble a cell.
        ELSE C,                     f is false.  Assemble a byte.
        THEN ;


: RR,   ( operand1 oprand2 --- )    Assemble a register to register instruction.
        RMID                        Operand1 to r/m field.
        SWAP RLOW                   Operand2 to reg field.
        OR                          Register to register operand.
        300                         Register to register mode.
        OP,                         Assemble the reg-reg second instruction byte for addressing.
        ;

VARIABLE LOGICAL                    True while assembling logical instructions.

: B/L?                  ( n --- f ) BIG? of LOGICAL.
        BIG? LOGICAL @ OR
        ;

: MEM,          ( disp mr rmid --- )    Assemble a memory reference instruction. It takes a
                                        displacement, and memory/ register, and a register as
                                        arguments and encode them into an instruction.
        OVER #) =                       Is it in immediate indirect mode?
        IF                              Yes.
                RMID 6 OP,              Assemble immediate indirect instruction.
                DROP                    No need of mr now.
                ,                       Assemble disp, which is the immediate value.
        ELSE                            Not immediate indirect.
                RMID OVER RLOW OR       OR together the registers.
                -ROT                    Save it.
                [BP] =                  mr=[BP]?
                OVER 0= AND             AND disp=0?
                IF
                SWAP                    Get the register field to top.
                100 OP,                 Byte displacement mode instruction. Mode 1.
                C,                      With the byte displacement.
                ELSE
                SWAP                    Examine disp.
                OVER BIG?               More than 8 bits?
                IF                      Yes.
                        200 OP,         Mode 2 instruction.
```

```
                ,               With cell displacement.
            ELSE
                OVER 0=         Is disp=0?
                IF              Yes.
                    C,          Assemble byte instruction.
                    DROP        No displacement.
                ELSE            All tests failed to reach here.
                    100 OP,     Assemble mode 1 instruction anyway.
                    C,          Append a byte displacement.
                THEN
            THEN
            THEN
        THEN ;

: WMEM,     ( disp mem reg op --- )     Assemble a word memory reference instruction.
        OVER W,                 Pack the word referencing bit into reg and assemble opcode.
        MEM,                    Use MEM, to assemble the right mode instruction.
        ;

: R/M       ( mr reg --- )      Assemble either a register to register or register to memory
                                instruction.
        OVER REG?               Is it in a register mode?
        IF RR,                  Yes.  Assemble a register to register instruction.
        ELSE MEM,               Else assemble a memory referencing instruction.
        THEN ;

: WR/SM     ( rm reg op --- )   Assemble either a register mode instruction with size field,
                                or a memory mode instruction with size from SIZE.
        2 PICK                  Get the mode.
        DUP REG?                Is it register mode?
        IF                      Yes.
            W,                  Squeeze in the word bit.
            RR,                 Assemble a register-register instruction.
        ELSE                    Not register mode.
            DROP                Discard mode.
            SIZE,               Use SIZE for word bit.
            MEM,                Assemble memory instruction.
        THEN
        SIZE ON                 Set default size to 16 bits.
        ;

VARIABLE INTER                  True if doing inter-segment jump, call, or return.

: FAR               ( --- )     Set INTER true.
        FAR ON  ;

: ?FAR              ( n1 --- n2 )   If INTER is true, set the far bit in the instruction.
        INTER @                 If INTER is true,
```

```
IF 10 OR THEN          set bit 3 in the instruction.
INTER OFF              Reset far flag.
;
```

**Figure 24.2 8086 instruction types.**

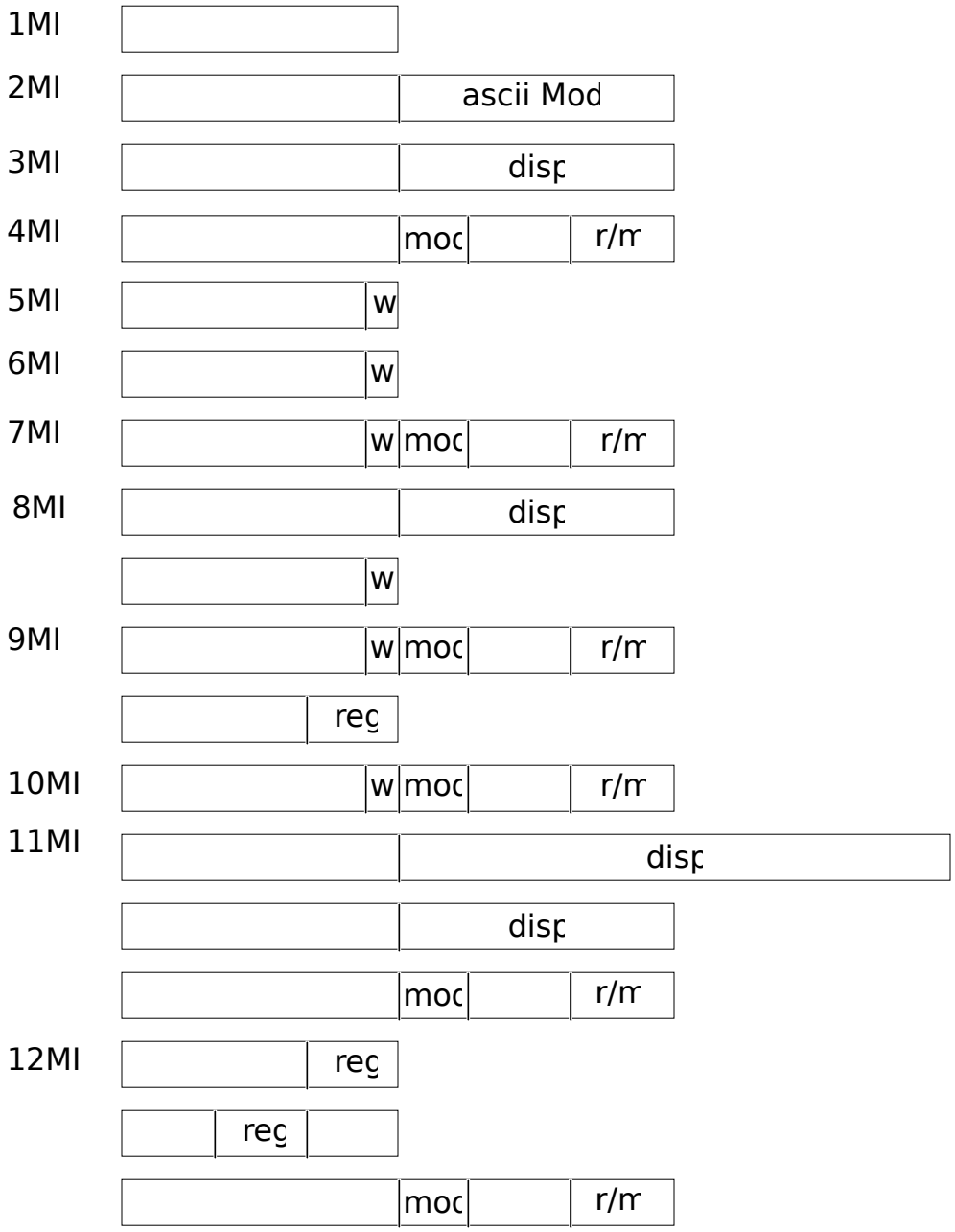
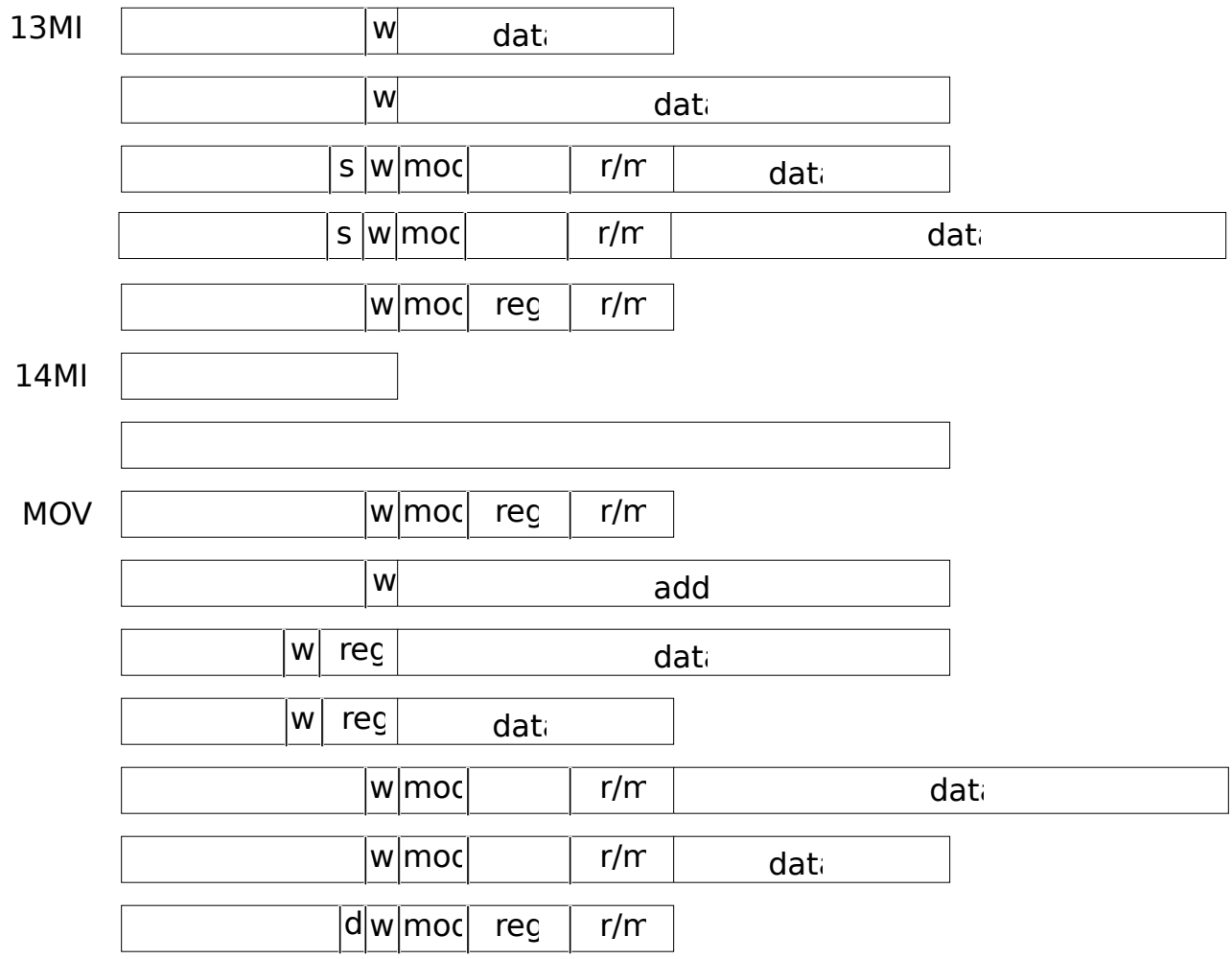


**Figure 24.2 8086 instruction types (continued).**



## 24.4. DEFINING WORDS TO GENERATE OPCODES

8086 is a rather complicated microprocessor. It was designed to be 8080 downward compatible so that it must be able to execute all the 8080 instructions. With lots of 16 bit machine instructions and operators for the extra registers and different mode of operation, the instruction set becomes very involved. Consequently, the assembler also becomes complicated in order to take care of these diverse types of instructions. There are 14 identifiable classes of instructions in 8086. A defining word is used to generate opcodes for each class of instructions. .new .56 Fig. 24.2. 8086 instruction types.

```
: 1MI               ( opcode --- )  Define one byte constant instructions.
        CREATE C,                   Create header and compile the opcode.
        DOES>       ( --- )
        C@                          Fetch opcode from the parameter field of the assembler word.
        C,                          Assemble it.
        ;

HEX
37 1MI AAA      3F 1MI AAS      98 1MI CBW      F8 1MI CLC
FC 1MI CLD      FA 1MI CLI      F5 1MI CMC      99 1MI CWD
27 1MI DAA      2F 1MI DAS      F4 1MI HLT      CE 1MI INTO
CF 1MI IRET     9F 1MI LAHF     F0 1MI LOCK     90 1MI NOP
9D 1MI POPF     9C 1MI PUSHF    F2 1MI REP      F2 1MI REPZ
9E 1MI SAHF     F9 1MI STC      FD 1MI STD      FB 1MI STI
9B 1MI WAIT     D7 1MI XLAT
OCTAL

: 2MI               ( opcode --- )  Define ASCII instructions.
        CREATE C,                   Header and parameter field.
        DOES>       ( --- )
        C@ C,                       Assemble the opcode.
        12 C,                       Assemble the ASCII mode byte.
        ;

HEX   D5 2MI AAD   D4 2MI AAM   OCTAL

: 3MI               ( opcode --- )  Define branch instructions with one byte offset.
        CREATE C,
        DOES>       ( addr --- )
        C@ C,                       Assemble opcode.
        HERE - 1-                   Offset from current address.
        C,                          Assemble the offset.
        ;

HEX
77 3MI JA       73 3MI JAE      72 3MI JB       76 3MI JBE
E3 3MI JCXZ     74 3MI JE       7F 3MI JG       7D 3MI JGE
7C 3MI JL       7E 3MI JLE      75 3MI JNE      71 3MI JNO
79 3MI JNS      70 3MI JO       7A 3MI JPE      7B 3MI JPO
```

```
78 3MI JS               E2 3MI LOOP         E1 3MI LOOPE        E0 3MI LOOPNE
OCTAL

: 4MI                   ( opcode --- )  Define LDS, LEA, LES instructions.
        CREATE C,
        DOES>           ( disp mr rmid --- )
        C@ C,                           Assemble opcode.
        MEM,                            Memory reference.
        ;

HEX   C5 4MI LDS    8D 4MI LEA   C4 4MI LES    OCTAL

: 5MI                   ( opcode --- )  Define string instructions.
        CREATE C,                       Store opcode.
        DOES>           ( --- )
        C@ SIZE,                        Assemble opcode with size bit.
        SIZE ON                         Enable word addressing.
        ;

HEX   A6 5MI CMPS   A4 5MI MOVS   AE 5MI SCAS   OCTAL

: 6MI                   ( opcode --- )  Define string instructions where byte/word mode is
                                        determined at assembly time.
        CREATE C,                       Store opcode.
        DOES>           ( mr --- )
        C@                              Opcode.
        SWAP W,                         Use mr to decide the word bit and assemble accordingly.
        ;

HEX   AD 6MI LODS   AA 6MI STOS   OCTAL

: 7MI                   ( opcode --- )  Define multiply and divide instructions.
        CREATE C,                       Store opcode.
        DOES>           ( r/m --- )
        C@                              The opcode will be put in the reg field of the second byte.
        366                             The real first byte opcode.
        WR/SM,                          Assemble the whole mess.
        ;

HEX   30 7MI DIV   38 7MI IDIV   28 7MI IMUL   20 7MI MUL
        10 7MI NOT   OCTAL

: 8MI                   ( opcode --- )  Define input/output instructions.
        CREATE C,                       Opcode.
        DOES>           ( port --- )
        C@                              Opcode.
        OVER R16?                       Is the port# a 16 bit number?
        1 AND OR                        OR the word bit to opcode.
```

```
        OVER # =                Is there an immediate operator?
        IF                      Yes, a port# is given.
                C,              Assemble opcode.
                C,              Assemble port number.
        ELSE                    Implied port.
                10 OR           Set the implied port bit in opcode.
                C,              Assemble one byte i/o instruction.
        THEN ;

HEX   E4 8MI IN   E6 8MI OUT   OCTAL

: 9MI                ( opcode --- )  Define increment/decrement instructions.
        CREATE C,                    Store opcode.
        DOES>        ( reg --- )
        C@                           Get opcode first.
        OVER R16?                    Mode 1 operation?
        IF                           Yes.
                100 OR               Opcode for one byte inc/dcr instruction.
                SWAP RLOW            Retain only r/m field.
                OP,                  Assemble one byte instruction.
        ELSE                         Other modes.
                376                  First byte opcode.
                WR/SM,               Use stored opcode as second byte instruction.
        THEN ;

HEX   08 9MI DEC   00 9MI INC   OCTAL

: 10MI          (opcode --- )        Define Shift/rotate instructions.
        CREATE C,                    Store opcode.
        DOES>        ( reg --- , or reg CL --- )
        C@                           Stored opcode.
        OVER CL =                    Top register is CL?
        IF                           Multiple bit shift.
                NIP                  Discard CL because it is implied.
                322                  Number of bits shifted in CL.
        ELSE                         Single bit shift.
                320
        THEN
        WR/SM,                       Assemble the two-byte instruction.
        ;

HEX   10 10MI RCL   18 10MI RCR   00 10MI ROL   8 10MI ROR
        38 10MI SAR   20 10MI SHL   28 10MI SHR   OCTAL


: 11MI ( opcode1 opcode2 --- )       Define call/jump instructions.
        CREATE                       Header.
        C,                           Indirect call/jmp opcode.
```

```
        C,                          Direct call/jmp opcode.
        DOES>       ( addr --- )
        OVER #) =                   Immediate address?
        IF                          Yes.
                NIP                 Discard #) mode operator.
                C@                  Get the opcode.
                INTER @ IF          If it is intersegment addressing,
                        1 AND       and a jump?
                        IF 352      Yes.  Jump opcode.
                        ELSE 232 THEN       No.  Call opcode.
                        C,                  Compile jmp/call opcode.
                        SWAP , ,            Compile offset and segment.
                ELSE                        Not intersegment addressing.
                        SWAP                Target address addr.
                        HERE - 2-           Displacement.
                        SWAP        ( disp opcode --- )
                        2DUP 1 AND          Is it JMP?
                        SWAP BIG? NOT AND       And disp<256?
                        IF                  If so, assemble short jump.
                                2 OP,       Short jump opcode.
                                C,          Byte displacement.
                        ELSE                Long jump or call.
                                C,          Opcode.
                                1-          Offset for three-byte instruction.
                                ,           Long displacement.
                        THEN
                THEN
        ELSE                        Not immediate addressing.
                DUP S#) =           Is it intrasegment addressing?
                IF DROP #) THEN     Yes.  Restore the immediate address code.
                377 C,              Assemble opcode.
                1+ C@               Get the initial r/m mode code.
                ?FAR                Add the intersegment far bit if necessary.
                R/M                 Append it to the opcode.
        THEN ;

HEX   10 EB 11MI CALL   20 E9 11MI JMP   OCTAL

: 12MI ( reg-op seg-op r/m-op --- )  Define push and pop instructions.
        CREATE                      Header.
        C, C, C,                    Store three different opcodes for push or pop.
        DOES>       ( reg --- )
        OVER REG?                   Register mode?
        IF                          Yes.
                C@                  Register mode opcode.
                SWAP RLOW OP,       Assemble it.
        ELSE
                1+                  Point to the second opcode.
```

```
            OVER SEG?           Segment register mode?
            IF                  Yes.
                C@              Get segment opcode.
                RLOW            Save only r/m field.
                SWAP RMID       Put in the reg field.
                OP,             Assemble.
            ELSE
                COUNT           Get second opcode and point to the third opcode.
                SWAP C@         Get the third opcode.
                C,              Assemble the third opcode as the first byte of instruction.
                MEM,            Assemble the addressing mode, second byte of instruction.
        THEN
        THEN ;

HEX   8F 07 58 12MI POP   FF 36 50 12MI PUSH   OCTAL

: 13MI ( op1 op2 --- )          Define arithmetic and logic instructions.
        CREATE                  Make header.
        C, C,                   Store opcodes.
        DOES>                   ( operand1 operand2 --- )
        COUNT >R                Fetch and store opcode1.
        C@ LOGICAL !            Save opcode2 in LOGICAL.
        DUP REG?                Is operand2 a register?
        IF                      Yes.
            OVER REG?           Is operand1 also a register?
            IF                  Yes.  A reg-reg math/logic operation.
                R>                          Get opcode1.
                OVER W,                     Assemble opcode1 with w field.
                SWAP RR,                    Assemble addressing byte.
            ELSE                            Operand1 is not a register.
                OVER DUP MEM?               Memory reference?
                SWAP #) = OR                Or memory indirect?
                IF                          Yes.
                    R> 2 OR                 Assemble opcode1 with direction field.
                    WMEM,                   Memory reference.
                ELSE                        Not memory referencing.
                    NIP                     Discard operand1.
                    DUP RLOW 0=             Is operand2 the accumulator?
                    IF                      Yes.
                        R> 4 OR             One byte math instruction.  Fill the math field
                                            (bit 2).
                        OVER W,             Fill in the w field.
                        R16? ,/C,           Assemble the byte or word immediate value.
                ELSE                        Operand2 is not the accumulator.
                        OVER B/L?           Big and long?
                        OVER R16?           Operand2 a 16 bit register?
                        2DUP AND            True for 16 bit logic instruction.
                        -ROT                Save the flag.
```

```
                                        1 AND           W field.
                                        SWAP            16 bit-logic flag.
                                        NOT 2 AND       Sign extension field.
                                        OR              Combine s and w fields.
                                        200 OP,         Assemble first byte opcode.
                                        SWAP RLOW           r/m field.
                                        300 OR          Mode 3.
                                        R> OP,          Second byte mode instruction.
                                        ,/C,            Third byte or word value.
                                THEN
                        THEN
                THEN
        ELSE                            Operand2 is not a register.
                ROT DUP REG?            Is operand1 a register?
                IF R> WMEM,             Yes.  Assemble memory referencing math instruction.
                ELSE                    Not memory referencing.  Must be immediate value.
                        DROP            It is not a register. Discard it because it is a memory code.
                2 PICK                  Pick the displacement.
                B/L?                    Larger than 255?
                DUP NOT 2 AND           Fill in the s field.
                200 OR SIZE,            Assemble first instruction with w field.
                -ROT                    Save the BIG? flag.
                R> MEM,                 Assemble the mode byte.
                SIZE†@                  Must be BIG and word size.
                AND ,/C,                Assemble the immediate value.
                SIZE ON                 Reinitialize SIZE to 16 bit.
                THEN
        THEN ;

HEX   0 10 13MI ADC   0 00 13MI ADD   2 20 13MI AND
        0 38 13MI CMP   2 08 13MI OR    0 18 13MI SBB
        0 28 13MI SUB   2 30 13MI XOR   OCTAL

: 14MI                  ( --- )         Returns.
        CREATE C,                       Compile the opcode.
        DOES>
        C@                              Get opcode.
        DUP ?FAR                        Add the intersegment bit if necessary.
        C,                              Assembler opcode.
        1 AND 0=                        If it has immediate offset,
        IF , THEN                       Assemble the address offset.
        ;

HEX   C3 14MI RET   C2 14MI +RET   OCTAL
```

**24.5. SPECIAL OPCODES**

A small number of instructions do not belong to any of the above types. They are defined individually as colon definitions which have to do special assembly work to assemble their respective machine instructions.

```
HEX

: ESC           ( source opcode --- )   Escape to external device.
        RLOW                            Retain only the low register field.
        0DB OP,                         Assemble the ESC opcode.
        R/M,                            With the r/m code.
        ;

: INT                   ( n --- )       Assemble an interrupt instruction.
        0CD C,                          INT, interrupt instruction.
        C,                              n, the interrupt vector number.
        ;

: SEG                   ( seg --- )     Assemble a segment instruction.
        RMID                            Mask over the segment field.
        26 OP,                          Opcode for segment instruction.
        ;

: XCHG          ( mr1 mr2 --- )         Assemble register exchange instruction.
        DUP REG?                        mr2 a register?
        IF DUP AX =                     And the AX register?
                IF                      mr2=AX.
                        DROP            AX is implied.
                        RLOW 90 OP,             Assemble opcode 90 with mr1.
                ELSE                    m2 is not AX.
                        OVER AX =       Is m1 AX?
                        IF                              m1=AX.
                                NIP                     No need of m1 anymore.
                                RLOW 90 OP,             Assemble XCHG with m2.
                        ELSE                            Neither is AX.
                                86 WR/SM,               Assemble XCHG with a mode byte.
                        THEN
                THEN
        ELSE                            mr2 is not a register.
                ROT 86 WR/SM,           Assemble XCHG with mode byte.
        THEN ;

: CS:  CS SEG ;                 Code segment override.
: DS:  DS SEG :                 Data segment override.
: ES:  ES SEG ;                 Extra segment override.
: SS:  SS SEG ;                 Stack segment override.

: MOV ( source dest --- )               Assemble a MOV instruction, the most complicated
                                        instruction in 8086.
```

```
DUP SEG?                    Is dest a segment register?
IF 8E C,                    Assemble segment MOV,
        R/M,                and the mode byte with source.
ELSE DIP REG?               Is dest a register?
        IF                  Dest is a register.
                OVER #) =       Source is from memory?
                OVER RLOW 0= AND        And dest is AX?
                IF A0 SWAP W,           Yes. Assemble mem to AX MOV,
                        DROP            discard dest, and assemble memory address.
                ELSE OVER SEG?          Is source a segment register?
                        IF              Yes.
                                SWAP 8C C,      Assemble segment to r/m MOV,
                                RR,             with the mode byte.
                        ELSE            Source and dest are not segment register.
                                OVER # =        Immediate source?
                                IF NIP          Yes.  Discard # code.
                                        DUP
                                        R16?            Is dest 16 bit?
                                        SWAP
                                        RLOW            reg field of dest.
                                        OVER 8 AND OR   Combine reg field and w field.
                                        B0 OP,          Assemble immediate to reg MOV,
                                        ,/C,            with the immediate value.
                                ELSE                    Not immediate source.
                                        8AOVER W,       Assemble segment to r/m MOV,
                                R/M,                    with a mode byte.
                        THEN
                THEN
        THEN
ELSE                        Dest is not a register. Treat it as memory reference.
        ROT DUP SEG?        Is source a segment register?
        IF 8C C,            Yes.  Assemble segment to memory MOV,
                MEM,        with memory reference mode byte.
        ELSE DUP # =        Immediate source?
                IF DROP                 Yes.  Discard immediate code.
                        C6 SIZE,        Assemble immediate to reg/mem MOV,
                        0 MEM,          with a mode byte having 0 reg field,
                        SIZE @ ,/C,     and the immediate value.
                ELSE OVER #) =          s dest a memory reference?
                        OVER RLOW 0= AND        And the source is AX?
                        IF A2 SWAP W,           Assemble AX to memory MOV.
                                DROP            Memory code #).
                                ,               Memory address.
                        ELSE                    Non of above.  Must be register to re/m MOV.
                                88 OVER W,      Assemble reg-r/m MOV instruction,
                                R/M,            with the mode byte.
                        THEN
                THEN
```

```
        THEN
        THEN
    THEN
    SIZE ON                 Default size is 16 bit words.
    ;

: TEST( source dest --- )   Assemble TEST instruction which AND source with dest
                            and set the status register.
    DUP REG?                Is destination a register?
    IF OVER REG?            And is source also a register?
        IF                  Both operands are registers.
            204 OVER W,         Assemble opcode.
            SWAP RR,            Assemble reg to reg mode byte.
        ELSE                    Source is not a register.
            OVER DUP MEM?       Is source a memory reference
            SWAP #) = OR        or an immediate address?
            IF 204 WMEM,        Assemble memory to register code and mode byte.
        ELSE                    Immediate data.
                NIP DUP
                RLOW 0=         Is the source AL register?
                IF 250          Yes.  Code for AL reg and immediate data  mode.
                    SWAP W, C,          Assemble code and the immediate byte.
                ELSE                    Memory-immediate data mode.
                    366 OVER W,         Assemble code 366 with the word field.
                    DUP RLOW 300 OP,    Assemble immdiate byte value.
                    R16? ,/C,           If 16 bit data, assemble high byte.
                THEN
            THEN
        THEN
    ELSE                    Destination is not a register.
        ROT UP REG?         Is source a register?
        IF 204 WMEM,        Yes.  Assemble reg-mem TEST instruction.
        ELSE                    Immediate value.
            DROP 366 SIZE,      Immediate data and reg/mem mode.
            0 MEM,              Memory reference.
            SIZE @ ,/C,         If 2 bytes operand, assemble the second byte.
            SIZE ON             Activate 16 word mode.
        THEN
    THEN ;
```

### 24.6. STRUCTURES IN CODE DEFINITIONS

Structures similar to those in the regular colon definitions can also be assembled in code definitions. However, the structures in code definitions are constructed using the branching and looping machine instructions. The test condition for branching are not a flag on top of the data stack but the condition flags kept in the CPU status register.

Forward and backward branching are constructed using some words similar to the MARK and

RESOLVE words in the colon definition.

```
: A?>MARK          ( --- f addr )   Set up a forward branch in code definition.
        TRUE                        Leave a flag on stack for error checking.
        HERE                        Address to branch from.
        0 C,                        A dummy byte later to be resolved to a branching offset.
        ;

: A?>RESOLVE       ( f addr --- )   Resolve a forward branching.
        HERE OVER 1+ -              Calculate the branching offset.
        SWAP C!                     Store it after the branch instruction.
        ?CONDITION                  Abort if the flag is not true.
        ;

: A?<MARK          ( --- f addr )   Set up a backward branch in code definition.
        TRUE                        Set the flag.
        HERE                        Leave current address on stack.
        ;


: A?<RESOLVE       ( f addr --- )   Resolve a backward branch.
        HERE 1+ -                   Backward branch offset.
        C,                          Assemble the offset. Complete the branching instruction.    ?
        CONDITION                   Abort if flag is not true.
        ;
```

The branching instructions are vectored through the words >MARK, >RESOLVE, <MARK, and <RESOLVE. The execution routines vectored by these words can now be resolved by pointing them to the above structure words just defined in the assembler.

```
' A?>MARK   ASSEMBLER IS ?>MARK
' A?>RESOLVE   ASSEMBLER IS ?>RESOLVE
' A?<MARK   ASSEMBLER IS ?<MARK
' A?<RESOLVE   ASSEMBLER IS ?<RESOLVE
```

Conditionals in the assembler are machine codes to be assembled by the structure commands like IF, UNTIL, and WHILE. The conditionals are defined as constants to be assembled:

```
HEX
75 CONSTANT 0=    74 CONSTANT 0<>    79 CONSTANT 0<     78 CONSTANT 0>=
7D CONSTANT <     7C CONSTANT >=     7F CONSTANT <=     7E CONSTANT >
73 CONSTANT U<    72 CONSTANT U>=    77 CONSTANT U<=    78 CONSTANT U>
71 CONSTANT OV
DECIMAL

: IF      ( opcode --- f addr )     Assemble a conditional branch instruction to start a forward
                                    branch.
```

```
        C,                          Assemble conditional opcode.
        ?>MARK                      Set up forward branch.
        ;

: THEN          ( f addr --- )      Close a conditional branch.
        ?>RESOLVE  ;

: ELSE( f1 addr1 --- f2 addr2 )     Resolve forward branch for IF and set up another
                                    forward branch to THEN.
        0EB                         Unconditional branch opcode.
        IF                          Assemble it here.
        2SWAP THEN                  Resolve forward branch from IF.
        ;

: BEGIN         ( --- f addr )      Set up a backward branch.
        ?<MARK       ;

: UNTIL         ( f addr opcode --- )   Resolve the backward branch to BEGIN.
        C,                          Assemble the conditional opcode.
        ?<RESOLVE                   Resolve the branch offset.
        ;

: AGAIN         ( f addr opcode --- )   Resolve the backward branch with an unconditional branch
                                    instruction.
        0EB                         Unconditional branch.
        UNTIL                       Let UNTIL do the resolving and assembling.
        ;

: WHILE         ( --- f addr )      Forward branch.
        IF       ;

: REPEAT        ( f1 addr1 f2 addr2 --- )       Branch back unconditionally.
        2SWAP                       Get the BEGIN location.
        AGAIN                       Assemble unconditional branch to BEGIN.
        THEN                        Resolve WHILE clause.
        ;

: DO            ( n --- addr )      Set up an assembler do-loop.
        # CX MOV                    Assemble an instruction setting up the loop counter.
        HERE                        Leave address for branch instructions.
        ;

: NEXT          ( --- )             The inner interpreter.
        >NEXT #) JMP                Assemble an indirect jump.
        ;

DECIMAL
```