# CHAPTER 10. PASM ASSEMBLER

## 10.1. ASSEMBLER SOURCE CODE

PASM is an assembler which is based on the 8086 assembler published in Dr. Dobb's Journal, February 1982, by Ray Duncan. This assembler was subsequently modified by Robert L. Smith to repair bugs, and support the prefix assembler notation. Bob discovered a very simple method to let a postfix assembler to assemble prefix code, by deferring assembly until the next assembler command or the end of line, when all the arguments for the previous assembler command are piled on the top of the data stack.

```
:VOC SYS
:VOC ASSEMBLER
0 :VAR DPL
HERE DPL !
DEFS DECIMAL
\ The ASSEMBLER follows:
ONLY ROOT
 ALSO
ASSEMBLER DEFS

0 :VAR PRIOR 2 ALLOT
: KICKOFF DROP ;
: DOOLD   R> PRIOR 2@  2SWAP PRIOR 2! >R ;
ROOT DEFS
: CODE    ;[ ASSEMBLER ]: 0 LIT KICKOFF 3 + PRIOR 2!
              ALSO -IMM ASSEMBLER HEAD, ;
ALSO ASSEMBLER DEFS
: END-CODE     PREVIOUS PRIOR 2@ >R ;
\ 8088 Assembler, based on Ray Duncan's Dr. Dobb's article.
: ERROR3  ."  Illegal Operand " (B COLD ;
0 :VAR <#>   0 :VAR <TD>   0 :VAR <TS>   0 :VAR <RD>   0 :VAR <RS>
0 :VAR <W>   0 :VAR <WD>   0 :VAR <OD>   0 :VAR <OS>   0 :VAR <D>
0 :VAR <SP>  0 :VAR <FR>   0 :VAR <AO>
: !SP     SP@ <SP> ! ;
: ?<SP>   <SP> @ SP@ - 2- 2/ ;
\ Destination Register processing.
: DREG    :BUILD C, C, C,
          ;: DUP C@ DUP #0FF = ?[ DROP ][ DUP <W> !  <WD> ! ]?
          1+ DUP C@ <TD> !  1+ C@ <RD> !  <#> @
          ?[ ."  Immediate Data not allowed " (B COLD ]?
          <TD> @ 4 = ?[ ?<SP> 0 > ?[ <OD> ! ]? ]? ;
\ Source Register processing.
: SREG    :BUILD C, C, C,
          ;: DUP C@ DUP #0FF = ?[ DROP ][ <W> ! ]?  1+ DUP C@ <TS> !
          1+ C@ <RS> !  <TS> @ 4 = ?[ ?<SP> 0 > ?[ <OS> ! ]? ]? ;
\ Source Register Definitions
\    Reg  Type W        Name          Reg   Type  W        Name
     0    2    0  SREG  AL            0     3     1  SREG  AX
     1    2    0  SREG  CL            1     3     1  SREG  CX
     2    2    0  SREG  DL            2     3     1  SREG  DX
     3    2    0  SREG  BL            3     3     1  SREG  BX
     4    2    0  SREG  AH            4     3     1  SREG  SP
     5    2    0  SREG  CH            5     3     1  SREG  BP
     6    2    0  SREG  DH            6     3     1  SREG  SI
     7    2    0  SREG  BH            7     3     1  SREG  DI
```

```
     0    4    -1 SREG  [BX+SI]        0    4    -1 SREG  [SI+BX]
     1    4    -1 SREG  [BX+DI]        1    4    -1 SREG  [DI+BX]
     2    4    -1 SREG  [BP+SI]        2    4    -1 SREG  [SI+BP]
     3    4    -1 SREG  [BP+DI]        3    4    -1 SREG  [DI+BP]
     4    4    -1 SREG  [SI]           5    4    -1 SREG  [DI]
     6    4    -1 SREG  [BP]           7    4    -1 SREG  [BX]

     0    5    -1 SREG  ES             1    5    -1 SREG  CS
     2    5    -1 SREG  SS             3    5    -1 SREG  DS

\ Destination Register Definitions

     0    5    -1 DREG  ES,            1    5    -1 DREG  CS,
     2    5    -1 DREG  SS,            3    5    -1 DREG  DS,

     0    2    0  DREG  AL,            0    3    1  DREG  AX,
     1    2    0  DREG  CL,            1    3    1  DREG  CX,
     2    2    0  DREG  DL,            2    3    1  DREG  DX,
     3    2    0  DREG  BL,            3    3    1  DREG  BX,
     4    2    0  DREG  AH,            4    3    1  DREG  SP,
     5    2    0  DREG  CH,            5    3    1  DREG  BP,
     6    2    0  DREG  DH,            6    3    1  DREG  SI,
     7    2    0  DREG  BH,            7    3    1  DREG  DI,

     0    4    -1 DREG  [BX+SI],       0    4    -1 DREG  [SI+BX],
     1    4    -1 DREG  [BX+DI],       1    4    -1 DREG  [DI+BX],
     2    4    -1 DREG  [BP+SI],       2    4    -1 DREG  [SI+BP],
     3    4    -1 DREG  [BP+DI],       3    4    -1 DREG  [DI+BP],
     4    4    -1 DREG  [SI],          5    4    -1 DREG  [DI],
     6    4    -1 DREG  [BP],          7    4    -1 DREG  [BX],

\ Miscellaneous Operators
: TS@     <TS> @ ;
: TD@     <TD> @ ;
: RD@     <RD> @ ;
: RS@     <RS> @ ;
: +D      <D> @ 2* + ;
: +W      <W> @ + ;
: +RD     <RD> @ + ;
: +RS     <RS> @ + ;
: MOD1    #3F AND #40 OR ;
: MOD2    #3F AND #80 OR ;
: MOD3    #3F AND #C0 OR ;
: RS0    <RS> @ 8 * ;
: RSD    RS0 +RD ;
: MD,    RS0 6 + C, ;
: MS,    RD@ 8 * 6 + C, ;
: RDS    RD@ 8 * +RS ;
: CXD,   C@ MOD3 +RD C, ;
: CXS,   C@ MOD3 +RS C, ;
\ Equates to Addressing Modes
0 :CON DIRECT       1 :CON IMMED     2 :CON REG8
3 :CON REG16        4 :CON INDEXED   5 :CON SEGREG
\ Initialize all variables and flags
: RESET   0 <#> !   0 <W> !   0 <OS> !  0 <RD> !
          0 <TD> !  0 <TS> !  0 <OD> !  0 <SP> !
          0 <D> !   0 <WD> !  0 <RS> !  0 <FR> ! ;
```

```
: REG?      REG8 OVER = SWAP REG16 = OR ;
: DREG?    TD@ REG? ;
: ADREG?   DREG? RD@ 3 AND 0= AND ;
: ASREG?   TS@ REG? RS@ 3 AND 0= AND ;
: SUBREG   C@ #38 AND ;
: +S,      <AO> @
               ?[ OVER #80 + #81 U< ?[ 2 OR C, C, ][ C, , ]?
               ][ C, , ]? ;
\ Init. Direction Pointer
: DSET     TS@ DUP INDEXED = SWAP DIRECT = OR NEG <D> ! ;
: DT       1 <D> ! ;     \ Set Direction Flag True.
: OFFSET8,      HERE 1+ - DUP ABS OVER 0< + #7F >
               ?[ ." Address out of range " (B COLD ]? C, ;
: OFFSET16,     HERE 2+ - ;
\ Calculate and store displacement for MEM/REG Instructions.
: DISP,    <D> @ ?[ <OS> ][ <OD> ]? @ DUP
                ?[ DUP ABS #7F > ?[ SWAP MOD2 C, , ][ SWAP MOD1 C, C, ]?
                ][ DROP DUP 7 AND 6 = ?[ MOD1 C, 0 ]? C, ]? ;
\ Calculate the M/R 2nd operator byte
: M/RS,    #38 AND TS@
           [[ DIRECT  =?[ 6 + C, ,                          ][
              REG8    =?[ #C0 + +RS C,                      ][
              REG16   =?[ #C0 + +RS C,                      ][
              INDEXED =?[ <OS> @ #80 + #81 U<
                          ?[ #40 + +RS C, <OS> @ C,
                          ][ #80 + +RS C, <OS> @ ,  ]?  ][
              ERROR3                                        ]]?
           RESET ;
: M/RD,    TD@ <TS> !  <RS> @ <RD> !  <OS> @ <OD> !  M/RS, ;
: 8/16,    <W> @ ?[ , ][ C, ]? ;
\ Words to build the instructions:
: 1MI      :BUILD C, ;: DOOLD C@ C, RESET ;          \ Single Byte Inst.
: 2MI      :BUILD C, ;: DOOLD C@ C, OFFSET8, RESET ; \ Cond Jumps, Loops
: 3MI      :BUILD C, ;: C@ C, ;                      \ Segment Over-ride
: 4MI      :BUILD C, C, C, ;: DOOLD TS@              \ Reg. Push and Pop
           [[ SEGREG =?[ C@ RS@ 8 * + C,       ][
              REG16  =?[ 1+ C@ +RS C,          ][
              REG8   =?[ ERROR3                ][
              DROP 2+ C@ DUP C, 30 AND M/RS,   ]]?
           RESET ;
: 5MI      :BUILD C, C, ;: DOOLD TS@                 \ Iseg. Jump, Call
           [[ DIRECT  =?[ 1+ C@ C, OFFSET16,        ][
              REG16   =?[ #FF C, CXS,               ][
              INDEXED =?[ DSET #FF C, C@ +RS DISP,  ][
              ERROR3                                ]]?
           RESET ;
: 6MI      :BUILD C, C, ;: DOOLD DUP C@ 2 AND        \ IN and OUT
           ?[ TD@ TS@ ][ TS@ TD@ ]?
           <W> @ ?[ 1+ C@ +W C, ][ C@ +W C, C, ]? RESET ;
\ ADC, ADD, AND, etc.
: 7MI      :BUILD C, C, C, ;: DOOLD DUP 1+ C@ 1 AND <AO> !
           TS@ IMMED =
           ?[ ADREG?
                ?[ 2+ C@ +W C, RD@ REG8 ?[ C, ][ , ]?
                ][ DUP 1+ C@ #FE AND +W ROT <OS> !  <AO> @
                     ?[ <OS> @ #80 + #81 U<
                          ?[ 2 OR C, #38 AND M/RS, <OS> @ C,
                          ][ C, #38 AND M/RS, <OS> @ ,
```

```
                    ]?
                ][ C, #38 AND M/RS, <OS> @ ,
                ]?
            ]?
        ][ C@ TS@ REG?
            ?[ +W C, RS@ 8 * M/RD,
            ][ #84 OVER - ?[ 2 OR ]? +W C, TD@ REG?
                ?[ RD@ 8 * M/RS, ][ ERROR3 ]?
            ]?
        ]? RESET ;
: 8MI   :BUILD C, C, ;: DOOLD DUP 1+ C@ +W C, C@ M/RS, RESET ;
: 9MI   :BUILD C, C, ;: DOOLD DUP 1+ C@ <WD> @ +
        TS@ 1 > ?[ 2+ C, ][ C, NIP ]? C,  C@ M/RD, RESET ;
: 10MI  :BUILD C, C, ;: DOOLD DUP 1+ C@ C, C@ C, RESET ;
: 11MI  :BUILD C, C, ;: DOOLD TS@ REG?
            ?[ C@ +RS C, ][ 1+ C@ #FE +W C, M/RS, ]? RESET ;
: 12MI  :BUILD ;: DOOLD DROP                        \ MOV Instruction
        [[ TD@ SEGREG = ?[ #8E C,  RD@ 8 * M/RS,    ][
          TS@ SEGREG = ?[ #8C C,  RS@ 8 * M/RD,     ][
          TS@ IMMED = TD@ REG? AND
              ?[ #16 +W 8 * +RD C, 8/16,            ][
          TS@ 0= TD@ 0= OR ADREG? ASREG? OR AND
              ?[ #A0 +W TS@ ?[ 2+ ]? C, 8/16,       ][
          TS@ IMMED = ?[ #C6 +W C, 0 M/RS, 8/16,    ][
          #88 +W TD@ REG?
                    ?[ 2+ C, RD@ 8 * M/RS,          ][
          TS@ REG? ?[ C, RS@ 8 * M/RD, ][ ERROR3    ]]?
        RESET ;
: 13MI  :BUILD ;: DOOLD DROP #86 +W                 \ XCHG Instruction
        TS@ REG? 0=
            ?[ TD@ REG? 0=
                ?[ ERROR3 ][ RD@ 8 * M/RS, ]?
            ][ RS@ 8 * M/RD,
            ]? RESET ;
: 14MI  :BUILD C, ;: DOOLD C@ C, TD@ REG?
            ?[ RD@ 8 * M/RS, ][ ERROR3 ]? RESET ;
: 15MI  :BUILD ;: DOOLD DROP DUP 3 =
        ?[ DROP #CC C, ][ #CD C, C, ]? RESET ;
\ Now let's create the actual instructions.
HEX

37          1MI   AAA      FC          1MI   CLD
D5 0A      10MI   AAD      FA          1MI   CLI
D4 0A      10MI   AAM      F5          1MI   CMC
3F          1MI   AAS      3C 81 38    7MI   CMP
14 81 10    7MI   ADC      A6          1MI   CMPSB
04 81 00    7MI   ADD      A7          1MI   CMPSW
24 80 20    7MI   AND      99          1MI   CWD
E8 10       5MI   CALL     27          1MI   DAA
98          1MI   CBW      2F          1MI   DAS
F8          1MI   CLC      08 48      11MI   DEC

F6 30       8MI   DIV      73          2MI   JAE
F4          1MI   HLT      72          2MI   JB
F6 38       8MI   IDIV     76          2MI   JBE
F6 28       8MI   IMUL     76          2MI   JC
EC E4       6MI   IN       E3          2MI   JCXY
00 40      11MI   INC      74          2MI   JE
```

```
          15MI   INT      7F         2MI   JG
CE         1MI   INTO     7D         2MI   JGE
CF         1MI   IRET     7C         2MI   JL
77         2MI   JA       7E         2MI   JLE

E9 20      5MI   JMP      7F         2MI   JNLE
76         2MI   JNA      71         2MI   JNO
72         2MI   JNAE     7B         2MI   JNP
73         2MI   JNB      79         2MI   JNS
77         2MI   JNBE     75         2MI   JNZ
73         2MI   JNC      70         2MI   JO
75         2MI   JNE      7A         2MI   JP
7E         2MI   JNG      7A         2MI   JPE
7C         2MI   JNGE     7B         2MI   JPO
7D         2MI   JNL      78         2MI   JS

74         2MI   JZ       E0         2MI   LOOPNE
9F         1MI   LAHF     E0         2MI   LOOPNZ
C5        14MI   LDS      E1         2MI   LOOPZ
8D        14MI   LEA               12MI   MOV
C4        14MI   LES      A4         1MI   MOVSB
F0         1MI   LOCK     A5         1MI   MOVSW
AC         1MI   LODSB    F6 20      8MI   MUL
AD         1MI   LODSW    F6 18      8MI   NEG
E2         2MI   LOOP     90         1MI   NOP
E1         2MI   LOOPE    F6 10      8MI   NOT

0C 80 08   7MI   OR       F2         1MI   REPNE
EE 08      6MI   OUT      F2         1MI   REPNZ
8F 58 07   4MI   POP      F3         1MI   REPZ
9D         1MI   POPF     C3         1MI   RET
FF 50 06   4MI   PUSH     D0 00      9MI   ROL
9C         1MI   PUSHF    D0 03      9MI   ROR
D0 10      9MI   RCL      9E         1MI   SAHF
D0 18      9MI   RCR      D0 38      9MI   SAR
F3         1MI   REP      1C 81 18   7MI   SBB
F3         1MI   REPE     AE         1MI   SCASB

AF         1MI   SCASW    AB         1MI   STOSW
D0 20      9MI   SAL      2C 81 28   7MI   SUB
D0 20      9MI   SHL      A4 F6 84   7MI   TEST
D0 28      9MI   SHR      9B         1MI   WAIT
F9         1MI   STC               13MI   XCHG
FD         1MI   STD      D7         1MI   XLAT
FB         1MI   STI      34 80 30   7MI   XOR
AA         1MI   STOSB    \                ESC

\ Segment over-ride commands:
26         3MI   ES:
2E         3MI   CS:
36         3MI   SS:
3E         3MI   DS:

: FAR      1 <FR> ! ;
: BYTE     0 <W> !   0 <WD> ! ;
: WORD     1 <W> !   1 <WD> ! ;
: #        1 <W> !   1 <TS> ! ;
: ,        DEPTH 1 < ?[ ." Destination address missing " (B ]? ;
```

```
: []        0 <W> ! ;
: 3*        DUP 2* + ;
:BUILD LFWD $24 ALLOT
: L         7 AND 3* LFWD + DUP C@
            [[ 0 =?[ 1 OVER C! HERE 1+ SWAP 1+ ! HERE 2+
            ][ 1 =?[ 1+ DUP @ SWAP HERE 2+ SWAP !
            ][ 2 =?[ 1+ @
            ][ ."  Bad Reference " (B COLD ]]? ;
: -C@       C@ #7F OVER > ?[ #FF00 OR ]? ;
: CRESOLVE  DUP 1+ @
                [[ 2DUP - SWAP 1- -C@ DUP >R OVER + -ROT 1- C! R> ?] ;
: L:        >R DOOLD R> 7 AND 3* LFWD + DUP C@
            [[ 0 =?[ 2 OVER C!  1+ HERE SWAP !
            ][ 1 =?[ CRESOLVE 2 OVER C! 1+ HERE SWAP !
            ][ ."  Bad Label " (B COLD ]]? ;
: NEXT      ( -- )   LODSW   JMP AX  ;
ONLY ROOT DEFS
DECIMAL
ONLY FORTH ALSO DEFS

: OK  CR ."  HI " ;
OK
```

## 10.2. EXAMPLES OF PASM PROGRAMMING

PASM uses prefix assembly words to assemble 8086 machine code words to run under LaForth. The general syntax of a code definition is :

```
CODE GIZZMO         ( n1 -- n2 )
            POP AX
            MOV BX, AX
            ...
            ...
            PUSH AX
            NEXT
END-CODE
```

CODE initiates a low level code definition, and invokes the ASSEMBLER vocabulary for all the assembly mnemonic words. Assembly instructions are then given in a form very close to the regular Intel-Microsoft MASM format. NEXT is required at the end of a code definition to pass control to the next Forth word. END-CODE is also required as the last instruction in a code definition, which terminates the assembly process.

It is impossible to discuss PASM in details. The best we can do is to include a list of valid PASM assembly code, from which an experience assembly programmer can infer other instructions he will be using.

**Comparison of assembly syntax**

| PREFIX | POSTFIX | MASM |
|---|---|---|
| AAA | AAA | AAA |
| ADC AX, SI | SI AX ADC | ADC AX,SI |
| ADC DX, 0 [SI] | 0 [SI] DX ADC | ADC DX,0[SI] |
| ADC 2 [BX+SI], DI | DI 2 [BX+SI] ADC | ADC 2[BX][SI],DI |

| | | |
|---|---|---|
| ADC MEM BX | BX MEM #) ADC | ADC MEM,BX |
| ADC AL, # 5 | 5 # AL ADC | ADC AL,5 |
| AND AX, BX | BX AX AND | AND AX,BX |
| AND CX, MEM | CX MEM #) AND | AND CX,MEM |
| AND DL, # 3 | 3 # DL AND | AND DL,3 |
| CALL NAME | NAME #) CALL | CALL NAME |
| CALL FAR [] NAME | FAR [] NAME #) CALL | ????? |
| CMP DX, BX | BX DX CMP | CMP DX,BX |
| CMP 2 [BP], SI | SI 2 [BP] CMP | CMP [BP+2],SI |
| DEC BP | BP DEC | DEC BP |
| DEC MEM | MEM DEC | DEC MEM |
| DEC 3 [SI] | 3 [SI] DEC | DEC 3[SI] |
| DIV CL | CL DIV | DIV CL |
| DIV MEM | MEM DIV | DIV MEM |
| IN PORT# WORD | WORD PORT# IN | IN AX,PORT# |
| IN PORT# | PORT# IN | IN AL,PORT# |
| IN AX, DX | DX AX IN | IN AX,DX |
| INC MEM | BYTE MEM INC | INC MEM BYTE |
| INC MEM WORD | MEM #) INC | INC MEM WORD |
| INT 16 | 16 INT | INT 16 |
| JA NAME | NAME JA | JA NAME |
| JNBE NAME | NAME #) JNBE | JNBE NAME |
| JMP NAME | NAME #) JMP | JMP |
| JMP FAR [] NAME | NAME [] FAR JMP | JMP [NAME] |
| JMP FAR $F000 $E987 | | JMP F000:E987 |
| LODSW | AX LODS | LODS WORD |
| LODSB | AL LODS | LODS BYTE |
| LOOP NAME | NAME #) LOOP | LOOP NAME |
| MOV DX, NAME | NAME #) DX MOV | MOV DX,[NAME] |
| MOV AX, BX | BX AX MOV | MOV AX,BX |
| MOV AH, AL | AL AH MOV | MOV AH,AL |
| MOV BP, 0 [BX] | 0 [BX] BP MOV | MOV BP,0[BX] |
| MOV ES: BP, SI | ES: BP SI MOV | MOV ES:BP,SI |
| MOVSW | AX MOVS | MOVS WORD |
| POP DX | DX POP | POP DX |
| POPF | POPF | POPF |
| PUSH SI | SI PUSH | PUSH SI |
| REP | REP | REP |
| RET | RET | RET |
| ROL AX, # 1 | AX ROL | ROL AX,1 |
| ROL AX, CL | AX CL ROL | ROL AX,CL |
| SHL AX, # 1 | AX SHL | SHL AX,1 |
| XCHG AX, BP | BP AX XCHG | XCHG AX,BP |
| XOR CX, DX | DX, CX XOR | XOR CX,DX |