```
   let ConstOp[Op, a, b] = valof
   §CO
       switchon Op into
 5     §s
           case ADDsm:  resultis a + b
           case SUBsm:  resultis a - b
           case MULTsm: resultis a × b
           case DIVsm:  resultis a / b
10         case REMsm:  resultis a rem b
           case ORsm:   resultis a ∨ b
           case ANDsm:  resultis a ∧ b
           case EQVsm:  resultis a = b
           case NEQVsm: resultis a ≠ b
15         case LSHsm:  resultis a lshift b
           case RSHsm:  resultis a rshift b
           case EQ:     resultis a = b
           case NE:     resultis a ≠ b
           case LT:     resultis a < b
20         case LE:     resultis a ≤ b
           case GT:     resultis a > b
           case GE:     resultis a ≥ b
           default:     Error[106, HARD, Op, NullProgram]
                        resultis ERROR
25     §s
   §CO


   and Application[Exit] be
30 §A
       let RSSP = SSP
       Inst[LINKsm]
       Addto[ShuntVec, Ch]
       §   Expression[]
35         if PresentOutput = MFAPP ∧ SSP > RSSP + 1 do
               Code[ENDPARAM, RSSP]
       § repeatwhile Ch = COMMA
       unless Ch = SKET do CompilerError[3601]
       ManApp := DUMMY
40     Code[APPLYsm, RSSP]
       if Exit = RTEXIT do
       §   Read[]
           if Ch = SBRA do Exit := FNEXIT
       §
45     SSP := (Exit = RTEXIT ∨ FHN ≠ UNSET) → RSSP - 1, RSSP
       switchon ManApp into
       §s
           case DUMMY:
                   Code[JUMPPTsm, SSP]
50                 endcase

           case MANFN:
                   unless Exit = FNEXIT do
                       Error[270, HARD, DUMMY, NullProgram]
55                 endcase

           case MANRT:
                   unless Exit = RTEXIT do
                       Error[271, HARD, DUMMY, NullProgram]
60                 endcase
```

```
            default:CompilerError[3602]
        $s
    $A
65

    and FindName[C] = valof
    §FN
        let N = NameBlock[C]
70      if NonRecDef do N := NonRecName[N]
        switchon NameType[N] into
        §s
            case SIMPLE:
                    if Generation[N] > LocalGenNo resultis N
75
            case UNDEFINED:
                    unless Ignoring[] do
                        Error[203, HARD, C, NullProgram]

80          case GLOBAL:
            case MANIFEST:
            case STATIC:
            case TABLE:
            case MANFN:
85          case MANRT:
            case LABEL:
                    resultis N

            default:CompilerError[3603]
90      $s
    $FN


    and NonRecName[N] = valof
95  §NRN
        for i = FIRSTEL to NonRecVec↓PTR do
            if Generation[N] = NonRecVec↓i do
            §   if NameType[N] = MANIFEST ∧ NameNo[N] = SET loop
                if (NameType[N] = MANFN ∨ NameType[N] = MANRT)
100                     ∧ (UNDEFINED ≠ NameVal[N] ≠ UNUSED) loop
                resultis NonRecName[PreviousNameBlock[N]]
            §
        resultis N
    $NRN
105

    and Ignoring[] = valof
    §I
        for i = OutputVec↓PTR to FIRSTEL by -1 do
110         switchon OutputVec↓i into
            §s
                case IGNORE: resultis true

                case NORMAL: resultis false
115         $s
        CompilerError[3604]
    $I


120 and CheckScope[F, Line, Get] be
    §CS
        let G = PreviousNameBlock[F]
        let l, g = LineNo, GetNo
        if G = NILNAME return
125     switchon NameType[F] into
        §s
```

```
            case GLOBAL:
            case MANIFEST:
                    if NameType[F] = NameType[G] do
130                     if NameVal[F] = NameVal[G] return

            case LABEL:
            case STATIC:
            case TABLE:
135         case MANFN:
            case MANRT:
                    if Generation[F] = Generation[G] return
                    LineNo, GetNo := Line, Get
                    Error[206, SOFT, NameField[F], NullProgram]
140                 LineNo, GetNo := l, g
                    return

            default:CompilerError[3506]
        $s
145 $CS


    and DeleteCommand[] be
    §DC
150     switchon ChType into
        §s
            case SECTKET:
            case SEMICOLON:
            case OR:
155         case IFSO:
            case IFNOT:
            case REPEAT:
            case REPEATUNTIL:
            case REPEATWHILE:
160         case ENDBODY:
            case ENDFOR:
            case ENDVALOF:
                    return

165         case SECTBRA:
                    DeleteBlock[]

            default:Read[]
        $s repeat
170 §DC


    and DeleteBlock[] be
    §DB
175     let SectKet = SECTKET ∨ ValPart[Ch]
        Read[]
        switchon ChType into
        §s
            case SECTBRA:
180                 DeleteBlock[]
                    Read[]
                    endcase

            case SECTKET:
185                 unless Ch = SectKet do CompilerError[3606]
                    return

            case ENDPROG:
                    CompilerError[3607]
190
            default:Read[]
```

```
        $s repeat
    $DB

195
    and Prec[Left, CType] = valof
    $P
        switchon CType into
        $s
200         case BOTTOM:
            case VALOF:
            case SUBEXP:
                    resultis 0

205         case TO:
            case BY:
            case DO:
            case ASS:
            case OR:
210         case IFSO:
            case IFNOT:
            case INTO:
            case REPEAT:
            case REPEATWHILE:
215         case REPEATUNTIL:
            case SEMICOLON:
            case LET:
            case AND:
            case CCOLON:
220         case SECTBRA:
            case SECTKET:
            case ENDBODY:
            case ENDFOR:
            case ENDVALOF:
225                 resultis 1

            case RBRA:
            case SBRA:
                    resultis Left → 20, 2
230
            case RKET:
            case SKET:
                    resultis 2

235         case COND:
                    resultis Left → 3, 0

            case COMMA:
                    resultis 4
240
            case NEQV:
                    resultis 5

            case EQV:
245                 resultis 6

            case LOGOR:
                    resultis 7

250         case LOGAND:
                    resultis 8

            case NOT:
                    resultis Left → 20, 9
255
            case LSHIFT:
```

```
            case RSHIFT:
                    resultis Left → 10, 13

260         case EQ:
            case GT:
            case LT:
            case NE:
            case GE:
265         case LE:resultis Left → 12, 11

            case PLUS:
            case MINUS:
                    resultis 14
270
            case UPLUS:
            case UMINUS:
                    resultis Left → 20, 15

275         case MULT:
            case DIV:
            case REM:
                    resultis Left → 17, 16

280         case LV:
            case RV:resultis Left → 20, 18

            case VECAP:
            case VECOP:
285                 resultis 19

            default:CompilerError[3608]
        $s
    $P
290

    and ChangeOutput[] be
    §CO
        let n = OutputVec↓PTR
295     PresentOutput := OutputVec↓n
        switchon PresentOutput into
        §s
            case NORMAL:
                    NormalOutput := NormalOut
300                 CellOutput := CellOut
                    return

            case MFAPP:
                    NormalOutput := AppOut
305                 CellOutput := OutputtoMFApp
                    return

            case MFDEF:
                    NormalOutput := DefOut
310                 CellOutput := OutputtoMFDef
                    return

            case CONSTANT:
                    NormalOutput := ConstOut
315                 CellOutput := ConstOut
                    return

            case IGNORE:
                    NormalOutput := NullProgram
320                 CellOutput := NullProgram
                    return
```

```
            default:CompilerError[3609]
        $s
325 $CO


    and NormalOut[n, a, b, c, d] be
    §NO
330     switchon n into
        §s
            case 4: Write[d]
            case 3: Write[c]
            case 2: Write[b]
335         case 1: Write[a]
                    return

            default:CompilerError[3610]
        $s
340 $NO


    and AppOut[n, a, b, c, d] be
        OutputtoMFApp[lv a]
345

    and DefOut[n, a, b, c, d] be
        OutputtoMFDef[lv a]

350
    and CellOut[p] be
    §CO
        test p↓0 = SWITCH
          ifso  §   let n = p↓1 - 5
355                 TransferOut[Output, p + 5, n]
                    for i = 4 to 0 by -1 do Write[p↓i]
                §
          ifnot NormalOut[CellLength[p], p↓0, p↓1, p↓2, p↓3]
    $CO
360

    and ConstOut[] be
    §CO
        Error[205, HARD, DUMMY, NullProgram]
365     NormalOutput := NullProgram
        CellOutput := NullProgram
    $CO


370 and SetUpVec[Size] = valof
    §SUV
        let v = NewVec[Size]
        v↓SIZE := Size
        v↓PTR := FIRSTEL - 1
375     resultis v
    $SUV


    and Addto[Vec, El] be
380 §A
        let n = Vec↓PTR + 1
        unless n ≤ Vec↓SIZE do Error[8, HARD, DUMMY, Finish]
        Vec↓PTR := n
        Vec↓n := El
385 $A
```

```
    and Subtractfrom[Vec] be
    §S
390     let n = Vec↓PTR - 1
        unless n ≥ (FIRSTEL - 1) do CompilerError[3611]
        Vec↓PTR := n
    $S

395
    and CloseDownVec[Vec] be
    §CDV
        unless Vec↓PTR = (FIRSTEL - 1) do CompilerError[3612]
        ReturnVec[Vec, Vec↓SIZE]
400 $CDV


    and SetUp3b[] be
    §SU3b
405     SetUpManFunBuffs[]
        SetUpRA[]
        OutputVec := SetUpVec[DEPTHSIZE]
        NonRecVec := SetUpVec[DEPTHSIZE]
        ShuntVec := SetUpVec[SHUNTSIZE]
410     StartOutputMode[NORMAL]
        Addto[ShuntVec, BOTTOM]
    $SU3b


415 and CloseDown3b[] be
    §CD3b
        CloseManFunBuffs[]
        CloseRA[]
        Subtractfrom[ShuntVec]
420     EndOutputMode[]
        CloseDownVec[OutputVec]
        CloseDownVec[NonRecVec]
        CloseDownVec[ShuntVec]
    $CD3b
425

****
```