```
Pass 4.2
 *** Compilers 31/7/76


   let SetUpCode[] be
   §SC
       SetCodePtrs[]
 5     CodeFull := true
       SNec := false
       ExitNec := false
       HopNext := false
       HopPtNext := false
10     FnExitNext := false
       NextSS := UNSET
       Inst[RTEXITsm]
       LineNo, GetNo := 0, 0
   $SC
15

   and GenerateCode[] be
   §GC
       Ch := NextNecCh[]
20     ForHopCode[Ch]
   $GC repeatuntil Ch = ENDPROG


   and InstCode[Ch] be
25 §IC
       switchon Ch into
       §s
           case Ssm:
                   SMLCode[Ch, Next[In] + 1]
30                 endcase

           case APPLYsm:
                   StartNewWord[]

35         case INCsm:
           case Nsm:
           case RPsm:
           case SPsm:
                   SMLCode[Ch, Next[In]]
40                 endcase

           case LGsm:
           case RGsm:
           case SGsm:
45         case RAsm:
           case SAsm:
           case LPsm:
                   MLCode[Ch, Next[In]]
                   endcase
50
           case JUMPPTsm:
                   JumpPt[]
                   endcase

55         default:unless LengthField[Ch] = 1 do
                       CompilerError[4201]

           case RTEXITsm:
           case FNEXITsm:
60                 Inst[Ch]
                   endcase
```

```
            case SETCSEGGLOBAL:
                    SetCSegGlobal[]
                    endcase

            case SETCSEGSTATIC:
                    SetCSegStatic[]
                    endcase

            case SETMANFUN:
                    SetManFun[]
                    endcase

            case BACKHOP:
                    BackHop[]
                    endcase

            case BACKHOPPT:
                    BackHopPt[]
                    endcase

            case LOADSTATIC:
                    LoadStatic[]
                    endcase

            case LOADMANFUN:
                    LoadManFun[]
                    endcase

            case LOADSTRING:
                    LoadString[]
                    endcase

            case SWITCH:
                    ReadSwitchBlock[]
                    endcase

            case SWITCHsm:
                    Switch[]
                    endcase

            case ENDPROG:
                    StartNewWord[]
                    return
        $s
    $IC


and NextNecCh[] = valof
    §NNC
        §r
            let Ch = Next[In]
            switchon Ch into
            §s
                case RTEXITsm:
                case RETURNsm:
                      § let x = Next[In]  §
                        unless ExitNec endcase
                        ExitNec := false
                        SNec := false
                        resultis RTEXITsm

                case Ssm:
                        unless SNec do
                        §   Ch := Next[In]
```

```
                        endcase
                    §
                    SNec := false
130                 NextSS := Next[In]
                    endcase

            case FNEXITsm:
                  § let x = Next[In]  §
135
            default:SNec := true
                    ExitNec := true

            case FORHOPPT:
            case BACKHOPPT:
                    resultis Ch

            case ENDMFDEF:
                    Ch := Next[In]
145                 if MFLastUse↓Ch = UNSET do SkipMFDef[Ch]
                    endcase

            case ENDPARAM:
                    Ch := Next[In]
150                 endcase
        $s
    $r repeat
$NNC

155
and SkipMFDef[m] be
§SD
    let Type = Next[In]
    switchon Type into
160 §s
        case SETMANFUN:
              § let n = Next[In]
                let Ch = Next[In]
                if m = n return
165             endcase
              §

        case SWITCH:
                for i = 3 to Next[In] do
170             §   let x = Next[In]  §
                endcase

        default:
                for i = 2 to LengthField[Type] do
175             §   let x = Next[In]  §
                endcase

        case ENDPROG:
        case ENDOFSTRCH:
180             CompilerError[4202]
    $s
$SD repeat


185 and ForHopCode[Ch] be
§FHC
    switchon Ch into
    §s
        case FNEXITsm:
190             FnExitNext := true
                HopNext := false
```

```
                HopPtNext := false
                NextSS := UNSET
                InstCode[Ch]
195             return

        case FORHOP:
                HopNo := Next[In]
                NextSS := UNSET
200             SNec := (HopList↓HopNo > 0)
                if (HopPtNo = HopNo) ∧ HopPtNext return
                HopNext := true
                FnExitNext := false
                HopPtNext := false
205             return

        case CONDFORHOP:
                §   let n = Next[In]
                    let CondType = Next[In]
210                 if HopNext ∧ NextSS = UNSET ∧ CodeFull ∧
                            Abs[HopList↓n] = NextCodePos[] do
                    §   HopNext := false
                        n := HopNo
                        CondType := RevCondType[CondType]
215                 §
                    if HopNext do InsertForHop[]
                    SetStackifNec[]
                    ResetPtrs[]
                    MLCode[CondType, HopLength[n]]
220                 return
                §

        case SETLITGLOBAL:
                SetLitGlobal[]
225             return

        case SETLITSTATIC:
                SetLitStatic[]
                return
230
        case SETDSEGGLOBAL:
                SetDSegGlobal[]
                return

235     case SETDSEGSTATIC:
                SetDSegStatic[]
                return

        case SPsm:
240             Ch := Next[In]
                if ((WordafterHop[HopNo] = (FNEXITsm lshift 8) ∧
                        HopNext) ∨ FnExitNext) ∧ Ch = NextSS do
                §   NextSS := UNSET
                    return
245             §
                if HopNext do InsertForHop[]
                SetStackifNec[]
                ResetPtrs[]
                SMLCode[SPsm, Ch]
250             return

        default:if HopNext do InsertForHop[]
                SetStackifNec[]

255     case RTEXITsm:
                ResetPtrs[]
```

```
                InstCode[Ch]
                return

        case COERCETOVAL:
                if HopNext do InsertForHop[]
                SetStackifNec[]
                StartNewWord[]
                SCode[Nsm, 0]
                HopPtNo := Next[In]
                NextSS := Next[In]
                SetStackifNec[]
                StartNewWord[]
                HopList↓HopPtNo := NextCodePos[]
                test HopNext
                  ifso  InsertForHop[]
                  ifnot SCode[HOPsm, 1]
                Inst[TRUEsm]
                ResetPtrs[]
                return

        case FORHOPPT:
                HopPtNo := Next[In]
                HopPtNext := true
                if HopNext ∧ NextSS = UNSET do
                §   HopList↓HopPtNo := HopList↓HopNo
                    return
                $
                if HopNext do InsertForHop[]
                SetStackifNec[]
                StartNewWord[]
                HopList↓HopPtNo := (NextSS = UNSET → NextCodePos[],
                    -NextCodePos[])
                NextSS := UNSET
                HopNext := false
                return
    $s
$FHC


and ResetPtrs[] be
§RP
    HopNext := false
    HopPtNext := false
    FnExitNext := false
    NextSS := UNSET
$RP


and InsertForHop[] be
§IFH
    let l = HopLength[HopNo]
    if l = 0 ∧ CodeFull return
    switchon WordafterHop[HopNo] into
    §s
        case FNEXITsm lshift 8:
                Inst[FNEXITsm]
                return

        case RTEXITsm lshift 8:
                Inst[RTEXITsm]
                return

        default:SMLCode[HOPsm, l]
    $s
$IFH
```

```
    and JumpPt[] be
325 §JP
        let n = Next[In] + 1
        unless 0 ≤ n ≤ 255 do
        §   LCode[Ssm, n]
            n := 0
330     §
        StartNewWord[]
        MCode[JUMPPTsm, n]
    §JP

335
    and SetLitStatic[] be
    §SLS
        let n = Next[In]
        StaticType↓n := LIT
340     StaticVal↓n := Next[In]
    §SLS


    and SetLitGlobal[] be
345 §SLG
        let n = Next[In]
        GlobalType↓n := ACTUALg
        GlobalLoc↓n := Next[In]
        GlobalVal↓n := Next[In]
350 §SLG


    and SetDSegGlobal[] be
    §SDG
355     let n = Next[In]
        GlobalType↓n := DSEGg
        GlobalLoc↓n := Next[In]
        AddTable[Next[In]]
        GlobalVal↓n := SN + TDPtr
360 §SDG


    and SetDSegStatic[] be
    §SDS
365     let n = Next[In]
        StaticType↓n := DSEG
        AddTable[Next[In]]
        StaticVal↓n := SN + TDPtr
    §SDS
370

    and AddTable[n] be
    §AT
        TDPtr := TDPtr - n
375     if TDPtr < 0 do CompilerError[4203]
        for i = TDPtr + n - 1 to TDPtr by -1 do TableData↓i := Next[In]
    §AT


380 and SetCSegStatic[] be
    §SCS
        let n = Next[In]
        StartNewWord[]
        StaticType↓n := CSEG
385     StaticVal↓n := NextCodePos[]
      § let d = DiagNo[]
```

```
        DiagAddr↓d := NextCodePos[]
        DiagGlobNo↓d := -1
        DiagName↓d := ValPart[Next[In]]
390     DiagString↓d := UNSET
    §SCS


    and SetCSegGlobal[] be
395 §SCG
        let n = Next[In]
        StartNewWord[]
        GlobalType↓n := CSEGg
        GlobalVal↓n := NextCodePos[]
400     GlobalLoc↓n := Next[In]
      § let d = DiagNo[]
        DiagAddr↓d := NextCodePos[]
        DiagGlobNo↓d := GlobalLoc↓n
        DiagName↓d := ValPart[Next[In]]
405     DiagString↓d := UNSET
    §SCG


    and SetManFun[] be
410 §SMF
        let n = Next[In]
        StartNewWord[]
        MFVal↓n := NextCodePos[]
        n := Next[In]
415 §SMF


    and LoadManFun[] be
    §LMF
420     let p = CodePos[]
        and n = Next[In]
        LCode[Nsm, MFLastUse↓n]
        MFLastUse↓n := p
        n := Next[In]
425 §LMF


    and LoadStatic[] be
    §LS
430     let Type = Next[In]
        and p = CodePos[]
        let n = Next[In]
        LCode[Type, StaticLastUse↓n]
        StaticLastUse↓n := p
435 §LS


    and LoadString[] be
    §LS
440     let p = CodePos[]
        let n = Next[In]
        LCode[Nsm, StringUse↓n]
        StringUse↓n := p
    §LS
445

    and BackHop[] be
    §BH
        let n = Next[In]
450     HopList↓n := NextCodePos[]
        BackHopList↓n := CodePos[]
```

```
        LCode[HOPsm, DUMMY]
    $BH

455
    and BackHopPt[] be
    §BHP
        let n = Next[In]
        StartNewWord[]
460     UpdateCode[BackHopList↓n, -HopLength[n]]
    $BHP


    and ReadSwitchBlock[] be
465 §RSB
        let n = Next[In]
        let v = ProperVec[n - 2]
        TransferIn[In, v + 1, n - 2]
        SwitchList↓(v↓1) := v
470 $RSB


    and Switch[] be
    §S
475     let v = SwitchList↓(Next[In])
        let h, n = v↓2, v↓0
        StartNewWord[]
        WordtoCode[HopLength[v↓3] + 2]
        for i = 4 to n do
480     §   WordtoCode[v↓i]
            WordtoCode[HopLength[h + n - i] + 2]
        §
        WordtoCode[3 - n]
        Inst[SWITCHsm]
485     ReturnVec[v, n]
    $S


    and StartNewWord[] be
490     unless CodeFull do
        §   DecCodeBodyPtrs[]
            CodeFull := true
            UpdateCode[FirstCode, (FIRSTMEDINST ≤ SpareByte ≤ LASTMEDINST →
                                   SpareByte, SpareByte lshift 8)]
495     §


    and SetStackifNec[] be
        unless NextSS = UNSET do SMLCode[Ssm, NextSS + 1]
500

    and HopLength[n] = (Abs[HopList↓n] - NextCodePos[])


505 and SMLCode[Type, Length] be
        test 0 ≤ Length ≤ 15
          ifso  SCode[Type, Length]
          ifnot MLCode[Type, Length]

510
    and MLCode[Type, Length] be
        test 0 ≤ Length ≤ 255
          ifso  MCode[Type, Length]
          ifnot LCode[Type, Length]
515
```

```
    and SCode[Type, Length] be BytetoCode[ShortVersion[Type] ∨ Length]


520 and MCode[Type, Length] be
    §MC
        BytetoCode[Length]
        BytetoCode[MediumVersion[Type]]
    $MC
525

    and LCode[Type, Length] be
    §LC
        WordtoCode[Length]
530     BytetoCode[LongVersion[Type]]
    $LC


    and ShortVersion[Type] = InstField[Type] lshift 4
535

    and MediumVersion[Type] =
            0 ≤ InstField[Type] ≤ 817 → InstField[Type] ∨ MEDIUM,
                                        InstField[Type + 1]
540

    and LongVersion[Type] =
            0 ≤ InstField[Type] ≤ 817 → InstField[Type] ∨ LONG,
                                        InstField[Type]
545

    and Inst[T] be BytetoCode[InstField[T]]


550 and BytetoCode[W] be
        test CodeFull
          ifso  §   CodeFull := false
                    SpareByte := W
                §
555       ifnot §   DecCodeBodyPtrs[]
                    CodeFull := true
                    UpdateCode[FirstCode, SpareByte ∨ (W lshift 8)]
                §

560
    and WordtoCode[W] be
    §WC
        DecCodeBodyPtrs[]
        UpdateCode[FirstCode, W]
565 $WC


    and DecCodeBodyPtrs[] be
    §DBP
570     FirstCode := FirstCode - 1
        if FirstCode < FirstWord do Error[7, HARD, DUMMY, Finish]
    $DBP

****
```